Department of Energy/National Energy Technology Laboratory

DOE Award Number: DE-FE0031745

Final Report

Project Title: Secure Data Logging and Processing with Blockchain and Machine Learning

Principal Investigator (PI)

Leonel Lagos, Ph.D., PMP®
Florida International University (FIU)
lagosl@fiu.edu

Co-Principal Investigators (Co-PIs)

Himanshu Upadhyay, Ph.D., PMP®
Florida International University (FIU)
upadhyay@fiu.edu

Wenbing Zhao, Ph.D.
Cleveland State University (CSU)
w.zhao1@csuohio.edu

Key Personal:
Santosh Joshi, MS
Florida International University (FIU)
sajoshi@fiu.edu

Administrative Contact
Roberto M. Gutierrez, M.S.
(305) 348-2494
gutierrr@fiu.edu

Recipient Organization
Applied Research Center
Florida International University
10555 W. Flagler Street EC2100
Miami, FL 33174

Project Period: 9/1/2019 – 4/30/2023
Reporting Period End Date: 4/30/2023

Signature of Submitting Official

Table of Contents

1. EXECUTIVE SUMMARY:

Secure Data Logging and Processing with Blockchain and Machine Learning (ML) research is focused on the development of a platform to securely log and process sensor data in fossil power plants. The platform integrates two emerging technologies, blockchain and ML, and incorporates several innovative mechanisms to ensure the integrity, reliability, and resiliency of power systems. The goal is to protect the power plant from various cyberattacks such as false data injection and denial of service attacks using these technologies. The research goal was enabled by the following Research Project Objectives: 1) Secure authentication and identity verification of sensor nodes, actuators, and other equipment within a network. 2) Development of mechanisms that ensure only data sent by legitimate sensors are accepted and stored in the data repository. 3) Development of data aggregation methodologies using ML / Deep Learning (DL) algorithms to minimize noise / faulty data. 4) Implementation of the blockchain technologies to provide data security using secured IOTA framework & nodes.

2. INTRODUCTION:

This document presents the final report of the research activity completed during the three years of the project and includes the major challenges and accomplishments. The report presents specific details of the methodologies used to develop the pipeline for a secure data logging and processing platform using machine learning and blockchain and complete the project milestones. In particular, the report includes sections that address the problem statement, background, development of the entire pipeline which includes synthetic sensor data generation, sensor identity management, sensor data anomaly detection using ML, secure logging and storage of sensor data using blockchain and finally the web application to manage the entire pipeline. The report concludes with a discussion on potential paths forward and concluding remarks.

3. TECHNICAL APPROACH AND RESULTS:

This section contains the methodology used to develop the pipeline for secure data logging and processing platform using machine learning and blockchain. It begins with a section on the motivation and background of the research and is followed with a description of our approach to addressing the issues.

3.1 Background:

Power plants are expensive assets in the power industry, and maintaining healthy operation of power plants is very important. Unexpected faults may cause generators to be unable to meet performance standards, resulting in poor reliability and high penalties. The conventional maintenance activities are carried out according to time schedules, which can incur unnecessary shutdowns and manpower costs. With the deployment of advanced sensing techniques, data-driven condition monitoring has been widely recognized as an essential function by asset owners for its potential to improve asset longevity and reduce maintenance costs.

These power plants are equipped with many heterogeneous Internet of Things (IoT) devices that generate and collect massive amounts of data through the deployment of a multitude of sensors that are prone to the possibility to obtain anomalous or faulty data. This can be attributed to various causes such as device malfunctioning or tampering. This malicious data can cause issues for IoT systems and may provide incorrect insights. Many times, due to temporal patterns and high-dimensional data structures, these anomalies may be difficult to detect. Hence, there is a need to timely detect these anomalies in the data for maintenance and to prevent losses. Anomaly detection is a technique that identifies abnormal data or data that deviates from its historical pattern. The detection of such data is critical in a variety of applications, which includes fraud detection, and intrusion detection etc. Furthermore, detecting anomalies from a huge amount of data is becoming a requirement for several new systems/applications. Additionally, the results of anomaly detection using Artificial Intelligence (AI) and the massive volume of data collected by these IoT devices must be stored in a distributed and decentralized manner. While traditional databases and centralized storage can become a single point of failure, there is a need to store the huge amount of IoT data in a distributed ledger such as a blockchain that is decentralized and solves the issues of traditional and centralized storage systems. Therefore, there is a need to build a pipeline that integrates sensor identification, anomaly detection and blockchain to overcome the mentioned challenges.

Blockchain technology was introduced in 2008 by Satoshi Nakamoto and has become a disruptive and significant technology for Industry 4.0. Blockchain is a distributed ledger or database which is decentralized, immutable, and operates in a peer-to-peer manner without the need for any intermediary or central authority. The above-mentioned features of blockchain ensure that the data is secured, authentic, and immutable once it gets stored on the blockchain. Using this decentralized blockchain technology to store aggregated IoT data logs can be useful, particularly if transparency and immutability are required, such that every participant will be able to view the data without modifying it. However, blockchain technology has certain limitations, which restrict its integration with the IoT. For instance, the size of a block in the blockchain is limited to 1 MB and the mining time is about 10 min. additionally, scalability is a major challenge for many types of blockchain technology. This hinders its application for IoT systems where scalability is important. Furthermore, there is also a possibility of selfish mining in the blockchain, where miners can achieve higher rewards than their allocated amount. Therefore, to meet the current demand of the IoT systems, there is a need for a distributed ledger that can seamlessly integrate with IoT and overcome the limitations of the traditional blockchain. The IOTA distributed ledger was introduced in 2015, and it uses a data structure called the Tangle that is a Directed Acyclic Graph (DAG) in contrast to the chained block structure of blockchain. IOTA was designed to address the resource limitations of IoT devices. Hence, IOTA overcomes the limitations of blockchain, particularly the transactional throughput, which makes it the most suitable choice for IoT systems.

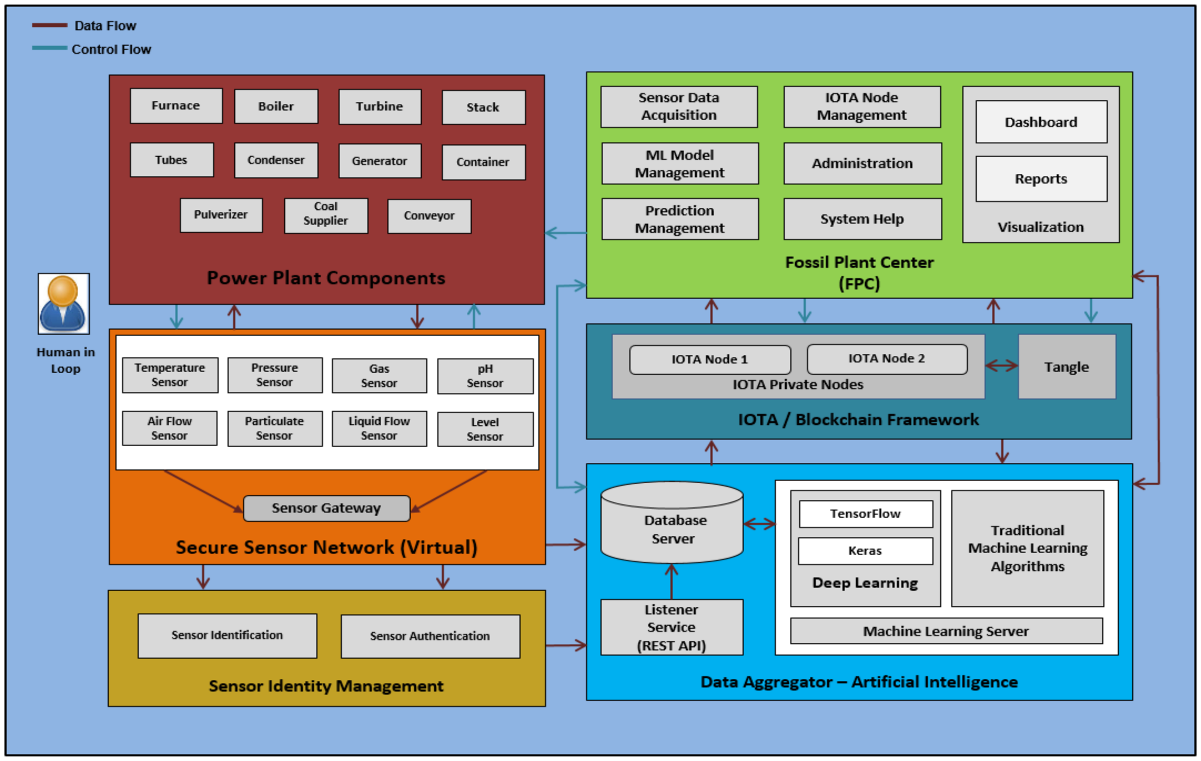


Figure 1: System Architecture - Secure Data Logging & Processing with Blockchain and Machine Learning System

This project aimed to build a platform which includes three main components: 1) Sensor identity management, 2) Anomaly detection using AI, and 3) Secure storage of sensor data using blockchain. The sensor identity management module ensures that the data transmitted from the sensors in a fossil fuel power plant is legitimate and not from malicious sensors. The anomaly detection module ensures that the power plant is up and running continuously by detecting faulty sensors or power plant component failures timely. The third module is for the secure storage and retrieval of sensor data by using blockchain technology. Finally, to demonstrate the usability of this platform, a front-end application was built with all the three modules. A schematic of the entire pipeline is shown in figure 1.

3.2 Sensor Identity Management:

In sensor identity management, research was conducted on fossil fuel power plants to identify various components and sensors used to collect data. Once we identified the components and the sensors, the next step was to generate synthetic sensor data which mimics the actual data collected from these plants. Finally, once the data was generated the last step was to implement identity management techniques to make sure that the data collected from these sensors is legitimate.

3.2.1 Research on Different Components and Sensors in Fossil-fuel Power Plants:

Research was conducted on different fossil-fuel power plant datasets and academic literature review on different types of sensors and components to understand threshold sensor values that are normally seen in different sections of the power plants as shown in figure 2.

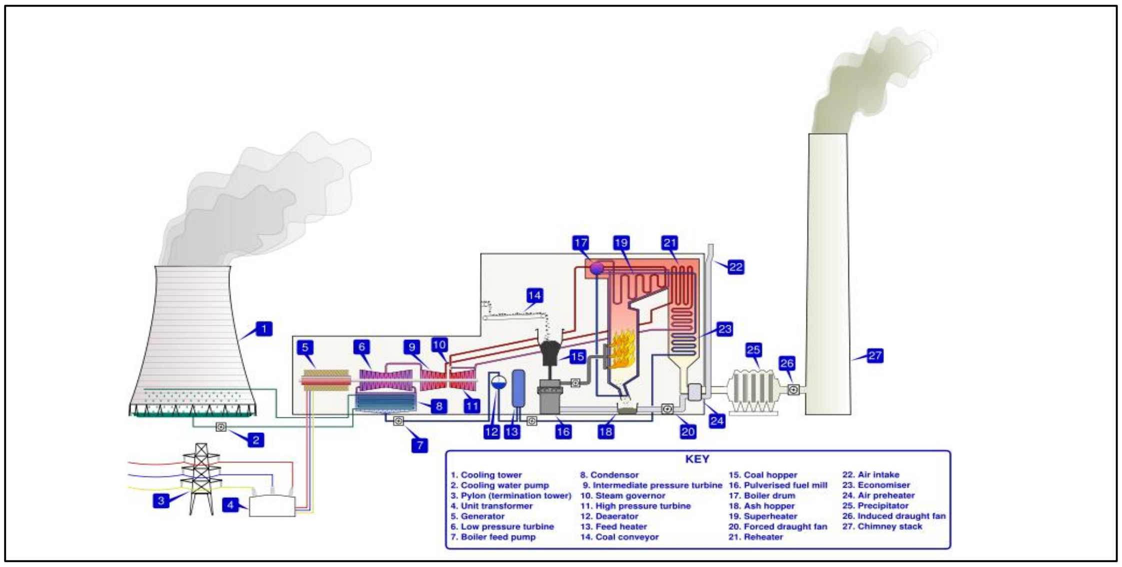


Figure 2: Components in a Traditional Power Plant

The purpose of finding these value ranges was to generate meaningful simulated data to train ML models that can detect anomalies in the plant's system components, hence improving the security of the facilities.

Based on the investigation, it was observed that the most amount of gas sensors are found in the flue stack of the plants. Additionally, temperature sensor ranges can vary between different components, and the same can be said for pressure sensors. On the other hand, different sensors can be used in the turbines to check the health of the component. Some of the sensors that can be found in the turbine are the vibration sensor and the pressure sensor as shown in table 1.

Table 1: Sensor Value Ranges in Fossil-Fuel Power Plants

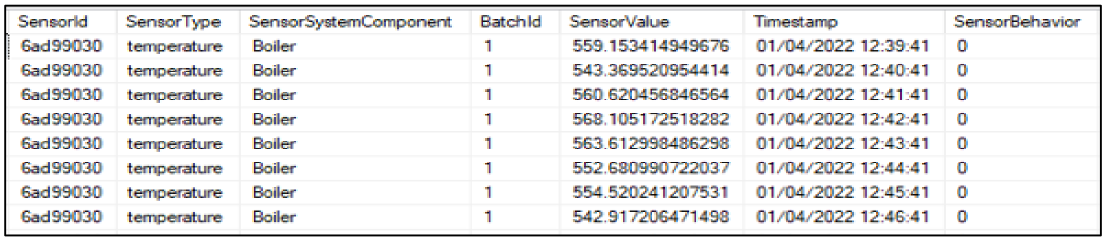
Sensor	Unit	System Component	Value Range
Temperature	°C	Turbine	1.81 – 37.11
Temperature	°C	Boiler	540 – 570
Pressure	hPa	Turbine	992.89 – 1033.3
Pressure	hPa	Boiler	600 – 2465
Vibration	µm (pk-pk)	Turbine	13.31 – 14.07
O_2	% in flue	Stack	1.6 – 4.1
Gas	ppm	Stack	10 – 1000
Gas	ppm	Boiler	0 – 100
CO_2 Gas	% in flue	Stack	8 – 10
NO_2 Gas	%in flue	Stack	10 – 12
pH	-	Boiler	7 - 9

3.2.2 Generating Sensor Data:

With the sensor information obtained from the research, artificial sensor data was generated to train the machine learning models. The data was generated by using Python scripts that would create random values for the output of the sensors at a specific instance of time. The scripts were able to generate the data for any of the sensors mentioned above and could create outputs for normal sensors and malicious sensors. Therefore, the use of anomaly detection models would be trained with generated datasets and would be validated with the same type of artificial data.

These models would be tested against the malicious sensor data type.

Dataset Features: The dataset contained six features and one label as shown in figure 3. The label was represented with values of '0' and '1' to signal if the sensor value was normal or malicious, respectively. Moreover, each record in the dataset was composed of a specific sensor ID, that belonged to a system component (also included as a feature), and the type of sensor which indicates if it is a temperature, pressure, vibration, gas, etc. The sensor value was a float data type value that is the output of the sensor at the designated timestamp (another feature of the dataset).



