2023 17th International Conference on Ubiquitous Information Management and Communication (IMCOM) | 978-1-6654-5348-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/IMCOM56909.2023.10035546

Data Acquisition Platform for The Energy Management of Smart Factories and Buildings

Shah M. Mahir
Dept of Electrical and Computer Engineering
Purdue University
West Lafayette, United States
0000-0002-6709-4633

Graham Koch
Dept of Electrical and Computer Engineering
Indiana Univ. Purdue Univ. Indianapolis
Indianapolis, United States
gk9@iu.edu

Jared Herne
Dept of Electrical and Computer Engineering
Indiana Univ. Purdue Univ. Indianapolis
Indianapolis, United States
jherne@iu.edu

John J. Lee
Dept of Electrical and Computer Engineering
Indiana Univ. Purdue Univ. Indianapolis
Indianapolis, United States
0000-0002-5335-9071

***Abstract*—With the rise of energy consumption, especially in the manufacturing sector, energy management systems are being developed to increase the energy efficiency of smart buildings and factories. These energy management systems are able to derive energy consumption parameters and visualize environmental data collected from a data acquisition platform. Internet of Things (IoT) technologies can be used to acquire this data, but will face challenges in industrial settings—lack of cyber secure communication, poor scalability, and minimal back-end functionality are to name a few. This paper presents a data acquisition platform for industrial energy management that combats these IoT challenges. The implemented platform utilizes the Message Queuing Telemetry Transport (MQTT) protocol to establish fast and simple communication between constrained IoT devices. A home automation platform called open Home Automation Bus (openHAB) is utilized as an integration platform due to its balance between functionality and user experience. However, since openHAB does not offer end-to-end payload encryption for MQTT, a method of secure communication to and from openHAB is developed using AES-128 and XChaCha20 algorithms. To improve IoT scalability, a method of dynamic sensor configuration is also developed, which mitigates openHAB configuration time. Finally, though many databases are supported, Influx Database (InfluxDB) is integrated alongside openHAB to collect and analyze sensor data. The study concludes that the implemented system provides a data acquisition platform with sensor data storage and secure communication for easy integration and use in buildings or smart factories.**

***Index Terms*—energy management, encryption, IoT, data acquisition**

I. Introduction

The number of IoT connected devices is predicted to reach nearly thirty billion within this decade [1]. From smart homes to smart cities, IoT has revolutionized the way data are collected and utilized. IoT plays a crucial role in commercial and industrial applications, especially when it comes to increasing efficiency. Whether it is collecting data through sensors or automating monotonous processes, with the growing number of interconnected devices and the increasing economic impact, energy efficiency is a significant factor in industries. However, a large portion of the energy used in the industrial sector is wasted due to poor management [2]. Therefore, it is crucial to have an industrial energy management system that is able to identify and resolve inefficiencies.

Recently, people have become more interested in integrating wireless sensor networks at home settings by utilizing sensors and IoT technologies to monitor and control household processes. These IoT networks have limitations that hinder implementation in the industrial sector, such as limited sensor compatibility, lack of efficient cyber secure communication, and lack of easy database integration. Overcoming these limitations would produce a system able to more easily and safely monitor data such as temperature, occupancy, power usage, tilt, and leakage so as to improve the efficiency of industry buildings.

In this study, we design a low cost, easy to implement, user friendly platform to combat these aforementioned limitations. By utilizing open-source software, a lightweight protocol, and low-cost IoT hardware such as openHAB [3], MQTT [4], and Node Microcontroller Unit (NodeMCU) [5], a data acquisition platform has been created that is relatively inexpensive, secure, easy to implement, and requires low maintenance for use in an industrial or manufacturing environment.

This system utilizes MQTT [4] as it is a lightweight, publish-subscribe, Machine-to-Machine (M2M) communication protocol, designed for not only constrained devices over unreliable networks but also fast and short communication in social networks. This short and simple communication method lowers system requirements, enabling a greater selection of constrained devices to choose from. The publish-subscribe model is based on three main parts: the publisher, which publishes messages to a user-defined topic; the subscriber,

This work has been supported by the US National Science Foundation (Award # 1757636).

which receives messages from the topics it is subscribed to; and finally the broker, similar to a controller, which is responsible for receiving a published message and sending it to all subscribers of that topic. Also, since the protocol is designed to be used by low power devices with unreliable connections, it has support for three Quality of Service (QoS) levels which define the importance of receiving a message.

In order to integrate our design, we decided to use openHAB—a Java-based open-source home automation software [3]. This highly modular vendor agnostic software supports a large variety of IoT devices and sensors, displays data in a user-friendly User Interface (UI), as well as automates device processes. Although the original purpose of this software was home automation, its use can be extended to commercial or industrial settings, where its flexibility in component support and powerful functionality are significantly beneficial.

The primary contributions of this study include:

- Design and development of a data acquisition platform for energy management and monitoring in industrial applications, utilizing MQTT and openHAB.
- Secure MQTT communication using lightweight cryptographic algorithms while adding MQTT payload encryption functionality to openHAB.
- Improving openHAB scalability by dynamically configuring the information of sensors based on a unique MQTT topic structure.