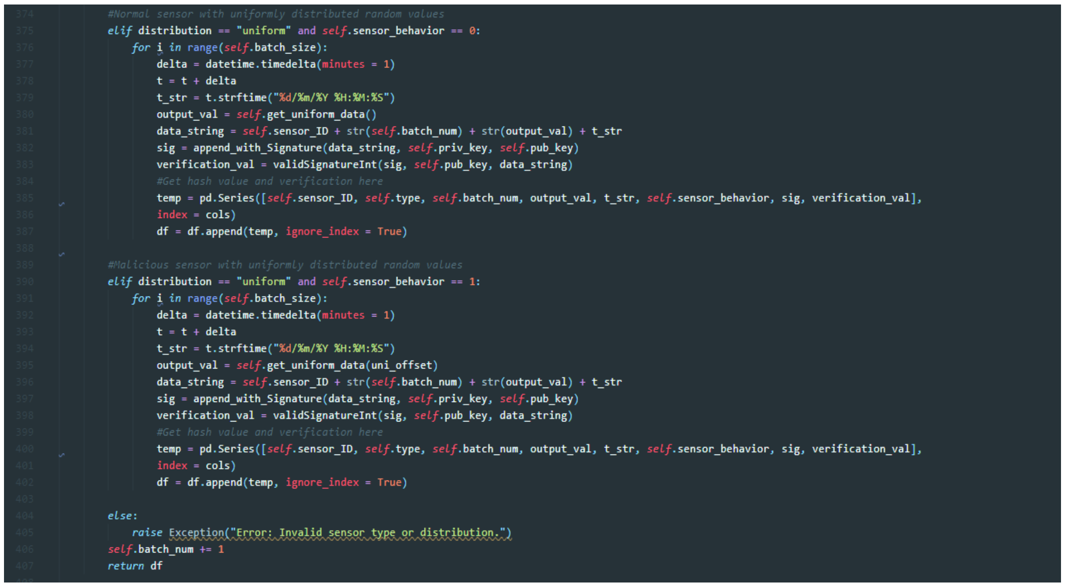
SensorId	SensorType	SensorSystemComponent	BatchId	SensorValue	Timestamp	SensorBehavior
6ad99030	temperature	Boiler	1	559.153414949676	01/04/2022 12:39:41	0
6ad99030	temperature	Boiler	1	543.369520954414	01/04/2022 12:40:41	0
6ad99030	temperature	Boiler	1	560.620456846564	01/04/2022 12:41:41	0
6ad99030	temperature	Boiler	1	568.105172518282	01/04/2022 12:42:41	0
6ad99030	temperature	Boiler	1	563.612998486298	01/04/2022 12:43:41	0
6ad99030	temperature	Boiler	1	552.680990722037	01/04/2022 12:44:41	0
6ad99030	temperature	Boiler	1	554.520241207531	01/04/2022 12:45:41	0
6ad99030	temperature	Boiler	1	542.917206471498	01/04/2022 12:46:41	0

Figure 3: Example of Generated Records for Training and Testing datasets

Dataset Description: The two datasets were created to develop the anomaly detection pipeline using machine learning algorithms. The first dataset was the training dataset, consisting of 40,000 records and containing only normal temperature sensor values. The testing dataset contained 8,000 examples with 80% of them labeled as "normal" data and the rest labeled as "malicious data." The testing dataset was later used for prediction / anomaly detection. Each record contained the type of sensor, the system component, the sensor ID, the batch ID, the sensor value, and the timestamp. These features were selected to identify each output by its sensor ID.

3.2.3 Implementation of Artificial Sensor Data Generation:

To create the dataset, a Python script, as shown in figure 4, was used to simulate the behavior of sensors.



```python
#Normal sensor with uniformly distributed random values
elif distribution == "uniform" and self.sensor_behavior == 0:
    for i in range(self.batch_size):
        delta = datetime.timedelta(minutes = 1)
        t = t + delta
        t_str = t.strftime("%d/%m/%Y %H:%M:%S")
        output_val = self.get_uniform_data()
        data_string = self.sensor_ID + str(self.batch_num) + str(output_val) + t_str
        sig = append_with_Signature(data_string, self.priv_key, self.pub_key)
        verification_val = validSignatureInt(sig, self.pub_key, data_string)
        #Get hash value and verification here
        temp = pd.Series([self.sensor_ID, self.type, self.batch_num, output_val, t_str, self.sensor_behavior, sig, verification_val],
        index = cols)
        df = df.append(temp, ignore_index = True)

#Malicious sensor with uniformly distributed random values
elif distribution == "uniform" and self.sensor_behavior == 1:
    for i in range(self.batch_size):
        delta = datetime.timedelta(minutes = 1)
        t = t + delta
        t_str = t.strftime("%d/%m/%Y %H:%M:%S")
        output_val = self.get_uniform_data(uni_offset)
        data_string = self.sensor_ID + str(self.batch_num) + str(output_val) + t_str
        sig = append_with_Signature(data_string, self.priv_key, self.pub_key)
        verification_val = validSignatureInt(sig, self.pub_key, data_string)
        #Get hash value and verification here
        temp = pd.Series([self.sensor_ID, self.type, self.batch_num, output_val, t_str, self.sensor_behavior, sig, verification_val],
        index = cols)
        df = df.append(temp, ignore_index = True)

else:
    raise Exception("Error: Invalid sensor type or distribution.")
self.batch_num += 1
return df
```

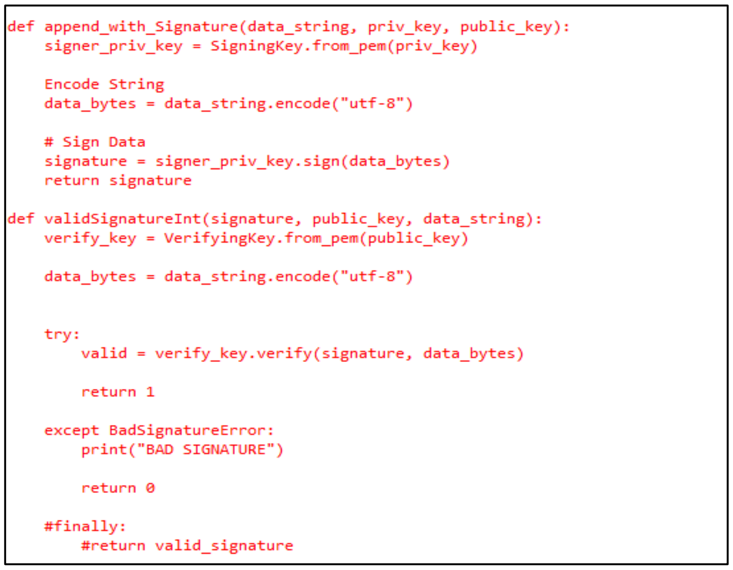
Figure 4: Sensor Class in Python script Generating Normal or Malicious Data

A series of sensor values were generated with a difference of one minute each. The normal sensors would generate uniformly distributed random values from a defined interval. Whereas the malicious sensors would generate sensor values from this uniform distribution mentioned earlier and an offset would be added or subtracted to the final value. This ensured that the malicious data would differ from the normal data.

Generating the training dataset entailed only instantiating one sensor and running the data generation method. However, the testing dataset needed to include normal data and malicious data. For this case, multiple processes were created to simulate if both sensors would generate data at the same time. One process generated normal data while the other generated malicious data. Subsequently, the records were inserted in a pandas Data Frame and then shuffled.

3.2.4 Sensor Identity Management:

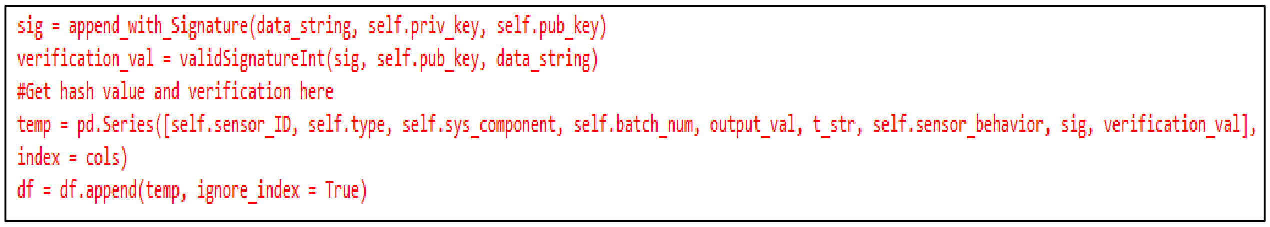
The team developed two python scripts that generate a private and public key to digitally sign and verify all data that is being generated. The first function, "append_with_Signature" as shown in figure 5, returned a signature generated from the data string that was encoded using UTF-8. The data string was a concatenation of the features of sensor data that included the sensor ID, batch number, sensor output value, and time. The private key was used to sign the "data_string", which was then returned as the signature. The second function, "valid_signature_int", will verify the signature generated by the first function and returns 0 if the signature is not valid and 1 if the signature is valid. The public key was used to verify the signature generated for each system component.



```
def append_with_Signature(data_string, priv_key, public_key):
    signer_priv_key = SigningKey.from_pem(priv_key)

    Encode String
    data_bytes = data_string.encode("utf-8")

    # Sign Data
    signature = signer_priv_key.sign(data_bytes)
    return signature

def validSignatureInt(signature, public_key, data_string):
    verify_key = VerifyingKey.from_pem(public_key)

    data_bytes = data_string.encode("utf-8")


    try:
        valid = verify_key.verify(signature, data_bytes)

        return 1

    except BadSignatureError:
        print("BAD SIGNATURE")

        return 0

    #finally:
        #return valid_signature
```

Figure 5: Digital Signature Validation

The data signature and the verification that was generated from the above functions were then appended into a data frame as shown in figure 6.



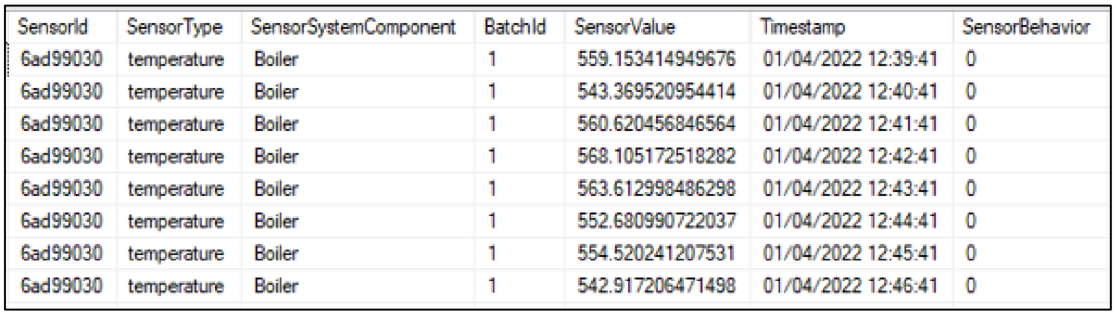
```
sig = append_with_Signature(data_string, self.priv_key, self.pub_key)
verification_val = validSignatureInt(sig, self.pub_key, data_string)
#Get hash value and verification here
temp = pd.Series([self.sensor_ID, self.type, self.sys_component, self.batch_num, output_val, t_str, self.sensor_behavior, sig, verification_val],
index = cols)
df = df.append(temp, ignore_index = True)
```

Figure 6: Appending Data Frame with Digital Signature

This method provided a way to uniquely identify sensors based on their unique characteristics and provided a technique to prove the authenticity of the sensors by using digital signatures.

3.2.5 Dataset Features:

The dataset contained six features and one label as shown in figure 7. The label was represented with values of ‘0’ and ‘1’ to signal if the sensor value was normal or malicious, respectively. Moreover, each record in the dataset was composed of a specific sensor ID, that belonged to a system component (also included as a feature), and the type of sensor which indicates if it is a temperature, pressure, vibration, gas, etc. The sensor value was a float data type value that is the output of the sensor at the designated timestamp (another feature of the dataset).



SensorId	SensorType	SensorSystemComponent	BatchId	SensorValue	Timestamp	SensorBehavior
6ad99030	temperature	Boiler	1	559.153414949676	01/04/2022 12:39:41	0
6ad99030	temperature	Boiler	1	543.369520954414	01/04/2022 12:40:41	0
6ad99030	temperature	Boiler	1	560.620456846564	01/04/2022 12:41:41	0
6ad99030	temperature	Boiler	1	568.105172518282	01/04/2022 12:42:41	0
6ad99030	temperature	Boiler	1	563.612998486298	01/04/2022 12:43:41	0
6ad99030	temperature	Boiler	1	552.680990722037	01/04/2022 12:44:41	0
6ad99030	temperature	Boiler	1	554.520241207531	01/04/2022 12:45:41	0
6ad99030	temperature	Boiler	1	542.917206471498	01/04/2022 12:46:41	0

Figure 7: Example of Generated Records for Training and Testing datasets

3.2.6 Dataset Description:

Two datasets were created to develop the anomaly detection pipeline using machine learning algorithms. The first dataset was the training dataset, consisting of 40,000 records and containing only normal temperature sensor values. The testing dataset contained 8,000 examples with 80% of them labeled as “normal” data and the rest labeled as “malicious data.” The testing dataset was later used for prediction / anomaly detection.

Each record contained the type of sensor, the system component, the sensor ID, the batch ID, the sensor value, and the timestamp. These features were selected to identify each output by its sensor

ID. This sensor, if normal, would later be enrolled in the IOTA framework, that is why all the previous data was collected. The sensor would output a value every minute, and this data could be either malicious or normal.

The training and testing datasets were imported into Python scripts to train the machine learning models for anomaly detection. In the next step, we generated and inserted the sensor data into a SQL database table so that the data could be queried by different services. The table could be queried by the web application to display information about the different sensors, it could be requested by stored procedures in the backend of the web application to be passed through the machine learning algorithms, or it could be requested by stored procedures to send the data to the IOTA Tangle.

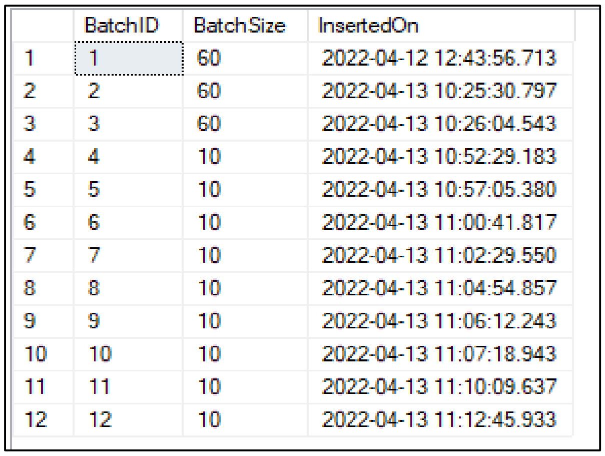
3.3 Data Aggregation and Anomaly Detection:

The data was aggregated using SQL Server with the data residing in database tables. This also served as a backend implementation for the web application where the backend requests were implemented as stored procedure queries.

The stored procedures consist of an embedded script called "sp_execute_external_scripts", which allows to run code from a scripting language like Python. The scripts processed the data from the tables, generated data from the tables, and passed the data to the web application.

3.3.1 Database Tables:

For generating the data and storing it in the database, three main tables were used. One to store the sensor generated data, another one to keep track of the number of batches generated, and the final one to store information about the system components of the power plant.

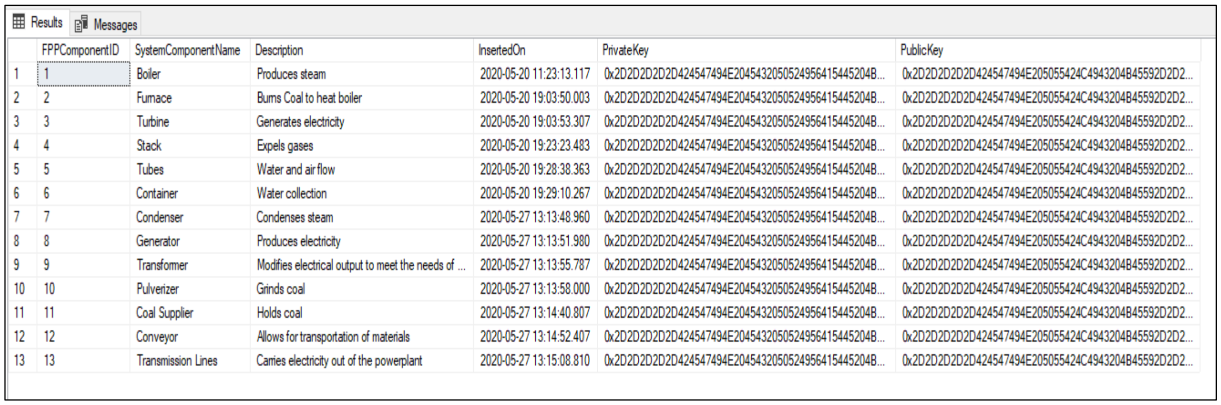


	BatchID	BatchSize	InsertedOn
1	1	60	2022-04-12 12:43:56.713
2	2	60	2022-04-13 10:25:30.797
3	3	60	2022-04-13 10:26:04.543
4	4	10	2022-04-13 10:52:29.183
5	5	10	2022-04-13 10:57:05.380
6	6	10	2022-04-13 11:00:41.817
7	7	10	2022-04-13 11:02:29.550
8	8	10	2022-04-13 11:04:54.857
9	9	10	2022-04-13 11:06:12.243
10	10	10	2022-04-13 11:07:18.943
11	11	10	2022-04-13 11:10:09.637
12	12	10	2022-04-13 11:12:45.933

Figure 8: Generated Sensor Parameters

The purpose of the Sensor Batches table was to keep track of the number of batches of the generated sensor data. Each sensor data record is associated with one batch number as shown in figure 8, which is a unique ID for the entire batch of artificial sensor data. A stored procedure was created that generated the random sensor data and obtained the batch number (referred to in the table as BatchID) by inserting a new size into the table. The insertion of a new record generated a new primary key, the BatchID, which was then queried by another stored procedure to generate a new batch of sensor data.

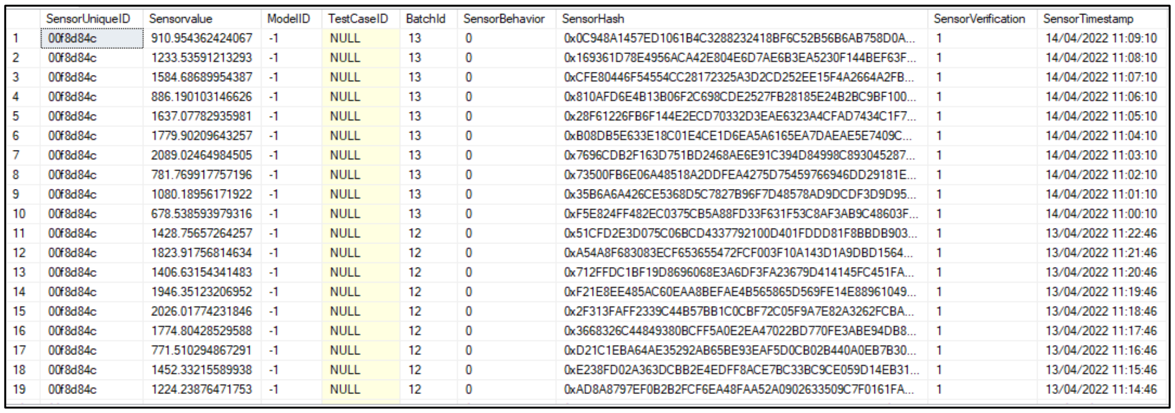
On the other hand, the system component table contained the data about all the system components of the power plant. Each record had a FPPComponentID column as shown in figure 9, which is the primary key of the table, the name of the component with a brief description, an InsertedOn column with the timestamp, which showed when the system component record was generated, and lastly 2 columns that represent the Private Key and Public Key. Each component had a pair of cryptographic keys which were used to sign every sensor value generated by the sensors belonging to that component. For instance, if a temperature sensor of the boiler is used to capture 60 different records, the keys from the boiler component will be used to sign and verify the signature of those 60 records.



	FPPComponentID	SystemComponentName	Description	InsertedOn	PrivateKey	PublicKey
1	1	Boiler	Produces steam	2020-05-20 11:23:13.117	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
2	2	Furnace	Burns Coal to heat boiler	2020-05-20 19:03:50.003	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
3	3	Turbine	Generates electricity	2020-05-20 19:03:53.307	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
4	4	Stack	Expels gases	2020-05-20 19:23:23.483	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
5	5	Tubes	Water and air flow	2020-05-20 19:28:38.363	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
6	6	Container	Water collection	2020-05-20 19:29:10.267	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
7	7	Condenser	Condenses steam	2020-05-27 13:13:48.960	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
8	8	Generator	Produces electricity	2020-05-27 13:13:51.980	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
9	9	Transformer	Modifies electrical output to meet the needs of ...	2020-05-27 13:13:55.787	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
10	10	Pulverizer	Grinds coal	2020-05-27 13:13:58.000	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
11	11	Coal Supplier	Holds coal	2020-05-27 13:14:40.807	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
12	12	Conveyor	Allows for transportation of materials	2020-05-27 13:14:52.407	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...
13	13	Transmission Lines	Carries electricity out of the powerplant	2020-05-27 13:15:08.810	0x2D2D2D2D2D424547494E20454320505249564154452048...	0x2D2D2D2D2D424547494E205055424C4943204B45592D2D2...

Figure 9: FPP Component table example

Similarly, a table was created to store the actual generated sensor data as shown in figure 10. The structure of the table is like the previously mentioned tables generated to train the machine learning model. The difference is that this table contained four extra columns, one column stored the signature hash of each reading, and one contained the output of signature verification. The SensorVerification column had either a '1' or '0' value depending on whether the signature verification was successful or not. The ModelID and TestCaseID columns were used for the machine learning models for anomaly detection.



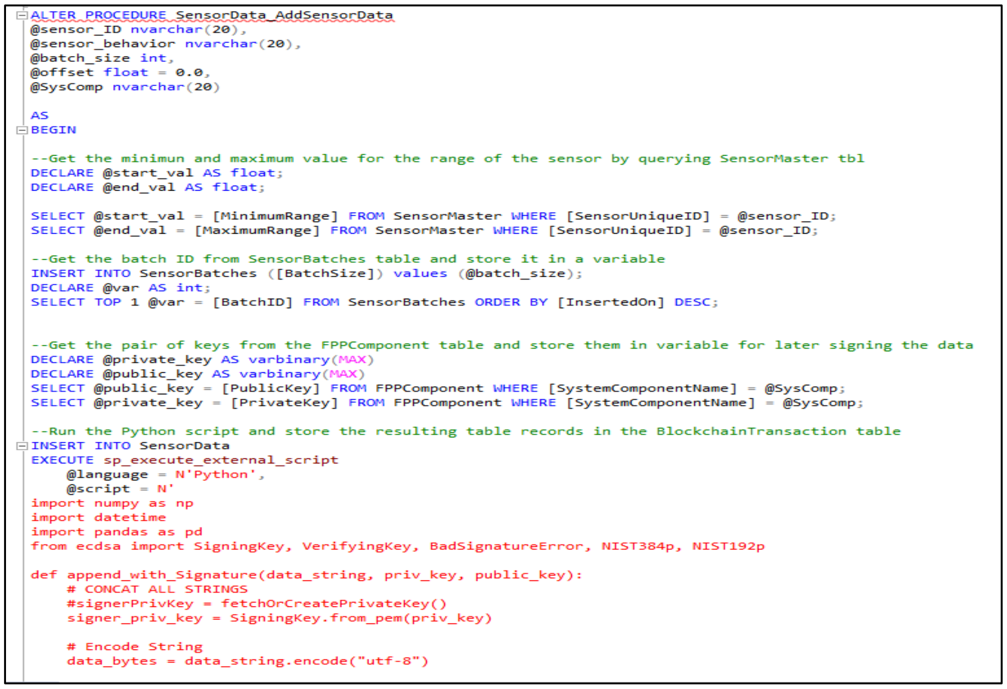
	SensorUniqueID	Sensorvalue	ModelID	TestCaseID	BatchId	SensorBehavior	SensorHash	SensorVerification	SensorTimestamp
1	00f8d84c	910.954362424067	-1	NULL	13	0	0x0C948A1457ED1061B4C3288232418BF6C52B56B6AB758D0A...	1	14/04/2022 11:09:10
2	00f8d84c	1233.53591213293	-1	NULL	13	0	0x169361D78E4956ACA42E804E6D7AE6B3EA5230F144BEF63F...	1	14/04/2022 11:08:10
3	00f8d84c	1584.68689954387	-1	NULL	13	0	0xCFE80446F54554CC28172325A3D2CD252EE15F4A2664A2FB...	1	14/04/2022 11:07:10
4	00f8d84c	886.190103146626	-1	NULL	13	0	0x810AFD6E4B13B06F2C698CDE2527FB28185E24B2BC9BF100...	1	14/04/2022 11:06:10
5	00f8d84c	1637.07782935981	-1	NULL	13	0	0x28F61226FB6F144E2ECD70332D3EAE6323A4CFAD7434C1F7...	1	14/04/2022 11:05:10
6	00f8d84c	1779.90209643257	-1	NULL	13	0	0xB08DB5E633E18C01E4CE1D6EA5A6165EA7DAEAE5E7409C...	1	14/04/2022 11:04:10
7	00f8d84c	2089.02464984505	-1	NULL	13	0	0x7696CDB2F163D751BD2468AE6E91C394D84998C893045287...	1	14/04/2022 11:03:10
8	00f8d84c	781.769917757196	-1	NULL	13	0	0x73500FB6E06A48518A2DDFEA4275D75459766946DD29181E...	1	14/04/2022 11:02:10
9	00f8d84c	1080.18956171922	-1	NULL	13	0	0x35B6A6A426CE5368D5C7827B96F7D48578AD9DCDF3D9D95...	1	14/04/2022 11:01:10
10	00f8d84c	678.538593979316	-1	NULL	13	0	0xF5E824FF482EC0375CB5A88FD33F631F53C8AF3AB9C48603F...	1	14/04/2022 11:00:10
11	00f8d84c	1428.75657264257	-1	NULL	12	0	0x51CFD2E3D075C06BCD4337792100D401FDDD81F8BBDB903...	1	13/04/2022 11:22:46
12	00f8d84c	1823.91756814634	-1	NULL	12	0	0xA54A8F683083ECF653655472FCF003F10A143D1A9DBD1564...	1	13/04/2022 11:21:46
13	00f8d84c	1406.63154341483	-1	NULL	12	0	0x712FFDC1BF19D8696068E3A6DF3FA23679D414145FC451FA...	1	13/04/2022 11:20:46
14	00f8d84c	1946.35123206952	-1	NULL	12	0	0xF21E8EE485AC60EAA8BEFAE4B565865D569FE14E88961049...	1	13/04/2022 11:19:46
15	00f8d84c	2026.01774231846	-1	NULL	12	0	0x2F313FAFF2339C44B57BB1C0CBF72C05F9A7E82A3262FCBA...	1	13/04/2022 11:18:46
16	00f8d84c	1774.80428529588	-1	NULL	12	0	0x3668326C44849380BCFF5A0E2EA47022BD770FE3ABE94DB8...	1	13/04/2022 11:17:46
17	00f8d84c	771.510294867291	-1	NULL	12	0	0xD21C1EBA64AE35292AB65BE93EAF5D0CB02B440A0EB7B30...	1	13/04/2022 11:16:46
18	00f8d84c	1452.33215589938	-1	NULL	12	0	0xE238FD02A363DCBB2E4EDFF8ACE7BC33BC9CE059D14EB31...	1	13/04/2022 11:15:46
19	00f8d84c	1224.23876471753	-1	NULL	12	0	0xAD8A8797EF0B2B2FCF6EA48FAA52A0902633509C7F0161FA...	1	13/04/2022 11:14:46

Figure 10: Sensor Data table example

3.3.2 Stored Procedures:

Various stored procedures were created to generate, insert, update, and delete records in different tables. These stored procedures can be divided into the ones that modify the sensor table and the ones that modify the FPPComponent table:

1) SensorData_AddSensorData: This stored procedure generated a batch of data for a specified component. It required five parameters: the sensor unique ID (nvarchar), the sensor behavior ('1' for malicious and '0' for normal), the batch size, the offset if the sensor is malicious (for normal sensors, there was no need to pass a value for this parameter), and the system component in which the sensor belongs.

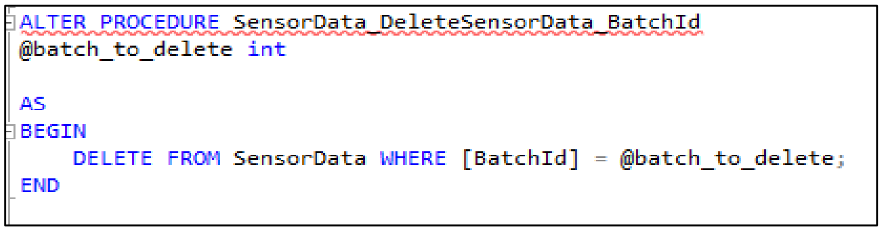


```
ALTER PROCEDURE SensorData_AddSensorData
@sensor_ID nvarchar(20),
@sensor_behavior nvarchar(20),
@batch_size int,
@offset float = 0.0,
@SysComp nvarchar(20)

AS
BEGIN

--Get the minimun and maximum value for the range of the sensor by querying SensorMaster tbl
DECLARE @start_val AS float;
DECLARE @end_val AS float;

SELECT @start_val = [MinimumRange] FROM SensorMaster WHERE [SensorUniqueID] = @sensor_ID;
SELECT @end_val = [MaximumRange] FROM SensorMaster WHERE [SensorUniqueID] = @sensor_ID;

--Get the batch ID from SensorBatches table and store it in a variable
INSERT INTO SensorBatches ([BatchSize]) values (@batch_size);
DECLARE @var AS int;
SELECT TOP 1 @var = [BatchID] FROM SensorBatches ORDER BY [InsertedOn] DESC;


--Get the pair of keys from the FPPComponent table and store them in variable for later signing the data
DECLARE @private_key AS varbinary(MAX)
DECLARE @public_key AS varbinary(MAX)
SELECT @public_key = [PublicKey] FROM FPPComponent WHERE [SystemComponentName] = @SysComp;
SELECT @private_key = [PrivateKey] FROM FPPComponent WHERE [SystemComponentName] = @SysComp;

--Run the Python script and store the resulting table records in the BlockchainTransaction table
INSERT INTO SensorData
EXECUTE sp_execute_external_script
    @language = N'Python',
    @script = N'
import numpy as np
import datetime
import pandas as pd
from ecdsa import SigningKey, VerifyingKey, BadSignatureError, NIST384p, NIST192p

def append_with_Signature(data_string, priv_key, public_key):
    # CONCAT ALL STRINGS
    #signerPrivKey = fetchOrCreatePrivateKey()
    signer_priv_key = SigningKey.from_pem(priv_key)

    # Encode String
    data_bytes = data_string.encode("utf-8")
```

Figure 11: SensorData_AddSensorData Stored Procedure

The procedure as shown in figure 11, queried the Sensor Batches table to obtain the BatchId and the FPPComponent table to obtain the pair of keys. It also queried the minimum and maximum sensor values from the SensorMaster table based on the sensor ID. Then, the Python script generated a random sensor value for as many times as specified with timestamps separated by 1 minute from the previous record, starting at the specific instance of time the stored procedure was called. Once the data was inserted into the table, the SensorHash and SensorVerification were generated for each record.