II. Related Works

There exist works related to parts of the data acquisition platform, yet none encompasses the entire system featuring secure encryption, scalability, and back-end functionality. Setiawan and Magfirawaty established secure MQTT communication between mobile and ESP32 devices using the AES-256 algorithm [6]. Terruggia and Garrone evaluated the security and encryption overhead of MQTT cloud communication using Transport Layer Security (TLS) [7]. Heimgaertner et al. developed a method to expand the range of an IoT system for public buildings using openHAB and MQTT [8]. Their design connects multiple MQTT systems in a master-slave configuration, which could be integrated alongside this study to create a complete platform for the data acquisition of large buildings. Chivarov et al. designed an open-source IoT solution for animal farms utilizing MQTT, openHAB, InfluxDB, and Grafana—a database visualization software [9]. Their system features the same open-source software used in our system, but lacks the secure communication needed for industrial applications. By utilizing these open-source software packages and existing encryption algorithms, we have developed a system suitable for the energy management of buildings and smart factories.

There exist many works discussing energy saving methods for homes, commercial buildings, and smart factories, though that is not the focus of this study. Sulistyanto et al. introduced an energy audit concept where energy related sensor data such as temperature, humidity, and current are used to calculate and analyze a metric called the Energy Use Intensity (EUI) [10]. Kychkin et al. also identified and calculated other energy efficient parameters according to the International Standard for Organization (ISO) 50001:2011 [11]. These parameters can then be used to improve energy consumption, scheduling, and management procedures. Yao et al. used IoT devices, energy storage, and appliance scheduling to lower household energy costs by buying energy from the grid when tariffs were low, which lessened the strain on the power grid during active hours [12]. Finally, Shrouf and Miragliotta discussed the energy saving benefits and challenges to consider when IoT energy management systems are being implemented [13]. This study, on the other hand, does not focus on the diverse procedures for energy management but instead develops a data acquisition platform needed for these various energy management methods.

III. Methodology

A. Overall Architecture

Fig. 1 illustrates the overall architecture of the implemented data acquisition system. Like any wireless sensor network, the communication protocol connects various sensing devices to a controlling platform. In this study, the sensing devices are connected to the server platform via encrypted MQTT communication. The MQTT broker serves as a gateway that receives and distributes MQTT messages between the IoT devices and server platform. The home automation platform controls connected devices by receiving, interpreting, and sending MQTT messages. It also stores sensor data to a separate database to be viewed and analyzed. This analysis can be preformed manually or automatically with an energy management algorithm such as [14]. Additionally, this architecture features a custom program to perform automatic GUI configuration in openHAB by dynamically rewriting configuration files. In this study, many open-source software packages are utilized to realize the functions of the server platform.

The following sections explain key aspects of the implemented system in detail.

B. Secure Communication

As discussed prior, lack of security is of chief concern for IoT systems in the industrial sector, and MQTT is not exempt from this concern. As it stands, it offers very few cyber secure options; however, with some modification, it can provide sufficient encryption standards for industrial applications.

1) Underlying Security Mechanism: By default, the MQTT protocol communicates via Transmission Control Protocol (TCP)/Internet Protocol (IP) port 1883 [4], which provides no encryption for communication packets. However, by using port 8883, MQTT can establish secure communication via TLS. This ensures a secure and authenticated connection between both parties—adding network level security as well as complexity. If, however, the network is breached, the messages on the server will be exposed since TLS does not encrypt payload [15]. Furthermore, due to the additional overhead of the TLS protocol [16], it is not ideal for resource constrained,

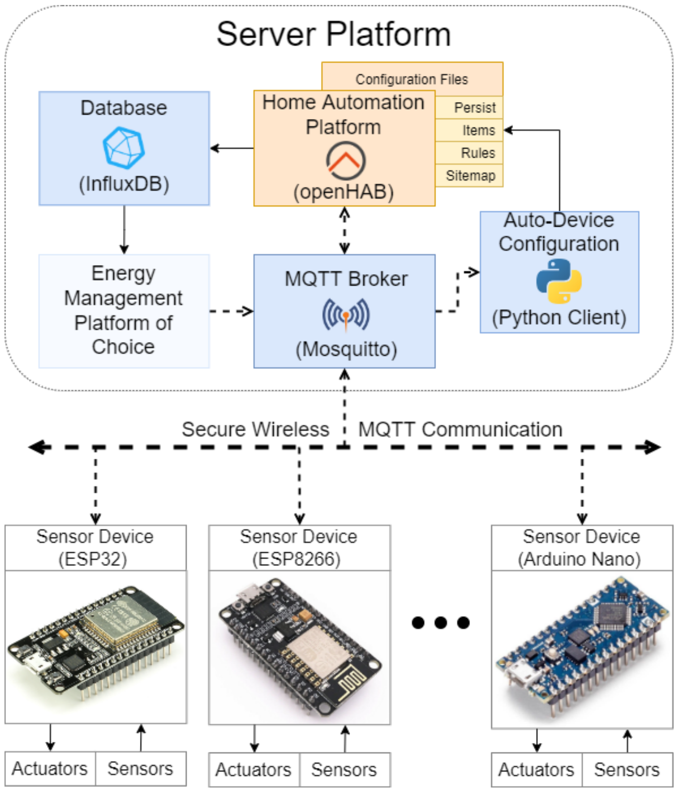


Fig. 1. Overall architecture of the implemented data acquisition system.