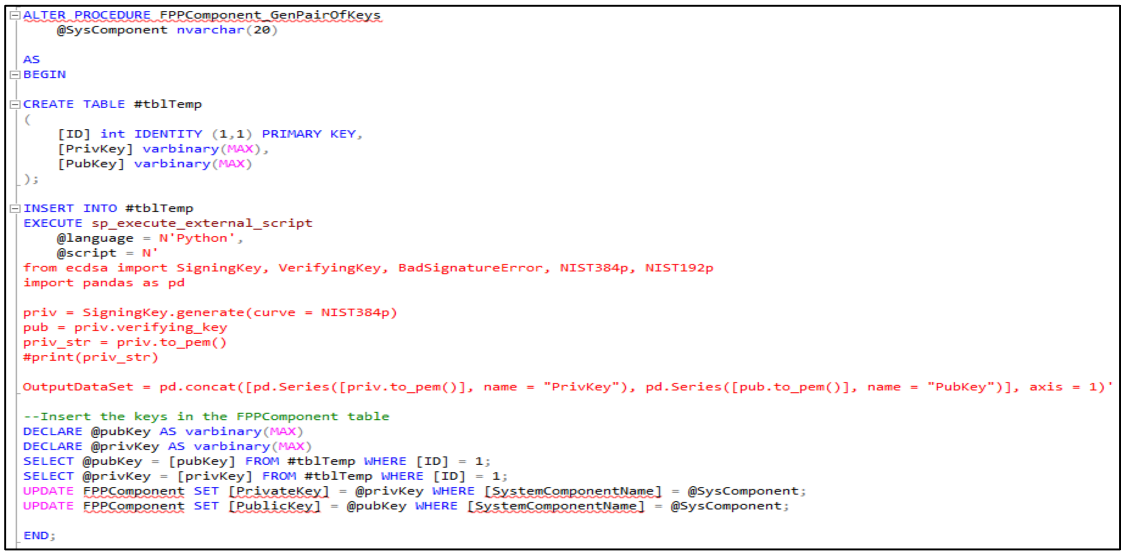
2) SensorData_DeleteSensorData_BatchId: This procedure as shown in figure 12 was created to delete all records from the SensorData table that matched the BatchId value, and this value was unique for every record inside this table. The only parameter it required was the batch ID to delete the records.



```sql
ALTER PROCEDURE SensorData_DeleteSensorData_BatchId
@batch_to_delete int

AS
BEGIN
    DELETE FROM SensorData WHERE [BatchId] = @batch_to_delete;
END
```

Figure 12: SensorData_DeleteSensorData_BatchId Stored Procedure

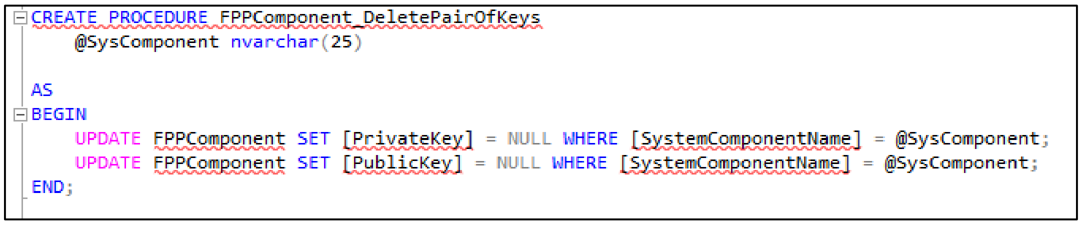
3) FPPComponent_GenPairOfKeys: This stored procedure as shown in figure 13 created a public and private key specifically for the system component which was passed as an argument (nvarchar). It used a Python script to generate the keys and store them in the proper table as varbinary (MAX) values.



```sql
ALTER PROCEDURE FPPComponent_GenPairOfKeys
    @SysComponent nvarchar(20)

AS
BEGIN

CREATE TABLE #tblTemp
(
    [ID] int IDENTITY (1,1) PRIMARY KEY,
    [PrivKey] varbinary(MAX),
    [PubKey] varbinary(MAX)
);

INSERT INTO #tblTemp
EXECUTE sp_execute_external_script
    @language = N'Python',
    @script = N'
from ecdsa import SigningKey, VerifyingKey, BadSignatureError, NIST384p, NIST192p
import pandas as pd

priv = SigningKey.generate(curve = NIST384p)
pub = priv.verifying_key
priv_str = priv.to_pem()
#print(priv_str)

OutputDataSet = pd.concat([pd.Series([priv.to_pem()], name = "PrivKey"), pd.Series([pub.to_pem()], name = "PubKey")], axis = 1)'

--Insert the keys in the FPPComponent table
DECLARE @pubKey AS varbinary(MAX)
DECLARE @privKey AS varbinary(MAX)
SELECT @pubKey = [pubKey] FROM #tblTemp WHERE [ID] = 1;
SELECT @privKey = [privKey] FROM #tblTemp WHERE [ID] = 1;
UPDATE FPPComponent SET [PrivateKey] = @privKey WHERE [SystemComponentName] = @SysComponent;
UPDATE FPPComponent SET [PublicKey] = @pubKey WHERE [SystemComponentName] = @SysComponent;

END;
```

Figure 13: FPPComponent_GenPairOfKeys stored procedure

4) FPPComponent_DeletePairOfKeys: This stored procedure as shown in figure 14 replaced the values of a specified system component (nvarchar) with NULL on the Public and Private Key columns.



```
CREATE PROCEDURE FPPComponent_DeletePairOfKeys
    @SysComponent nvarchar(25)

AS
BEGIN
    UPDATE FPPComponent SET [PrivateKey] = NULL WHERE [SystemComponentName] = @SysComponent;
    UPDATE FPPComponent SET [PublicKey] = NULL WHERE [SystemComponentName] = @SysComponent;
END;
```

Figure 14: FPPComponent_DeletePairOfKeys stored procedure

3.3.3 Machine Learning Algorithms for Anomaly Detection:

The normal dataset was used to train three anomaly detection algorithms which were semi-supervised / unsupervised algorithms are implemented. The models were trained only with normal data, and when predicting data with anomalies, they were able to detect the malicious data or outliers. The models that were tested were the Local Outlier Factor, Agglomerative Clustering, and AutoEncoder for Anomaly Detection.

3.3.4 Algorithms:

The first implemented model was the Local Outlier Factor. This model measured the local deviation density of a given sample with respect to its neighbors. Samples that have substantially lower density and their neighbors were labeled as anomalies as shown in figure 15. The implementation utilized the LocalOutlierFactor class from the scikit-learn library. The parameters passed were contamination of 1% in the training dataset and 10 as the number of neighbors to consider. Additionally, since the training data only contained normal points, the novelty parameter was set to True.

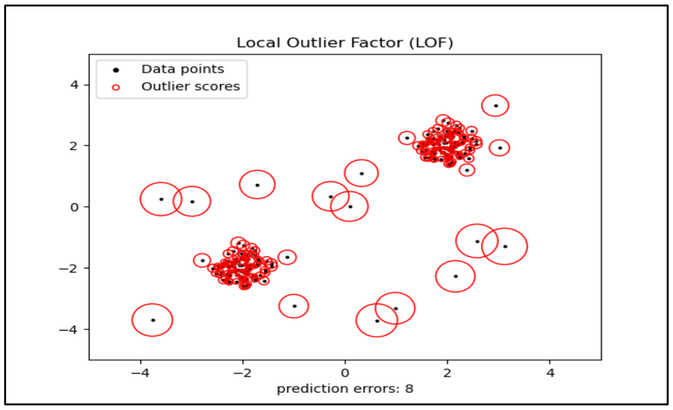


Figure 15: Example of Local Outlier Factor Graph, with Clear Outliers and Clusters

The Agglomerative Clustering algorithm was employed to make predictions on anomalous data. It uses linkage distance to recursively merge pairs of clusters of data. The number of clusters to find was set to ‘2’ and the linkage was set to ‘ward’ to compute the Euclidean distance.

The testing dataset was divided into two clusters, these clusters represent the anomalies and the normal data. With this knowledge, it was assumed that the distance of the anomalies to clusters could be computed recursively.

An AutoEncoder for anomaly detection is shown in figure 16. An AutoEncoder is a neural network that encodes some data to learn its structure, then it is decoded and the difference between the input and the output is regarded as the loss.

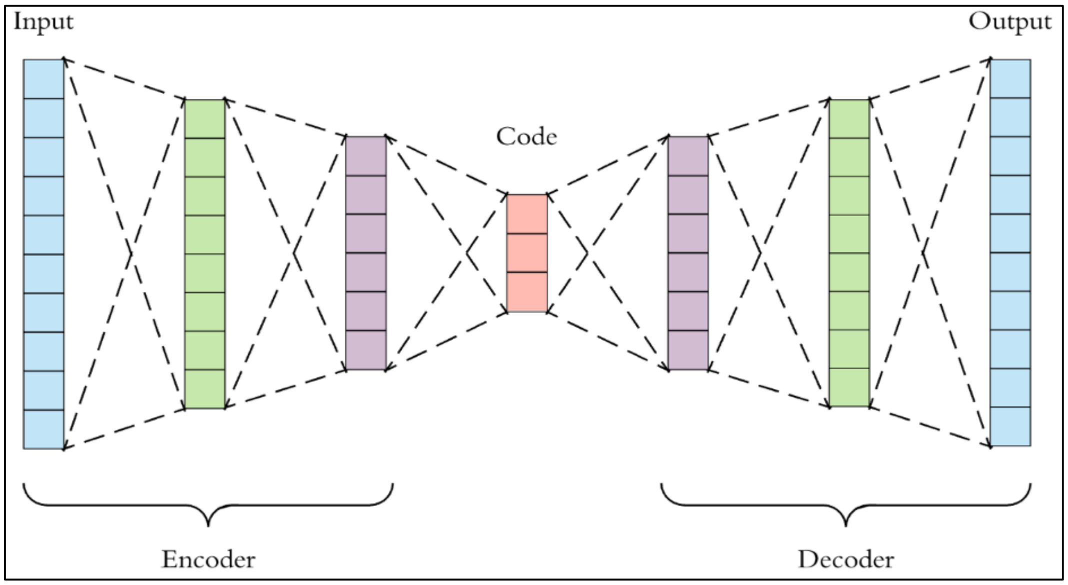


Figure 16: Basic AutoEncoder Neural Network

The use of loss can be leveraged to identify anomalies in a dataset. For the regular use of AutoEncoders, models need to be generalized by minimizing the loss of testing data. However, in this scenario where the model was trained using normal data, it is expected that the loss of anomalous data points would be bigger.

A simple decision model was designed in terms of the distribution of the loss. After observing the distribution of the loss, a lower and upper threshold was determined to detect the anomalies, consisting of the mean of the loss plus/minus 2 standard deviations. The loss distribution of the training and testing datasets is shown in figure 17.

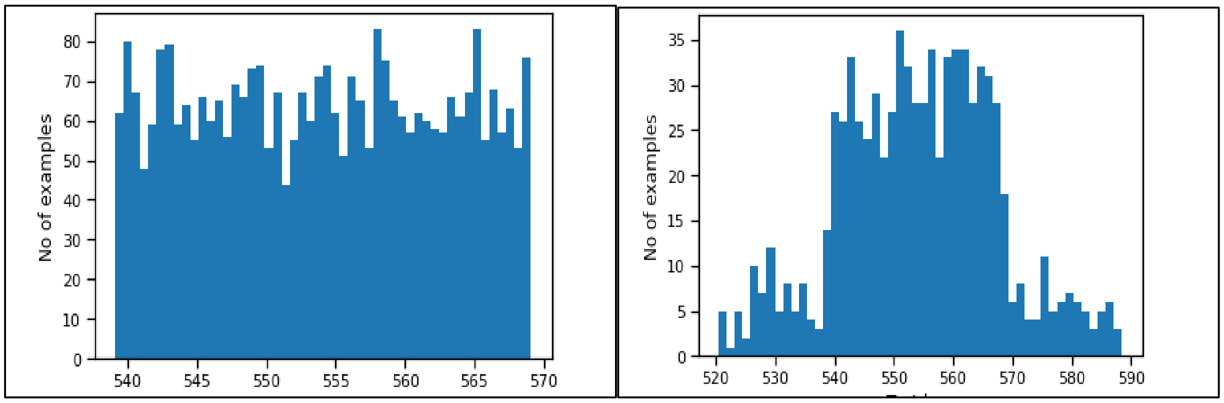
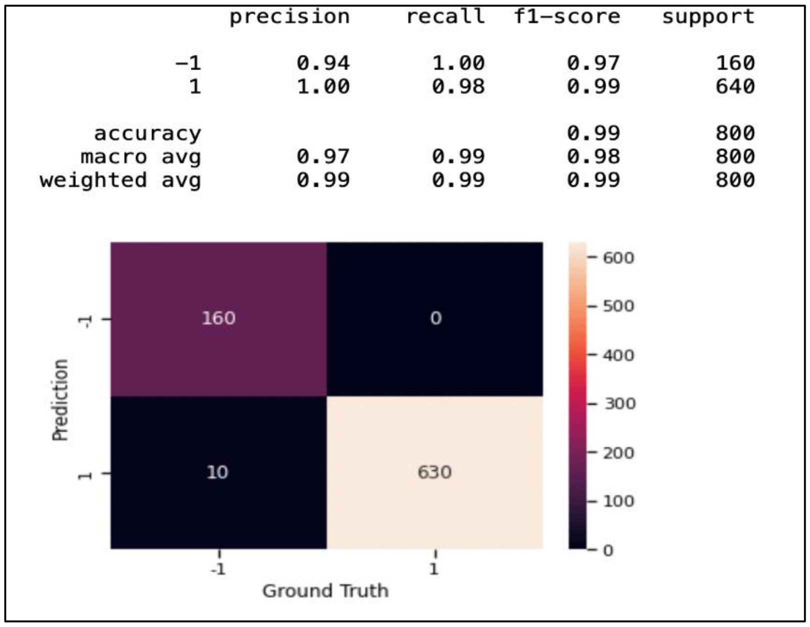


Fig. 17: Loss distribution of the training (left) and testing (right) datasets.

3.3.5 Results:

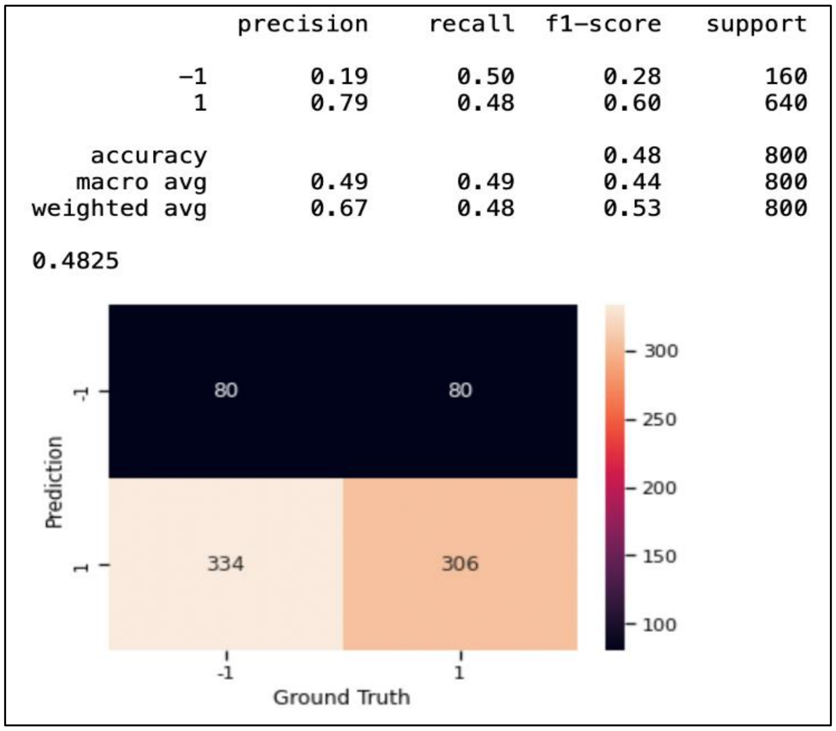
The Local Outlier Factor model prediction on the test dataset produced good results, with a **98.75%** accuracy on the testing set and **99.44%** accuracy on the training set. This shows that the model is not overfitting or underfitting. As seen in figure 18, the confusion matrix demonstrates that only 10 predictions were inaccurate.



```
              precision    recall  f1-score   support

          -1       0.94      1.00      0.97       160
           1       1.00      0.98      0.99       640

    accuracy                           0.99       800
   macro avg       0.97      0.99      0.98       800
weighted avg       0.99      0.99      0.99       800
```

Figure 18: Confusion Matrix and Performance Metrics for the LOF Model

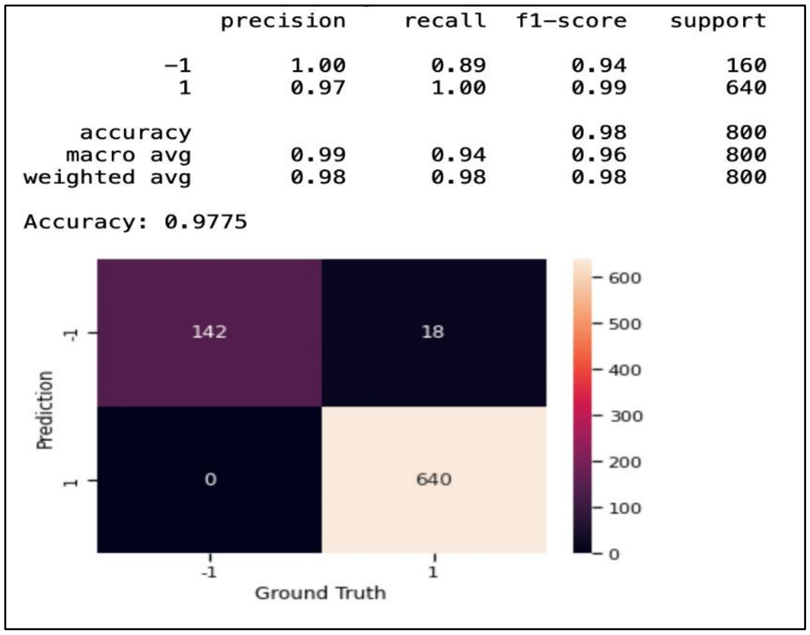
Conversely, the agglomerative clustering model did not perform well as shown in figure 19. When predicting the testing dataset. After different runs, the best accuracy turned out to be **48.25%** with complete linkage and computing of the Manhattan distance. The results leave this algorithm out of the possibilities of being considered for the study.



```
              precision    recall  f1-score   support

          -1       0.19      0.50      0.28       160
           1       0.79      0.48      0.60       640

    accuracy                           0.48       800
   macro avg       0.49      0.49      0.44       800
weighted avg       0.67      0.48      0.53       800

0.4825
```

Figure 19: Confusion Matrix and Performance Metrics for the Agglomerative Clustering

The AutoEncoder model as shown in figure 20 scored well in the different metrics, demonstrating an accuracy of **97.75%** and precision of 100% and 97% for the anomaly and normal class, respectively. This shows how powerful deep learning algorithms are when implementing unsupervised and semi-supervised models.



```
              precision    recall  f1-score   support

          -1       1.00      0.89      0.94       160
           1       0.97      1.00      0.99       640

    accuracy                           0.98       800
   macro avg       0.99      0.94      0.96       800
weighted avg       0.98      0.98      0.98       800

Accuracy: 0.9775
```

Figure 20: Confusion Matrix and Performance Metrics for the AutoEncoder Model

3.4 Secure Data Logging with Blockchain:

The fossil fuel power plant data was aggregated and analyzed to detect anomalies using machine learning and implement secure logging of sensor data using blockchain. In order to implement secure logging with blockchain, a SHA-256 checksum of the aggregated data was generated. Once the checksum was generated, the next step was to insert the sensor metadata to the blockchain platform for secure storage. A message ID was generated after the sensor meta data was uploaded to the blockchain nodes. This message ID is then used to fetch this data from the blockchain database whenever required. For the blockchain implementation, the IOTA distributed ledger was chosen due to its high scalability and zero transaction fees.

3.4.1 Blockchain Architecture:

To implement the blockchain platform, the IOTA framework was implemented, which is a Directed Acyclic Graph (DAG)-based blockchain. To preserve privacy and achieve high scalability, a private Tangle was constructed with three private nodes as shown in figure 21.

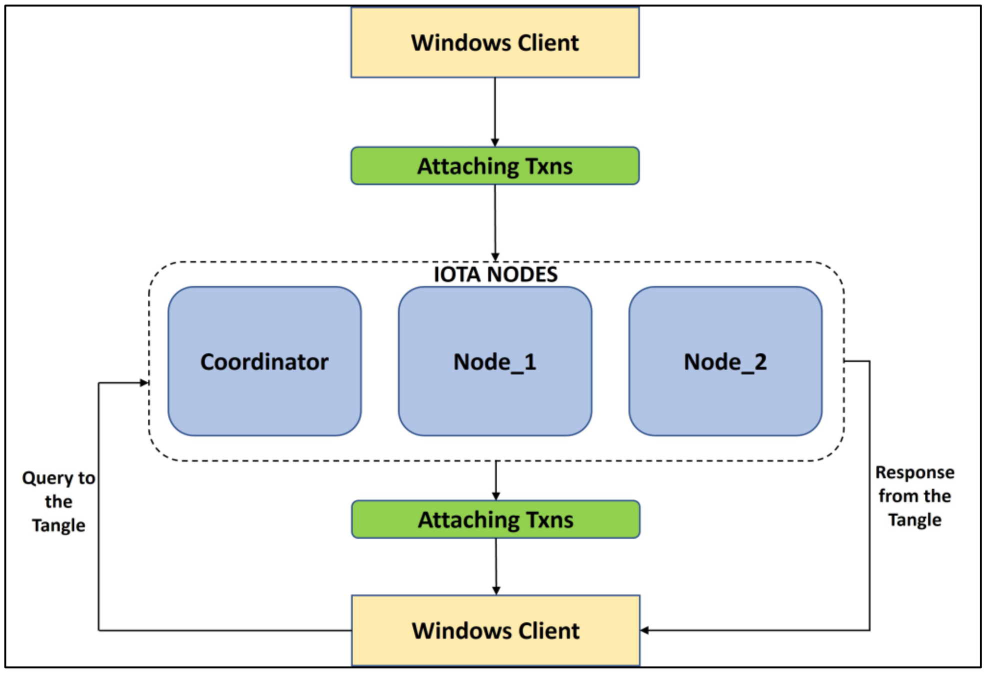
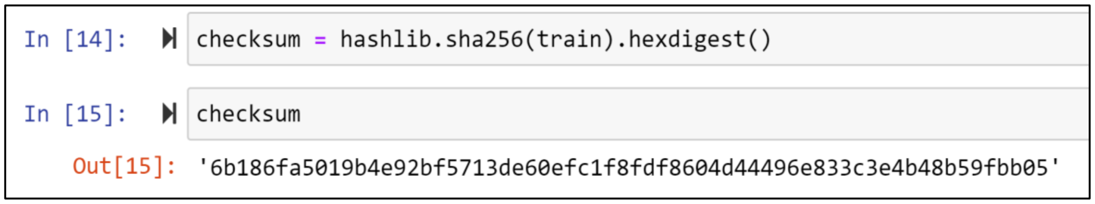


Figure 21: IOTA System Architecture

A private Tangle was created, which consisted of three full nodes, called the coordinator, and two neighbor nodes, namely, “Node_1” and “Node_2”. All three nodes were set up on three Linux Ubuntu servers with Hornet installed on them. Hornet is a powerful, community-driven IOTA node software written in the Go language and is a lightweight alternative to the IOTA reference implementation (IRI). Hornet was developed for the secure transfer of tokens or data, and for experimenting and implementing IOTA protocols between nodes or network participants. Machines can act as a node and connect to the IOTA network with the help of the Hornet software. These nodes or machines have functions such as authenticating the transactions, storing these authenticated transactions on the Tangle, and fetching these transactions back from the Tangle whenever required.

3.4.2 Checksum Generation:

The SHA-256 hashing algorithm which stands for "Secure Hashing Algorithm" to calculate the checksum was implemented. This algorithm was implemented in Python by using the hashlib library in the script called IOTA_Send_Data.py. We used this SHA-256 algorithm to calculate the checksum of the generated and aggregated power plant sensor data. The checksum generated was 256-bit in size and hence the name of the algorithm is SHA-256. The output of this algorithm can be seen in figure 22.



```
In [14]:  checksum = hashlib.sha256(train).hexdigest()

In [15]:  checksum
  Out[15]: '6b186fa5019b4e92bf5713de60efc1f8fdf8604d44496e833c3e4b48b59fbb05'
```

Figure 22: Implementation of SHA-256 Algorithm

The SHA-256 algorithm is a cryptographic hashing algorithm that always outputs a 256-bit hash. Whatever the size of the input is, the output will always be 256-bit in size. The SHA-256 belongs to the SHA-2 family of algorithms. The working of the SHA-256 hash function can be seen in figure 23.

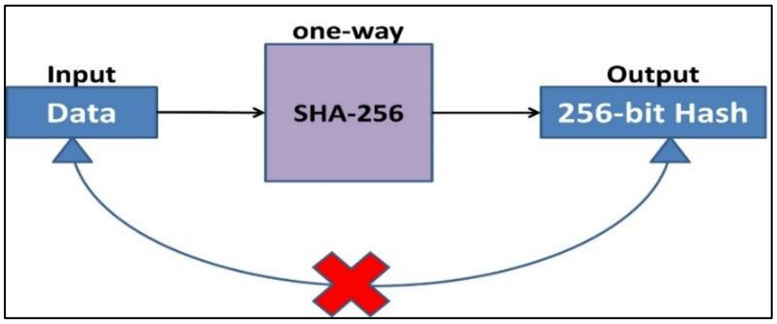


Figure 23: SHA-256 Working

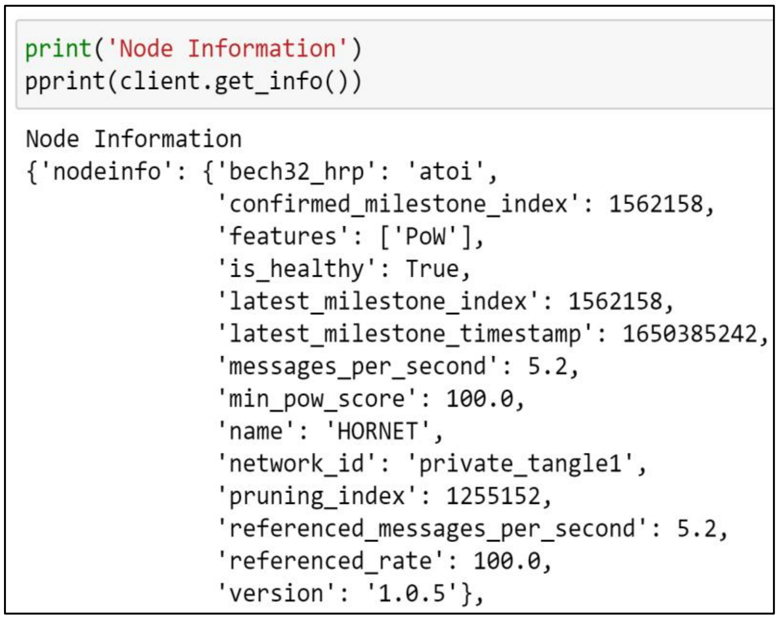
As shown in figure 23, the SHA-256 operates by taking input data and calculating the checksum, which is always 256-bit in size. This hash function is irreversible, meaning it is infeasible to obtain the input from the output. No matter how many times the input is hashed the output hash will always be the same in every run unless the data is changed or tampered with. Even a very minute change in the input will result in a completely different hash, thus notifying the user that the input data has been altered. Therefore, using this hashing method we can ensure the authenticity of the data by calculating the hash before transmission and after transmission. If the hash is the same on both sides, then we can confirm that the data which was in motion has not been tampered with.

3.4.3 Secure Storage of Sensor Metadata:

A Python script was developed and implemented named IOTA_Send_Data.py, which consisted of the following subtasks:

1. Imports the required libraries.
2. Reads the simulated power plant sensor data.
3. Generates a 256-bit checksum of the entire aggregated data.
4. Establishes a connection with the private IOTA nodes.
5. Calculates some statistical measures about the aggregated data.
6. Creates a JSON structure containing the sensor metadata.
7. Inserts and stores the JSON data into the IOTA framework.

The Python script reads the simulated sensor data from the database and calculates the aggregated 256-bit hash of the complete batch of data by using the SHA-256 algorithm.



```
print('Node Information')
pprint(client.get_info())
```

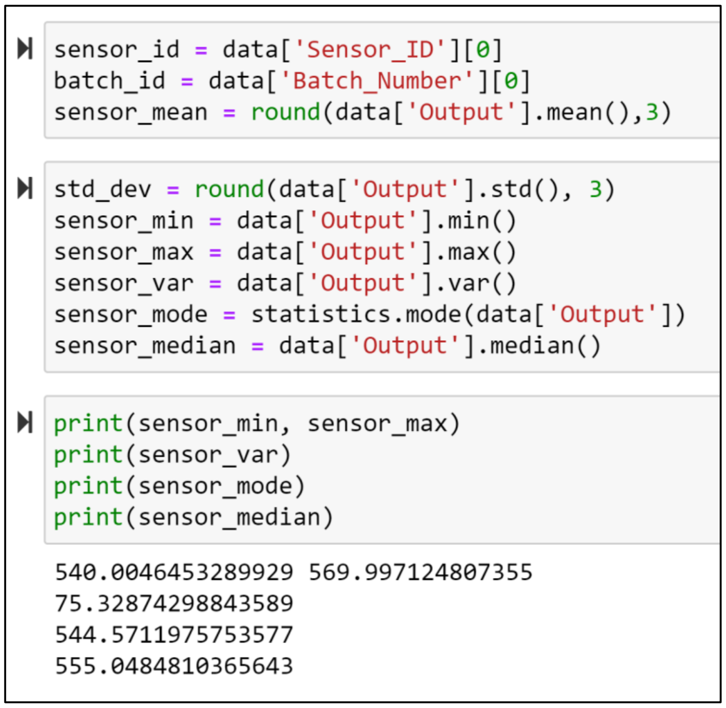
```
Node Information
{'nodeinfo': {'bech32_hrp': 'atoi',
              'confirmed_milestone_index': 1562158,
              'features': ['PoW'],
              'is_healthy': True,
              'latest_milestone_index': 1562158,
              'latest_milestone_timestamp': 1650385242,
              'messages_per_second': 5.2,
              'min_pow_score': 100.0,
              'name': 'HORNET',
              'network_id': 'private_tangle1',
              'pruning_index': 1255152,
              'referenced_messages_per_second': 5.2,
              'referenced_rate': 100.0,
              'version': '1.0.5'},
```

Figure 24: Node Information of IOTA Nodes

A connection was then established with the IOTA framework by employing the official IOTA Python library called "IOTA-client". We used this library's method called "client" which makes the connection with the IOTA nodes by passing the IP address and port number of the node to connect. Once the connection was successful, checked if the nodes are healthy and synced by acquiring the node information as shown in figure 24.

The figure 24, shows the information about the status of the nodes and the private Tangle. It can be seen that the node is healthy and synced with its peers with the help of this information.