low-power IoT devices. As an alternative, payload encryption is a popular choice for constrained devices, where only the payload is converted into ciphertext. In our approach to payload encryption, sensor data gets encrypted and decryped by MQTT clients (i.e., subscribers and publishers) using a lightweight cryptographic algorithm, providing secure communication with minimal overhead.

2) Implemented Algorithms: Cryptography, the study of secure communication techniques, can be divided into two main categories: symmetric key cryptography and asymmetric key cryptography. In symmetric encryption, the same private key is used to both encrypt and decrypt; however, in asymmetric encryption, a pair of keys is used: one key for encryption and the other key for decryption. Due to simplicity, symmetric encryption execution time is notably faster than asymmetric encryption [17], which makes it a reliable choice for constrained IoT devices. Popular algorithms for symmetric key encryption include AES [18], Data Encryption Standard (DES) [19], Blowfish [20], ChaCha20 [21], IDEA [22], etc. In this study, the AES and XChaCha20 algorithms are implemented and compared.

a) AES: Approved by the US Secretary of Commerce, AES is a federal government standard due to its security and performance. According to the National Institute of Standards and Technology (NIST), Rijndael is the cryptographic algorithm used in AES, with the block size of 128 bits and key sizes of 128, 192, or 256 bits in length [18]. In order to add randomness to the generated ciphertext and improve security, the Cipher Block Chaining (CBC) mode [23] is used in this study, with a 128-bit Initialization Vector (IV) or called nonce. An IV is a randomly generated string that is changed every transmission to make the ciphertext unique and untraceable [24].

b) XChaCha20: XChaCha20, which stands for eXtended-nonce ChaCha Cipher construction, is a variant of ChaCha20 with increased security due to the larger IV, decreasing the probability of reusing same nonce [25]. ChaCha is implemented in many widely known protocols such as TLS and Secure Socket Shell (SSH), operating systems such as Linux and Android, and popular software such as Google Chrome, Firefox, and Safari [26]. Unlike AES, ChaCha is a stream cipher, which is generally faster than block ciphers [27]. Furthermore, on constrained devices without built-in hardware support for AES, XChaCha20 is the faster pure software solution. It uses a key size of 256-bit with a 192-bit randomized IV, which results in a higher security margin compared to AES with its 128 bit key and IV [28].

C. Integration Platform

In order to integrate the system and provide sufficient functionality, an automation platform is desired. According to a recent study of open-source platforms, Home Assistant, a Python-based home automation platform, was the most adopted home automation platform with openHAB coming in second place. However, openHAB was found to outperform Home Assistant [29]. This is reasonable given that Java is typically faster than Python. Thus, openHAB was chosen over other open-source platforms because it combines ease of use, performance, and versatility.

Similar to many other platforms, openHAB supports numerous protocols and communication technologies such as Zigbee, Z-Wave, and MQTT. There exist four main concepts to the openHAB framework: Bindings, Things, Channels, and Items. Bindings are add-ons or installed software packages that allow openHAB to understand various supported communication protocols. This binding allows the platform to communicate with Things which are physical devices, such as an ESP32 or ESP8266, in the sensor network. Channels define the types of data that can be received or sent to a particular Thing. For example a temperature and humidity sensor would have two Channels: a temperature Channel and a humidity Channel. These Channels are then linked to Items that virtually store the physical state or value of a Channel. These basic components define how devices communicate sensor data to and from the automation platform. Additional functionality includes Rules and Transformations used to automate the control and formatting of data.

Although originally this open-source platform was designed for home automation, it offers functionality that can provide scalability required for industrial applications. This is accomplished in this study via a Python program that dynamically writes the files of openHAB items, things, persistence, and sitemaps. The purpose of the text files of items and things is self-explanatory: they are used to configure openHAB items and things. The persistence file configures the interval of data aggregation for each item in a database. The sitemap file is used to configure an orderly Graphical User Interface (GUI) to display sensor data and control widgets. Writing these files with a Python program will minimize the time needed to set up an IoT network and reduce configuration complexity.

D. Database Integration

One of the most important aspects of an energy management system is data aggregation and evaluation. In order to provide that functionality, sensor data must be collected in a database with some way to access, analyze, reshape, and visualize them. This can be achieved via the various openHAB compatible databases such as MongoDB, MySQL, and InfluxDB. Although SQL databases are supported by openHAB, we use InfluxDB because it is open-source time-series-based database that effortlessly integrates into openHAB. Time-series-databases save sensor data with a time stamp making them well suited for graphically displaying time-based sensor data. Additionally, an external program can query the InfluxDB database using the Flux query language [30] which permits feeding data sets to a visualization software or energy management algorithm.

IV. Implementation

A. Secure Communication with MQTT Payload Encryption

To realize this system, first, secure communication was considered. Since openHAB does not offer payload encryption for MQTT, Transformations, which restructure or reformat data, must be utilized. Fig. 2 illustrates the steps of encryption and decryption of messages sent from sensor nodes to the server platform (i.e., uplink) in our design. As shown in the figure, the sensor data or payload is encrypted on the sending device using the algorithm of choice. A private key is hard-coded on all clients and must remain secret; however, the encrypted payload and the IV do not need to be private, so they are both sent in the payload for decryption. In order to publish these decrypting parameters in the same message, they are stored in a Java Script Object Notation (JSON) array, which is then encoded in Base64 to prevent the loss or modification of binary data over transmission.

The MQTT broker transfers the encrypted and Base64-encoded message to openHAB's MQTT client, which passes it to the openHAB platform. openHAB features Incoming Value Transformations in which data are reshaped or pre-processed to a desired format before they are used or stored. In order to decrypt the payload on arrival, we used openHAB's EXEC Transformation Service [3] to run a decrypting program. As illustrated in Fig. 3, whenever a new message arrives, EXEC Transformation automatically runs a command that calls the decrypting program with the encrypted and Base64-encoded MQTT message as an input. The program decodes the incoming message to retrieve the JSON array containing the ciphertext and IV, and uses these parameters, along with the hard-coded key, to retrieve the decrypted message. Finally,

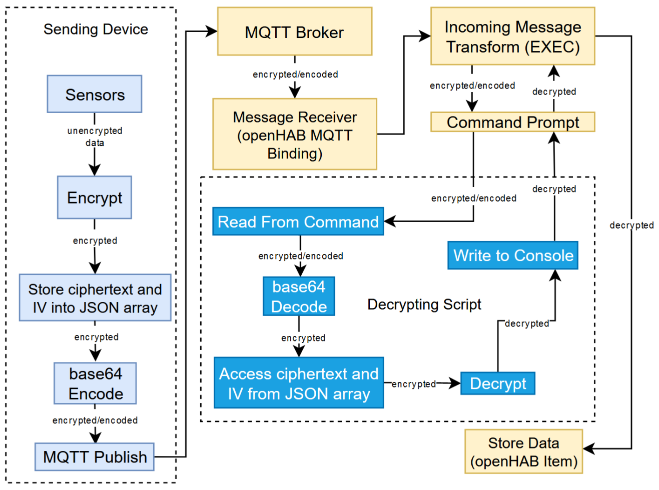
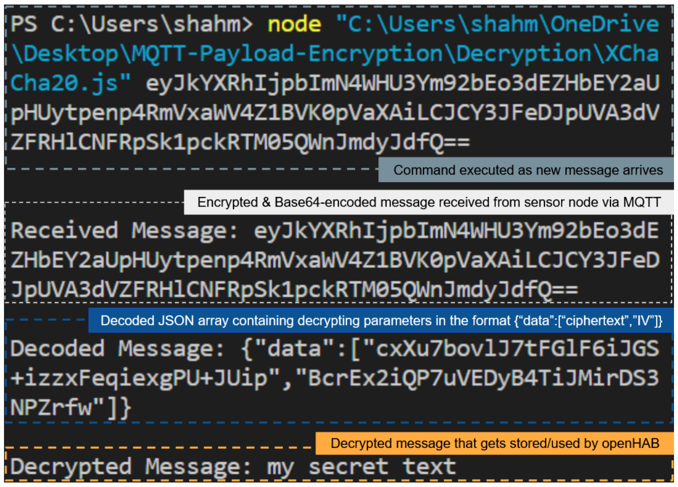


Fig. 2. The steps of message encryption and decryption from devices to platform.



```
PS C:\Users\shahm> node "C:\Users\shahm\OneDrive
\Desktop\MQTT-Payload-Encryption\Decryption\XCha
Cha20.js" eyJkYXRhIjpbImN4WHU3Ym92bEo3dEZHbEY2aU
pHUytpenp4RmVxaWV4Z1BVK0pVaXAiLCJCY3JFeDJpUVA3dV
ZFRHlCNFRpSk1pckRTM05QWnJmdyJdfQ==
```

Command executed as new message arrives

Encrypted & Base64-encoded message received from sensor node via MQTT

```
Received Message: eyJkYXRhIjpbImN4WHU3Ym92bEo3dE
ZHbEY2aUpHUytpenp4RmVxaWV4Z1BVK0pVaXAiLCJCY3JFeD
JpUVA3dVZFRHlCNFRpSk1pckRTM05QWnJmdyJdfQ==
```

Decoded JSON array containing decrypting parameters in the format {“data”:[“ciphertext”,”IV”]}

```
Decoded Message: {"data":["cxXu7bovlJ7tFGlF6iJGS
+izzxFeqiexgPU+JUip","BcrEx2iQP7uVEDyB4TiJMirDS3
NPZrfw"]}
```

Decrypted message that gets stored/used by openHAB

```
Decrypted Message: my secret text
```

Fig. 3. Console view of the uplink message decryption script.

the plaintext is printed to the console, and openHAB's Transformation service automatically reads the value and stores it in the designated item, resulting in a "decrypting transformation" of the encrypted payload.