The statistical analysis of the aggregated sensor data was performed to get some meaningful insights such as the distribution of the data. This was performed by calculating some statistical measures such as mean, median, mode, standard deviation, variance, minimum, and maximum of the aggregated sensor data. This can be seen in figure 25, which shows the calculation of the mentioned statistical parameters in the Python script.



```
sensor_id = data['Sensor_ID'][0]
batch_id = data['Batch_Number'][0]
sensor_mean = round(data['Output'].mean(),3)
```

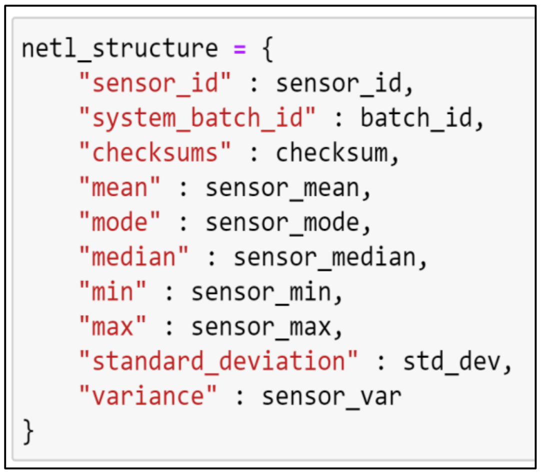
```
std_dev = round(data['Output'].std(), 3)
sensor_min = data['Output'].min()
sensor_max = data['Output'].max()
sensor_var = data['Output'].var()
sensor_mode = statistics.mode(data['Output'])
sensor_median = data['Output'].median()
```

```
print(sensor_min, sensor_max)
print(sensor_var)
print(sensor_mode)
print(sensor_median)
```

```
540.0046453289929 569.997124807355
75.32874298843589
544.5711975753577
555.0484810365643
```

Figure 25: Statistical Analysis of Sensor Data

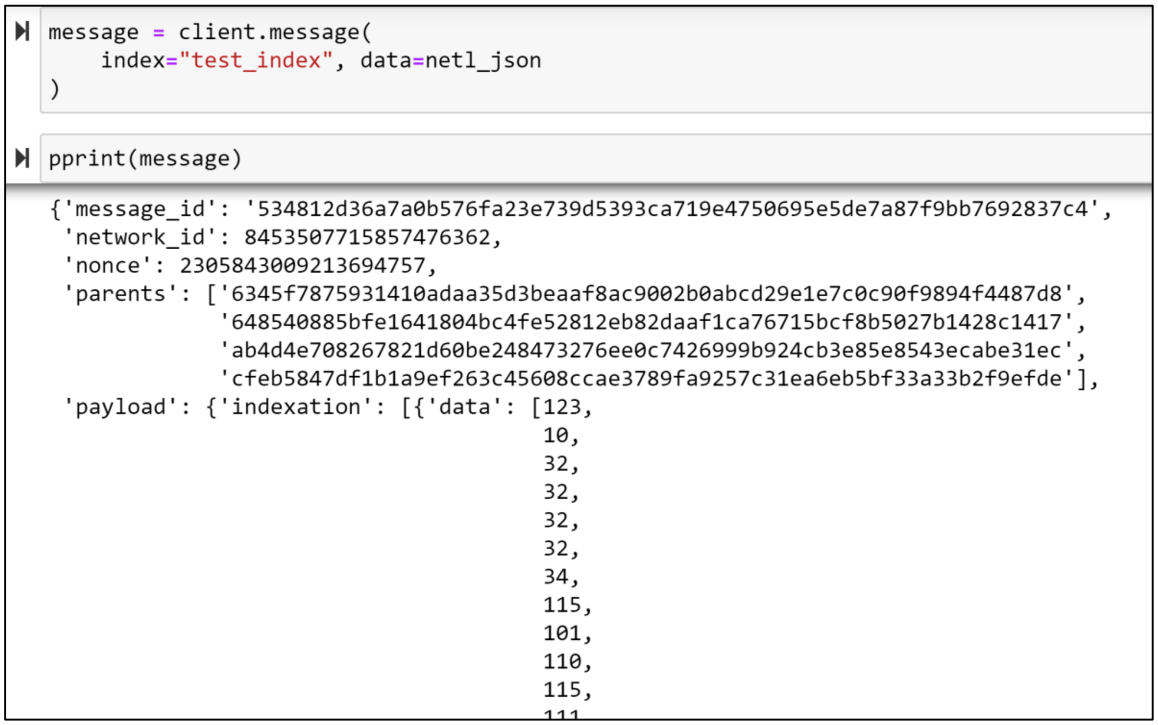
Once we performed the statistical analysis, we created a JSON structure that included all the statistical values, the sensor ID, and the sensor batch ID as shown in figure 26.



```
netl_structure = {
    "sensor_id" : sensor_id,
    "system_batch_id" : batch_id,
    "checksums" : checksum,
    "mean" : sensor_mean,
    "mode" : sensor_mode,
    "median" : sensor_median,
    "min" : sensor_min,
    "max" : sensor_max,
    "standard_deviation" : std_dev,
    "variance" : sensor_var
}
```

Figure 26: JSON Structure Inserted Into IOTA

Once the JSON structure was created, the data was published as a message transaction in the IOTA by using the client. message () function which sends and stores the transaction in the IOTA nodes and returns a message ID as shown in figure 27.



```
message = client.message(
    index="test_index", data=netl_json
)
```

```
pprint(message)
```

```
{'message_id': '534812d36a7a0b576fa23e739d5393ca719e4750695e5de7a87f9bb7692837c4',
 'network_id': 8453507715857476362,
 'nonce': 2305843009213694757,
 'parents': ['6345f7875931410adaa35d3beaaf8ac9002b0abcd29e1e7c0c90f9894f4487d8',
             '648540885bfe1641804bc4fe52812eb82daaf1ca76715bcf8b5027b1428c1417',
             'ab4d4e708267821d60be248473276ee0c7426999b924cb3e85e8543ecabe31ec',
             'cfeb5847df1b1a9ef263c45608ccae3789fa9257c31ea6eb5bf33a33b2f9efde'],
 'payload': {'indexation': [{'data': [123,
                                      10,
                                      32,
                                      32,
                                      32,
                                      32,
                                      34,
                                      115,
                                      101,
                                      110,
                                      115,
```

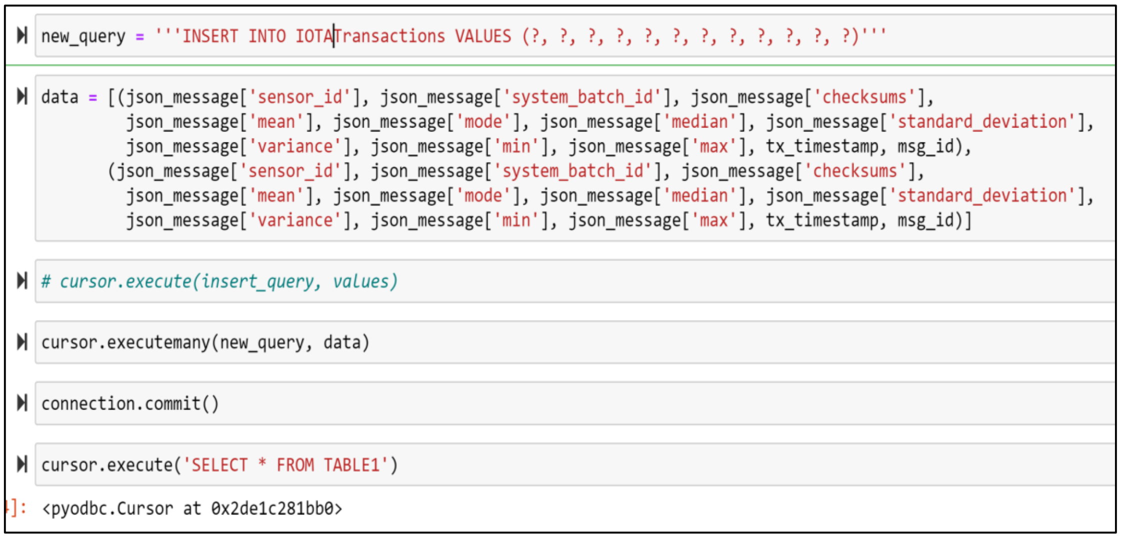
Figure 27: Sending Data to Private Tangle

3.4.4 Fetching Data from the IOTA:

Another Python script was developed called IOTA_Fetch_Data.py, which retrieves the data from the private IOTA nodes. The client machine queried the private Tangle by using the message ID where the nodes use this message ID to locate and fetch the queried transactions and return the data requested. Also, by using the message ID, we acquired the transaction timestamp which was generated when the transaction was sent to the IOTA framework.

3.4.5 IOTA Transactions Table:

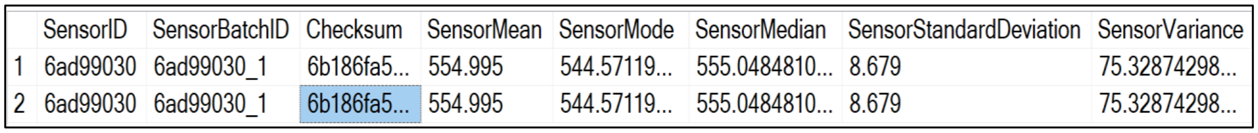
After the transactions containing the sensor metadata were fetched and retrieved in the Python script, the PYODBC driver was imported in Python to interact with the SQL database. A connection with the SQL server database was established in the Python script, as shown in figure 28, by passing the valid credentials.



```
new_query = '''INSERT INTO IOTATransactions VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)'''

data = [(json_message['sensor_id'], json_message['system_batch_id'], json_message['checksums'],
         json_message['mean'], json_message['mode'], json_message['median'], json_message['standard_deviation'],
         json_message['variance'], json_message['min'], json_message['max'], tx_timestamp, msg_id),
       (json_message['sensor_id'], json_message['system_batch_id'], json_message['checksums'],
         json_message['mean'], json_message['mode'], json_message['median'], json_message['standard_deviation'],
         json_message['variance'], json_message['min'], json_message['max'], tx_timestamp, msg_id)]

# cursor.execute(insert_query, values)

cursor.executemany(new_query, data)

connection.commit()

cursor.execute('SELECT * FROM TABLE1')

]: <pyodbc.Cursor at 0x2de1c281bb0>
```

Figure 28: Python Code for Inserting Transactional Data into Database

A table called IOTA Transactions was created which stored all the transactional data along with the transaction metadata, such as transaction timestamp and message ID, as shown in figure 29.



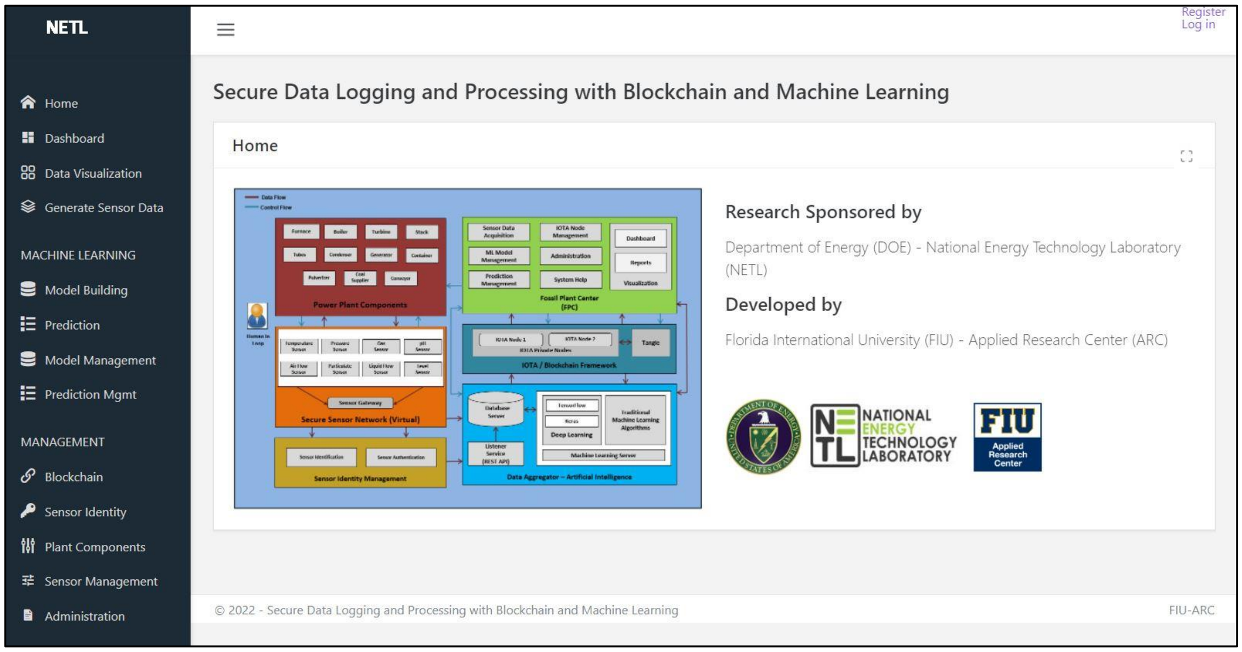
	SensorID	SensorBatchID	Checksum	SensorMean	SensorMode	SensorMedian	SensorStandardDeviation	SensorVariance
1	6ad99030	6ad99030_1	6b186fa5...	554.995	544.57119...	555.0484810...	8.679	75.32874298...
2	6ad99030	6ad99030_1	6b186fa5...	554.995	544.57119...	555.0484810...	8.679	75.32874298...

Figure 29: IOTA Transaction Table

3.5 Web Application Implementation:

A Web application was developed by the FIU team to manage the overall Secure Data Logging and Processing with Blockchain and Machine Learning framework. As seen in figure 30, the homepage consists of different modules with functionalities to generate sensor data, visualization on a graph, sensor identity management, and anomaly detection which consists of model building and prediction. Finally, the blockchain module pushes the sensor metadata into the IOTA framework for secure storage and logging.

Certain module screenshots are shown below:



Secure Data Logging and Processing with Blockchain and Machine Learning

Home

Research Sponsored by

Department of Energy (DOE) - National Energy Technology Laboratory (NETL)

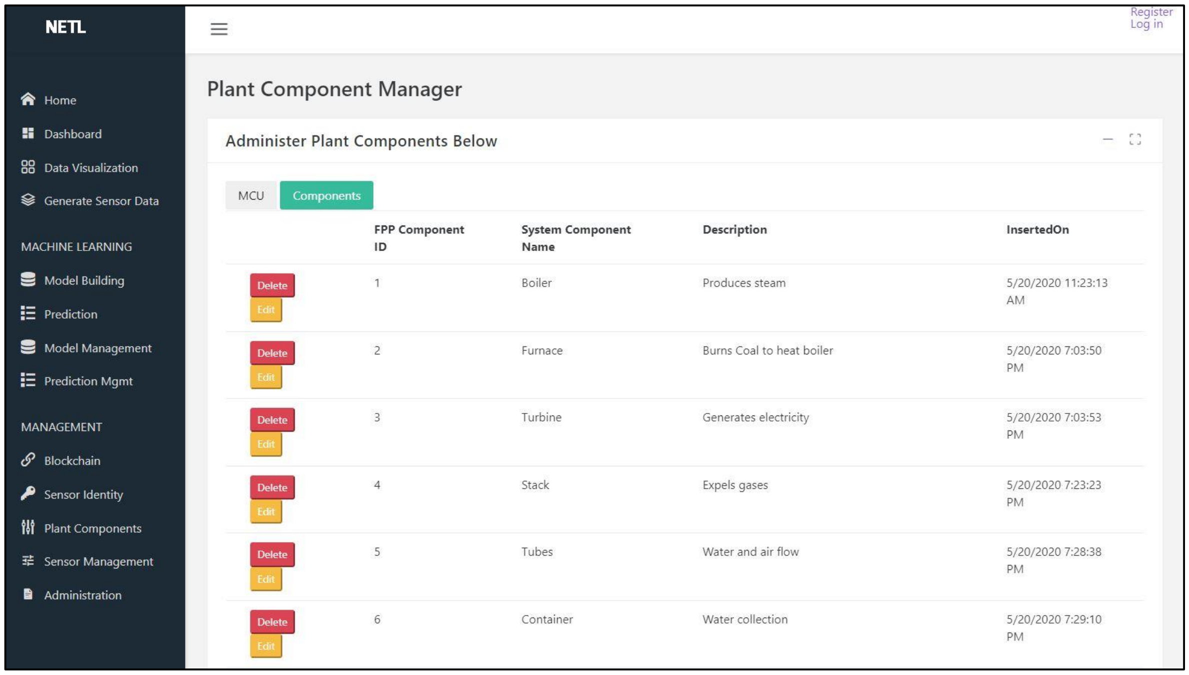
Developed by

Florida International University (FIU) - Applied Research Center (ARC)

© 2022 - Secure Data Logging and Processing with Blockchain and Machine Learning FIU-ARC

Figure 30: Home Page

Figure 31 shows the fossil power plant component manager where a user can add, edit, or delete various components of a plant. It also contains some data such as FPP Component ID, component name, its description and timestamp when the component was inserted into the application.