The steps of the encryption and decryption of messages sent from the platform to sensor nodes (i.e., downlink) are shown in Fig. 4. Similar to the uplink transformation, openHAB also supports Outgoing Value Transformations, where the messages needing to be published from the platform get transformed to a desired value or format. In this case, the raw message and IV are sent to an encrypting program as command line arguments via an EXEC Transformation. Then, the program encrypts the data, stores the encrypted data and IV in a JSON array, encodes in Base64, and outputs the ciphertext to the console. The EXEC Transformation passes the encrypted output to the MQTT client which publishes it to the broker. The receiving device can then receive, decode, and decrypt the payload to respond accordingly to the MQTT message.

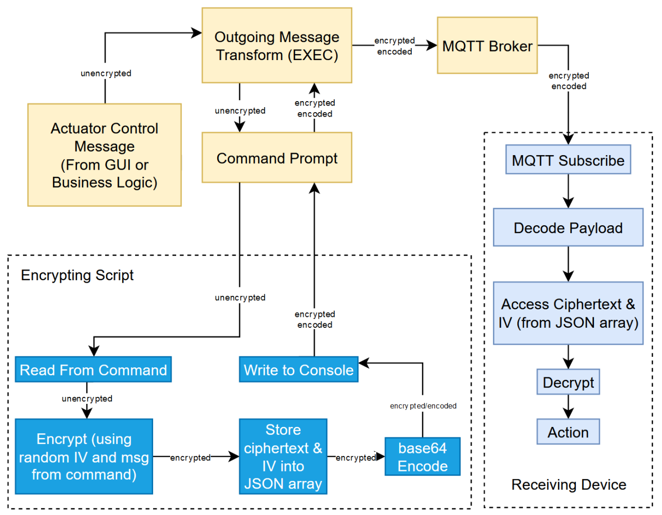


Fig. 4. The steps of message encryption and decryption from platform to devices.

B. Platform Integration

As discussed prior, this study utilizes openHAB to configure and control IoT devices. openHAB offers several methods of device configuration, namely a UI, text configuration files, and a command line, each with their own limitations and strengths. For this study, the UI and text configuration files are utilized with the UI being the most user-friendly. The UI can configure sensor items one by one, but this would become a time-consuming task with hundreds of sensors. Thus, the text configuration files may be more suitable for dynamically re-configuring GUI of openHAB when sensor changes are detected through MQTT topics.

1) Our Approach to Device Integration: One goal of this study is to facilitate easy integration of numerous sensors in a smart factory or building application. Our system attempts to accomplish this by dynamically writing the various openHAB configuration files using a Python program. These are text files that follow a specific format as outlined in the documentation section of openHAB's website [3]. The manner in which these configuration files are populated with sensor details is reliant on the MQTT protocol in our study.

Since MQTT uses a topic-based publish-subscribe approach, the topic can be structured such that it contains detailed sensor information. These details include aspects such as the location, unique Media Access Control (MAC) Identification (ID), sensor type, and sensor port number. The devised topic structure looks as follows: *Project Name / Building ID / Room Number / Device Name / MAC ID / Port Number / Sensor Type / Unit of Measurement*. With a combination of these elements, one can devise a naming scheme for all segments found in the openHAB configuration files. A few examples of these segments include unique item names, sensor types, item properties, and labels. Though the files were initially created manually, this process was automated with a Python program.

Multiple programming languages could have been chosen for this task, but Python was chosen for its simplicity when parsing the topic string and writing to the openHAB configuration files. Also, there exists an open-source Python MQTT client named Eclipse Paho MQTT Client which is able to preform many functions for MQTT communication. In order to write the openHAB files, the Python program needs to first determine if an MQTT topic is new and configure that new sensor. It then places the new topic in a database file that contains all unique topics and add the new sensor details to the openHAB configuration files. Using this Python client and a Comma Separated Value (CSV) spreadsheet file, the Python program is able to configure sensors when a new topic is received and display them to a sitemap GUI. In an industrial setting, this program would decrease the installation time and configuration complexity of new devices.

C. Database Integration

The sensor data are stored in the chosen database, InfluxDB. Since InfluxDB and openHAB integrate seamlessly, connecting the two was a matter of installing an add-on and configuring the two. Once set up, graphs could be viewed from the database GUI and could be queried by external programs.