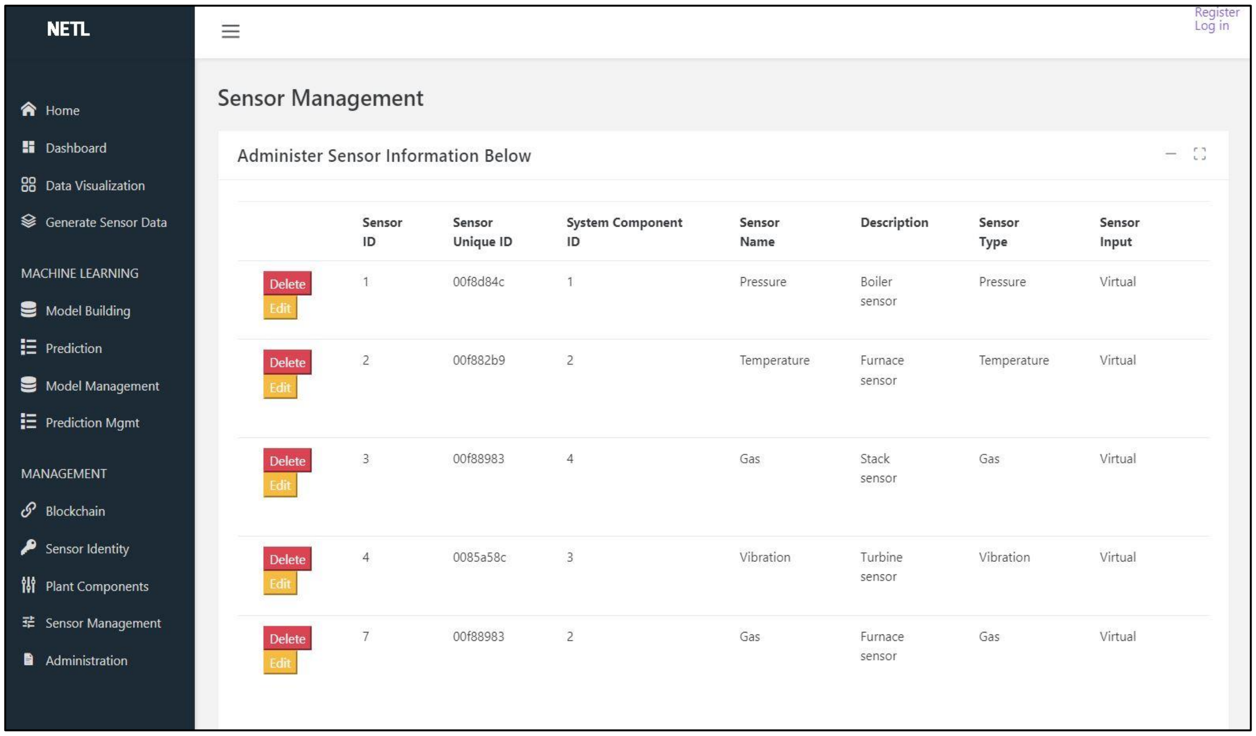
Plant Component Manager

Administer Plant Components Below

	FPP Component ID	System Component Name	Description	InsertedOn
Delete Edit	1	Boiler	Produces steam	5/20/2020 11:23:13 AM
Delete Edit	2	Furnace	Burns Coal to heat boiler	5/20/2020 7:03:50 PM
Delete Edit	3	Turbine	Generates electricity	5/20/2020 7:03:53 PM
Delete Edit	4	Stack	Expels gases	5/20/2020 7:23:23 PM
Delete Edit	5	Tubes	Water and air flow	5/20/2020 7:28:38 PM
Delete Edit	6	Container	Water collection	5/20/2020 7:29:10 PM

Figure 31: Fossil Power Plant Components Management

Figure 32 to 40 outlines the different modules of the system such as Sensor Management, Sensor Dashboard, Generating Synthetic Sensor Data, Sensor Data Visualization, Sensor Identification and Authentication, Machine Learning Model Building, Machine Learning Prediction, Machine Learning Model Management, Machine Learning Prediction Management.

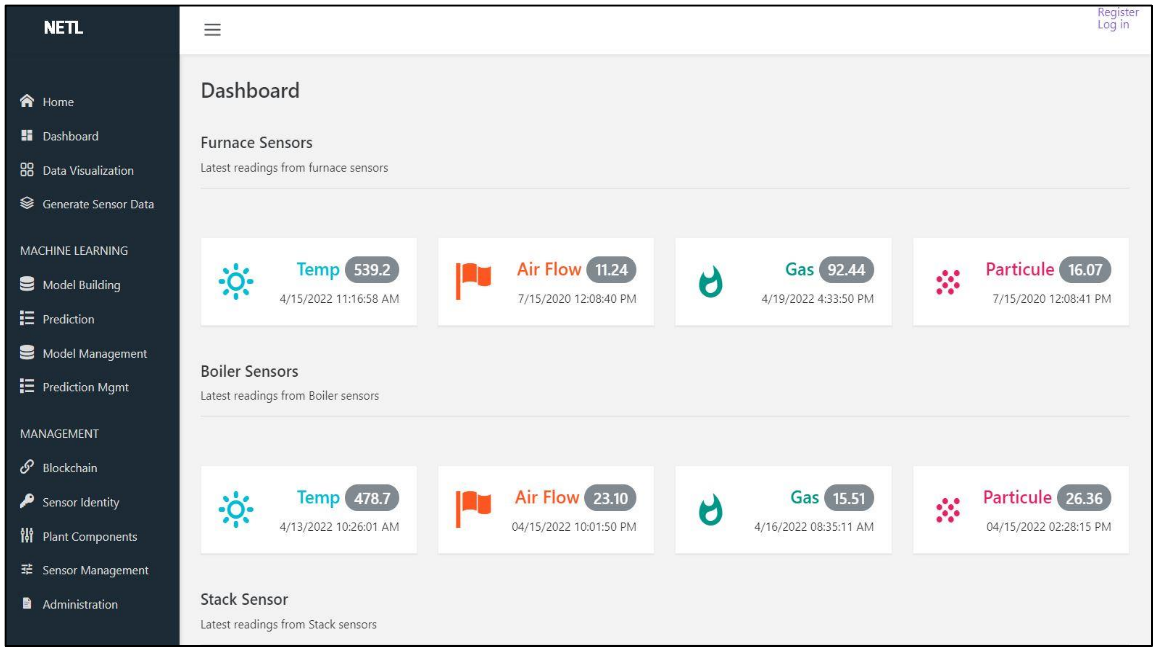


Sensor Management

Administer Sensor Information Below

	Sensor ID	Sensor Unique ID	System Component ID	Sensor Name	Description	Sensor Type	Sensor Input
Delete Edit	1	00f8d84c	1	Pressure	Boiler sensor	Pressure	Virtual
Delete Edit	2	00f882b9	2	Temperature	Furnace sensor	Temperature	Virtual
Delete Edit	3	00f88983	4	Gas	Stack sensor	Gas	Virtual
Delete Edit	4	0085a58c	3	Vibration	Turbine sensor	Vibration	Virtual
Delete Edit	7	00f88983	2	Gas	Furnace sensor	Gas	Virtual

Figure 32: Sensor Management



Dashboard

Furnace Sensors

Latest readings from furnace sensors

Temp 539.2	Air Flow 11.24	Gas 92.44	Particule 16.07
4/15/2022 11:16:58 AM	7/15/2020 12:08:40 PM	4/19/2022 4:33:50 PM	7/15/2020 12:08:41 PM

Boiler Sensors

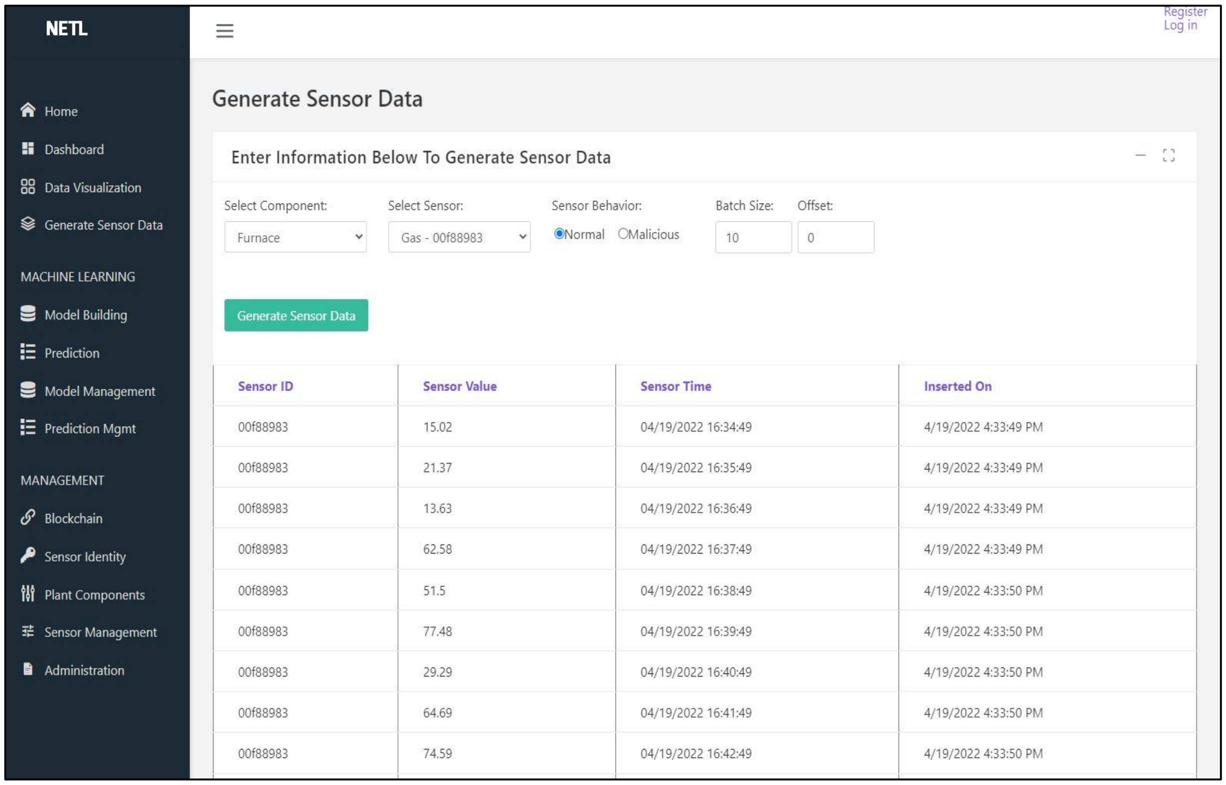
Latest readings from Boiler sensors

Temp 478.7	Air Flow 23.10	Gas 15.51	Particule 26.36
4/13/2022 10:26:01 AM	04/15/2022 10:01:50 PM	4/16/2022 08:35:11 AM	04/15/2022 02:28:15 PM

Stack Sensor

Latest readings from Stack sensors

Figure 33: Sensor Dashboard



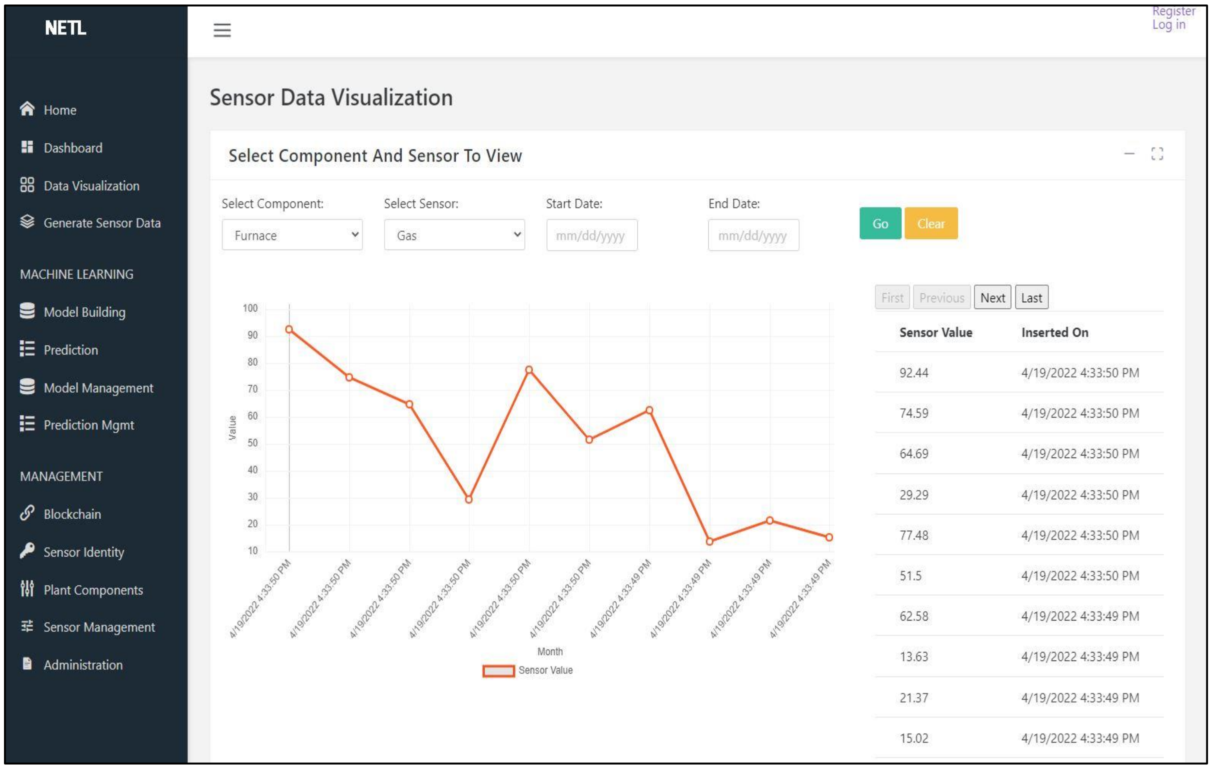
Generate Sensor Data

Enter Information Below To Generate Sensor Data

Select Component: Furnace
Select Sensor: Gas - 00f88983
Sensor Behavior: [✓ Normal] Malicious
Batch Size: 10
Offset: 0

Sensor ID	Sensor Value	Sensor Time	Inserted On
00f88983	15.02	04/19/2022 16:34:49	4/19/2022 4:33:49 PM
00f88983	21.37	04/19/2022 16:35:49	4/19/2022 4:33:49 PM
00f88983	13.63	04/19/2022 16:36:49	4/19/2022 4:33:49 PM
00f88983	62.58	04/19/2022 16:37:49	4/19/2022 4:33:49 PM
00f88983	51.5	04/19/2022 16:38:49	4/19/2022 4:33:50 PM
00f88983	77.48	04/19/2022 16:39:49	4/19/2022 4:33:50 PM
00f88983	29.29	04/19/2022 16:40:49	4/19/2022 4:33:50 PM
00f88983	64.69	04/19/2022 16:41:49	4/19/2022 4:33:50 PM
00f88983	74.59	04/19/2022 16:42:49	4/19/2022 4:33:50 PM

Figure 34: Generate Sensor Data



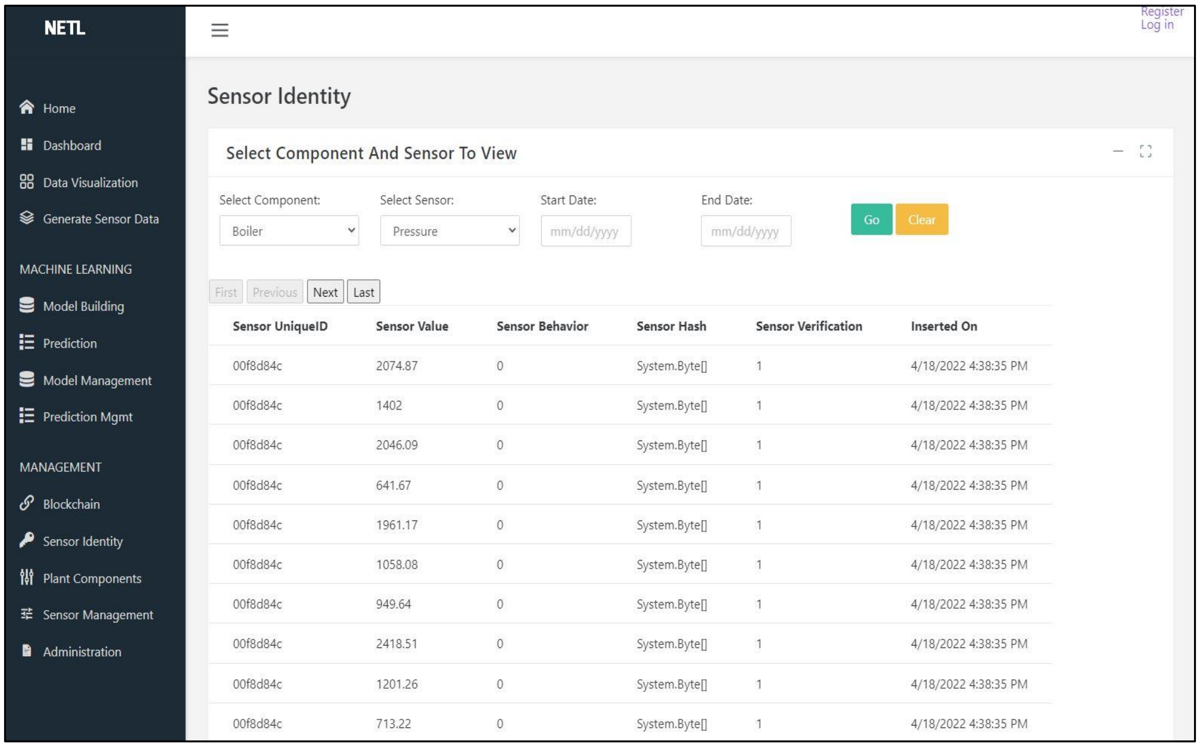
Sensor Data Visualization

Select Component And Sensor To View

Select Component: Furnace
Select Sensor: Gas
Start Date: mm/dd/yyyy
End Date: mm/dd/yyyy