V. Experimentation

In order to verify the functionality of the implemented system, various aspects such as transmission speeds, configuration functionality, and database querying were tested. To ensure that the system is suitable for industrial applications, we first tested the encryption speed of the openHAB Transformation and encryption script. Also the openHAB configuration program was tested to ensure the configuration files were created accurately. Finally, the whole system was verified by gathering sensor data, saving them to the database, and querying them by an external program.

A. Performance Testing

To measure the efficiency of incoming openHAB message decryption, a script was created to run a command identical to openHAB's EXEC Transformation Service. The script runs the command 1000 times, and in every iteration, the encrypted payload and IV are used to convert the payload back to plaintext. The measured time is the total time elapsed as the script starts, reads the input from command line arguments, decodes and decrypts information, and returns the readable plaintext. The scatter plot in Fig. 5 illustrates the execution time per trial for the one thousand trials for two encryption algorithms, with their averages shown as dashed lines. The average execution times measured were 12.716 milliseconds using AES, and 11.133 milliseconds using XChaCha20, which is 12.4% shorter than AES, even though XChaCha20 uses a 2X larger key and 1.5X larger nonce.

This variation between the algorithms was expected based on related works utilizing these algorithms. Also, since energy management systems do not require data to be transferred any faster than once per second or even minute, these speeds are sufficient to support numerous devices. More than likely,

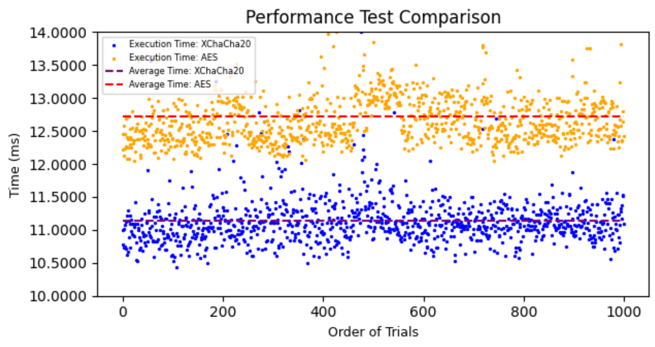
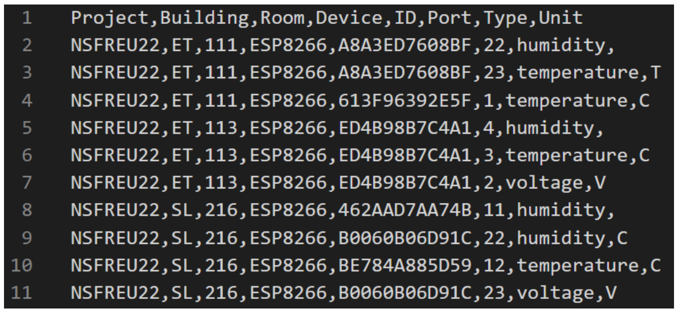


Fig. 5. Performance Test Results.

industries will run into network bandwidth issues before these speeds become an issue.

B. Configuration Functionality Test

A test was also performed to ensure that the dynamic file configuration program properly adds new sensors to openHAB. This test was conducted by altering the topic on an ESP8266 device according to the topic structure discussed previously and by publishing sample payload values to the broker. The topic and message were then sent via MQTT and received by the Python client. The program determined if the topic was unique. If so, that topic was saved to a CSV file containing all the unique sensor topics. Fig.6 shows the CSV file which contains the ten unique example topic strings that were used for this test.



```
Project,Building,Room,Device,ID,Port,Type,Unit
NSFREU22,ET,111,ESP8266,A8A3ED7608BF,22,humidity,
NSFREU22,ET,111,ESP8266,A8A3ED7608BF,23,temperature,T
NSFREU22,ET,111,ESP8266,613F96392E5F,1,temperature,C
NSFREU22,ET,113,ESP8266,ED4B98B7C4A1,4,humidity,
NSFREU22,ET,113,ESP8266,ED4B98B7C4A1,3,temperature,C
NSFREU22,ET,113,ESP8266,ED4B98B7C4A1,2,voltage,V
NSFREU22,SL,216,ESP8266,462AAD7AA74B,11,humidity,
NSFREU22,SL,216,ESP8266,B0060B06D91C,22,humidity,C
NSFREU22,SL,216,ESP8266,BE784A885D59,12,temperature,C
NSFREU22,SL,216,ESP8266,B0060B06D91C,23,voltage,V
```

Fig. 6. Example topic strings stored in a CSV file.

If the topic was unique and thus was added to this CSV file, then the openHAB text configuration files were written with the new sensor configuration by the Python client. A standard sitemap layout used by openHAB GUI renderer was created which organizes the sensors by location. Sensors in the same building are categorized together, and sensors in the same room are subdivided by room number. Each room number has a virtual switch that allows the user to show or hide sensors for that room. The standard layout can be observed in Fig.7. This program gives users organized and instant access to new sensor data.

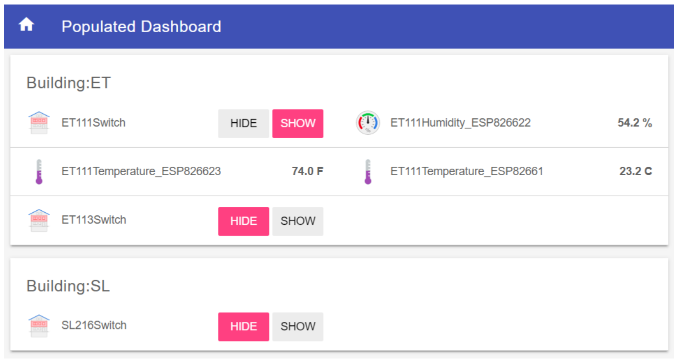


Fig. 7. The sitemap view output.

C. Query-able Sensor Data Acquisition

With the sensor configuration complete, sensor data could be sent via MQTT to the home automation platform and stored in the database. With an on-hand DHT22 sensor, the system saved temperature and humidity values in an air-conditioned building over a duration of two hours with a sampling period of ten seconds. That data was queried from the database using a Flux query language Python library and graphically illustrated. Fig.8 shows the results of our testing showing temperature and humidity values over two hours. This test, on a small scale, validated the aggregation of sensor data utilizing all aspects of our system.

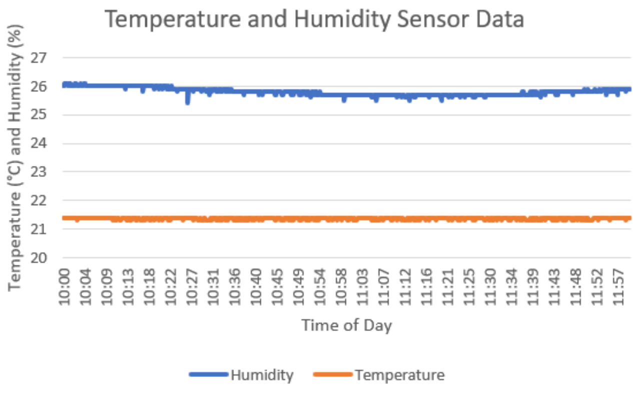


Fig. 8. Sensor Test Results

VI. Conclusion

In this study, an energy management data acquisition platform utilizing IoT technologies has been explored. This platform is based around the MQTT protocol and an open-source home automation platform called openHAB. The acquisition platform is designed with security in mind, and hence, two payload encryption algorithms, namely AES and XChaCha20, have been compared and integrated using openHAB's EXEC Transformation. Through the testing preformed, it was observed that the speed of encryption would not greatly hinder the performance of the whole system. Thus, this is a useful

MQTT-openHAB encryption method for industrial applications.

Scalability is another important aspect of industrial energy management systems, which is explored in this work. To improve scalability, an external Python program was coded and then used to parse descriptive MQTT topics to populate configuration files for new sensors. OpenHAB uses these files to generate a sitemap for instant visualization of sensor data. With this program in place, the system can automatically configure sensor data processing and GUI for the platform, thereby saving the user implementation and debugging time.

The platform is also designed to save sensor data to a database. To accomplish this, the platform integrates with an open-source database called InfluxDB. Through the database testing, the data acquisition platform saved sensor data properly enabling the easy visualization of them. The platform was also tested to be queried from an external program that could send data to an energy management platform of choice.

References

[1] L. Vailshery, "IOT connected devices worldwide 2019-2030," Statista, June 2022. [Online] Accessed July 2022, https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2] T. Phillips and L. West, "How to reduce energy waste in the manufacturing sector," Processing Magazine, June 2020. [Online] Accessed July 2022, https://www.processingmagazine.com/maintenance-safety/asset-management/article/21140469/how-to-reduce-energy-waste-in-the-manufacturing-sector

[3] "Empowering the smart home," openHAB. [Online] Accessed July 2022, https://www.openhab.org/

[4] "MQTT: The standard for IOT messaging," MQTT. [Online] Accessed July 2022, https://mqtt.org/

[5] "ESP8266," Espressif. [Online] Accessed July 2022, https://www.espressif.com/en/products/socs/esp8266

[6] F. Setiawan and Magfirawaty, "Securing data communication through MQTT protocol with AES-256 encryption algorithm CBC mode on ESP32-based smart homes," *Int. Conf. on Computer System, Information Technology, and Electrical Engineering (COSITE)*, pp. 166-170, 2021, doi: 10.1109/COSITE52651.2021.9649577.

[7] R. Terruggia and F. Garrone, "Secure IoT and cloud based infrastructure for the monitoring of power consumption and asset control," *AEIT Int. Annu. Conf. (AEIT)*, pp. 1-6, 2020, doi: 10.23919/AEIT50178.2020.9241195.

[8] F. Heimgaertner, S. Hettich, O. Kohlbacher, and M. Menth, "Scaling home automation to public buildings: A distributed multiuser setup for openHAB 2," *Global Internet of Things Summit (GIoTS)*, pp. 1-6, 2017, doi: 10.1109/GIOTS.2017.8016235.

[9] S. Chivarov, N. Chivarov, D. Chikurtev, and M. Pleva, "Cost-oriented software system for animal husbandry smart automation," *Int. Conf. Automatics and Informatics (ICAI)*, pp. 256-261, 2021, doi: 10.1109/ICAI52893.2021.9639708.

[10] M. Sulistyanto, K. Pranata, and Solikhan, "Preliminary study of utilizing Internet of Things for monitoring energy use in building to support energy audit process," *4th International Conf. on Computer Applications and Information Processing Technology (CAIPT)*, pp. 1-7, 2017, doi: 10.1109/CAIPT.2017.8320705.

[11] A. Kychkin, A. Deryabin, E. Neganova, and V. Markvirer, "IoT-based energy management assistant architecture design," *IEEE 21st Conf. on Business Informatics (CBI)*, pp. 522-530, 2019, doi: 10.1109/CBI.2019.00067.

[12] L. Yao, J. Y. Shen, and W. H. Lim, "Real-time energy management optimization for smart household," *IEEE Int. Conf. on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 20-26, 2016, doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.31.

[13] F. Shrouf and G. Miragliotta, "Energy management based on internet of things: practices and framework for adoption in production management," *Journal of Cleaner Production*, vol. 100, pp. 235-246, 2015, ISSN 0959-6526, doi: 10.1016/j.jclepro.2015.03.055.

[14] N. Kodama, T. Harada, and K. Miyazaki, "Home energy management algorithm based on deep reinforcement learning using multistep prediction," in *IEEE Access*, vol. 9, pp. 153108-153115, 2021, doi: 10.1109/ACCESS.2021.3126365.

[15] E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.3," Datatracker, August 2018. [Online] Accessed July 2022, https://datatracker.ietf.org/doc/html/rfc8446

[16] G. Restuccia, H. Tschofenig, and E. Baccelli, "Low-power IoT communication security: On the performance of DTLS and TLS 1.3," *9th IFIP Int. Conf. on Performance Evaluation and Modeling in Wireless Networks (PEMWN)*, pp. 1-6, 2020, doi: 10.23919/PEMWN50727.2020.9293085.

[17] F. Maqsood, M. Ahmed, M. Mumtaz, and M. Ali, "Cryptography: A comparative analysis for modern techniques," *Int. Journal of Advanced Computer Science and Applications*, vol. 8(6), 2017, doi: 10.14569/IJACSA.2017.080659.

[18] "Announcing the Advance Encryption Standard (AES)," FIPS 197, National Institute of Standards and Technology (NIST), November 2001. [Online] Accessed July 2022, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[19] "Announcing the data encryption standard" FIPS 46-3, National Institute of Standards and Technology (NIST), October 1999 [Online] Accessed July 2022, https://csrc.nist.gov/CSRC/media/Publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf

[20] B. Schneier, "The blowfish encryption algorithm," Schneier on Security. [Online] Accessed July 2022, https://www.schneier.com/academic/blowfish/

[21] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," Datatracker, June 2018. [Online] Accessed July 2022, https://datatracker.ietf.org/doc/html/rfc8439

[22] B. Schneier, "IDEA," in *Applied Cryptography*. Indianapolis, IN: John Wiley & Sons, 1996.

[23] C. Paar and J. Pelzl, "Understanding cryptography". Springer-Verlag Berlin Heidelberg, 2010, doi: 10.1007/978-3-642-04101-3.

[24] D. Blazhevski, A. Bozhinovski, B. Stojchevska, and V. Pachovski, "Modes of operation of the AES algorithm," *The 10th Conference for Informatics and Information Technology*, 2013, pp. 212-216.

[25] S. Arciszewski, "XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305 draft-arciszewski-xchacha-02," Datatracker, October 2018. [Online] Accessed July 2022, https://datatracker.ietf.org/doc/html/draft-arciszewski-xchacha-0

[26] "Chacha usage & deployment," IANIX. [Online] Accessed July 2022, https://ianix.com/pub/chacha-deployment.html

[27] S. Sharif and S. Mansoor, "Performance analysis of stream and block cipher algorithms," *3rd Int. Conf. on Advanced Computer Theory and Engineering (ICACTE)*, 2010, pp. V1-522-V1-525, doi: 10.1109/ICACTE.2010.5578961.

[28] J. Aumasson, "Too much crypto," *IACR Cryptol. ePrint Arch.*, 2019. [Online] Accessed July 2022, https://eprint.iacr.org/2019/1492

[29] B. Setz, S. Graef, D. Ivanova, A. Tiessen and M. Aiello, "A comparison of open-source home automation systems," in *IEEE Access*, vol. 9, 2021, pp. 167332-167352, doi: 10.1109/ACCESS.2021.3136025.

[30] "Query data with flux," *InfluxData*. [Online] Accessed June 2022, https://docs.influxdata.com/influxdb/v2.3/query-data/flux/