Sensor Value	Inserted On
92.44	4/19/2022 4:33:50 PM
74.59	4/19/2022 4:33:50 PM
64.69	4/19/2022 4:33:50 PM
29.29	4/19/2022 4:33:50 PM
77.48	4/19/2022 4:33:50 PM
51.5	4/19/2022 4:33:50 PM
62.58	4/19/2022 4:33:49 PM
13.63	4/19/2022 4:33:49 PM
21.37	4/19/2022 4:33:49 PM
15.02	4/19/2022 4:33:49 PM

Figure 35: Sensor Data Visualization



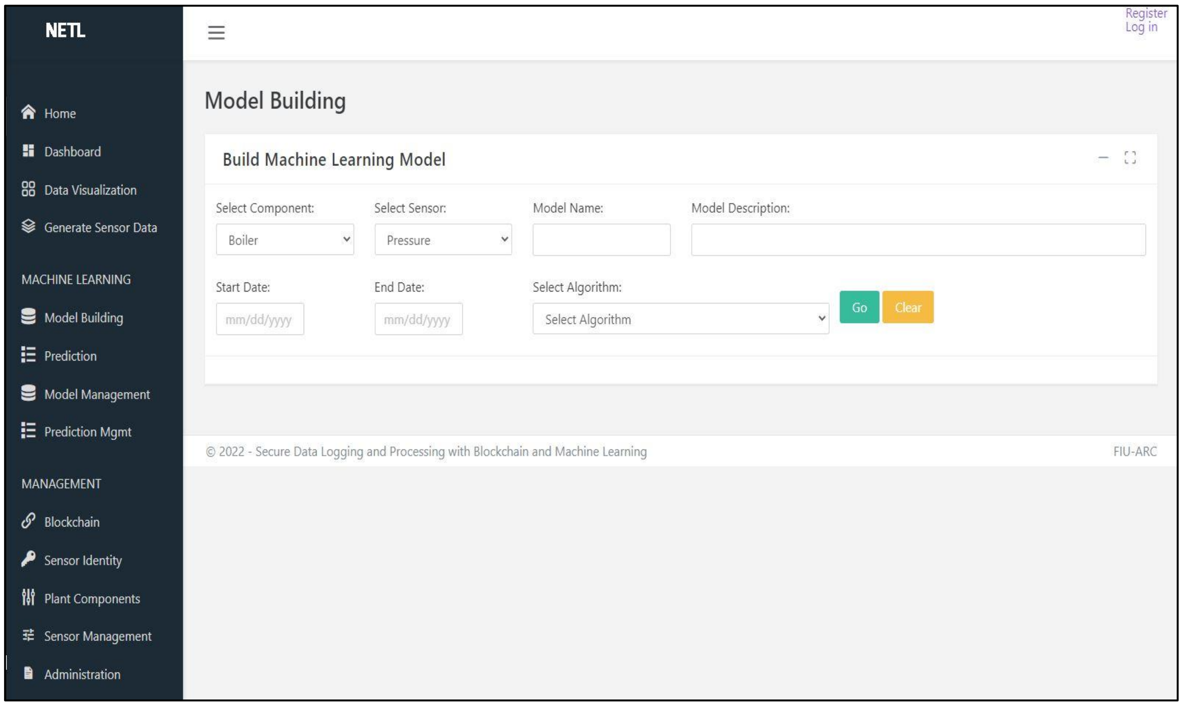
Sensor Identity

Select Component And Sensor To View

Select Component: Boiler
Select Sensor: Pressure
Start Date: mm/dd/yyyy
End Date: mm/dd/yyyy

Sensor UniqueID	Sensor Value	Sensor Behavior	Sensor Hash	Sensor Verification	Inserted On
00f8d84c	2074.87	0	System.Byte[]	1	4/18/2022 4:38:35 PM
00f8d84c	1402	0	System.Byte[]	1	4/18/2022 4:38:35 PM
00f8d84c	2046.09	0	System.Byte[]	1	4/18/2022 4:38:35 PM
00f8d84c	641.67	0	System.Byte[]	1	4/18/2022 4:38:35 PM
00f8d84c	1961.17	0	System.Byte[]	1	4/18/2022 4:38:35 PM
00f8d84c	1058.08	0	System.Byte[]	1	4/18/2022 4:38:35 PM
00f8d84c	949.64	0	System.Byte[]	1	4/18/2022 4:38:35 PM
00f8d84c	2418.51	0	System.Byte[]	1	4/18/2022 4:38:35 PM
00f8d84c	1201.26	0	System.Byte[]	1	4/18/2022 4:38:35 PM
00f8d84c	713.22	0	System.Byte[]	1	4/18/2022 4:38:35 PM

Figure 36: Sensor Identification and Authentication



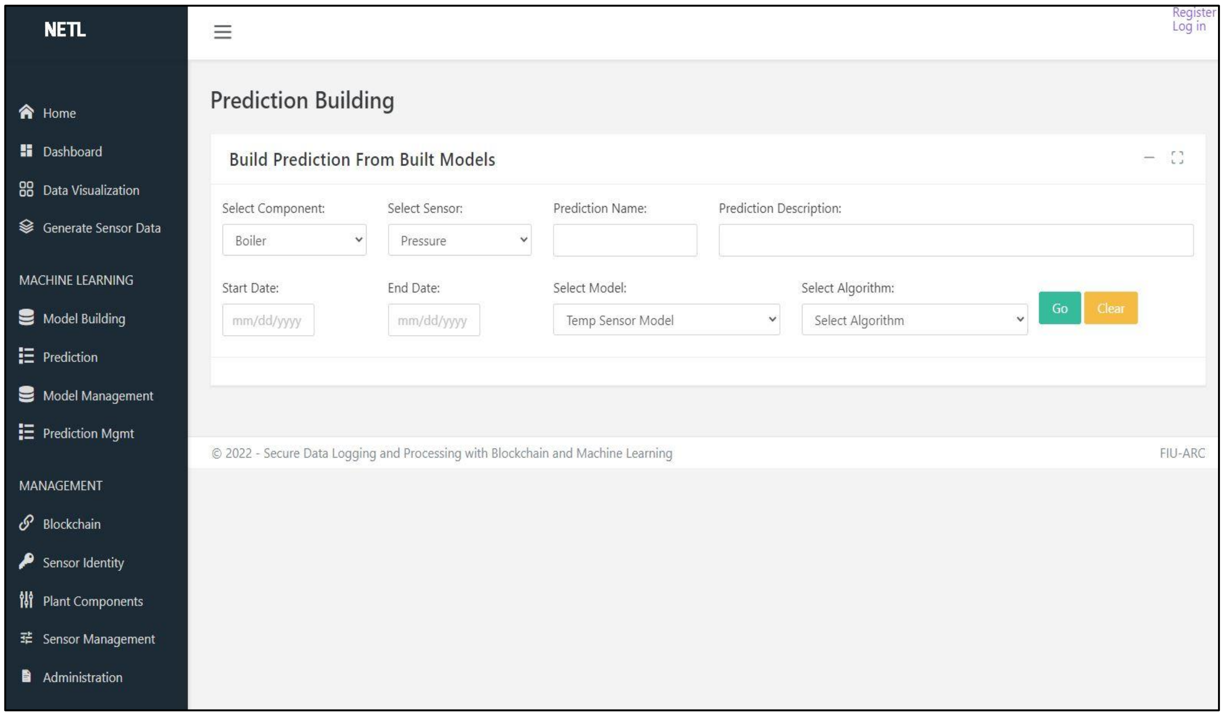
Model Building

Build Machine Learning Model

Select Component: Boiler
Select Sensor: Pressure
Model Name:
Model Description:
Start Date: mm/dd/yyyy
End Date: mm/dd/yyyy
Select Algorithm: Select Algorithm

© 2022 - Secure Data Logging and Processing with Blockchain and Machine Learning FIU-ARC

Figure 37: Machine Learning Model Building



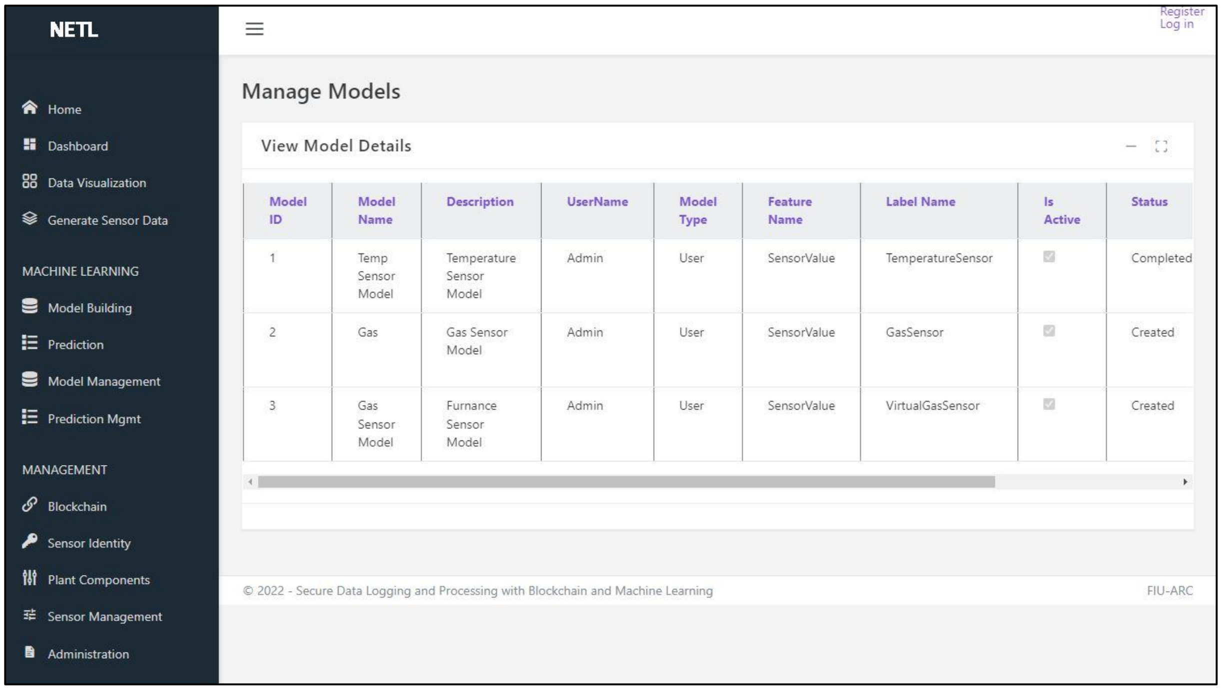
Prediction Building

Build Prediction From Built Models

Select Component: Boiler	Select Sensor: Pressure	Prediction Name:	Prediction Description:
Start Date: mm/dd/yyyy	End Date: mm/dd/yyyy	Select Model: Temp Sensor Model	Select Algorithm: Select Algorithm

© 2022 - Secure Data Logging and Processing with Blockchain and Machine Learning FIU-ARC

Figure 38: Machine Learning Prediction



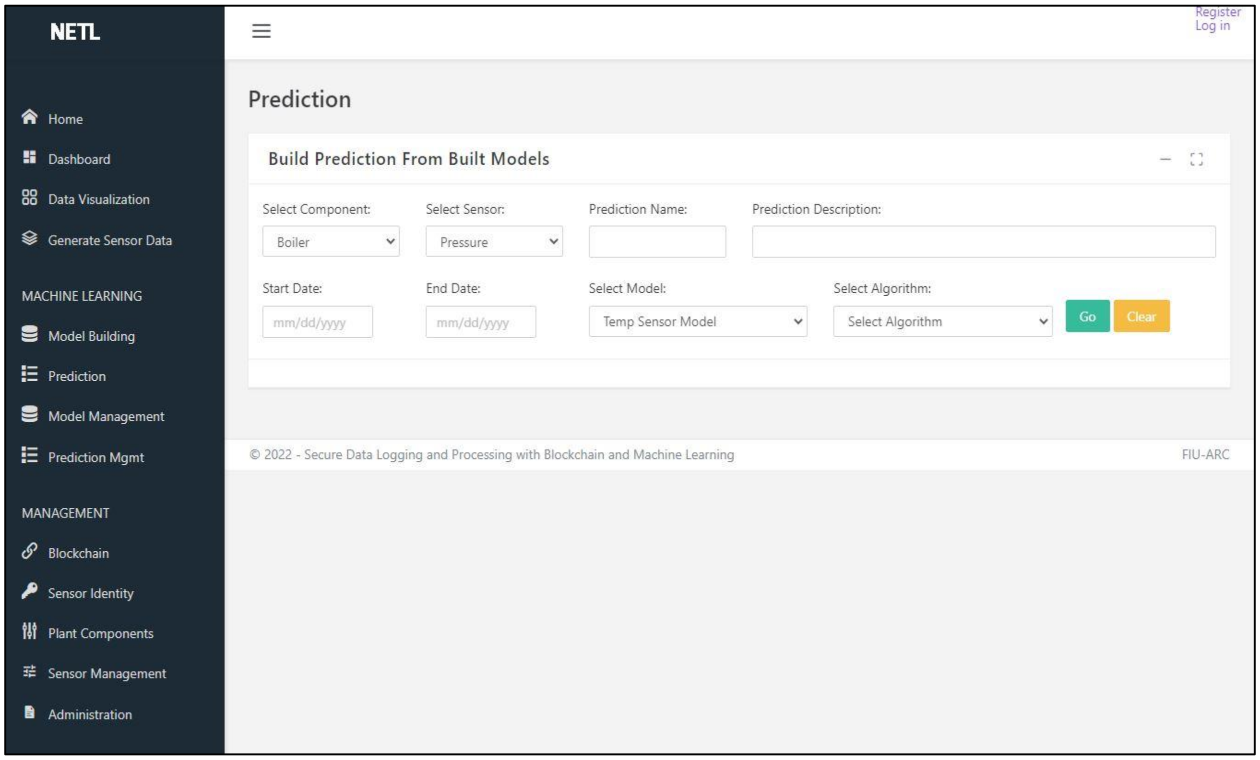
Manage Models

View Model Details

Model ID	Model Name	Description	UserName	Model Type	Feature Name	Label Name	Is Active	Status
1	Temp Sensor Model	Temperature Sensor Model	Admin	User	SensorValue	TemperatureSensor	✓	Completed
2	Gas	Gas Sensor Model	Admin	User	SensorValue	GasSensor	✓	Created
3	Gas Sensor Model	Furnance Sensor Model	Admin	User	SensorValue	VirtualGasSensor	✓	Created

© 2022 - Secure Data Logging and Processing with Blockchain and Machine Learning FIU-ARC

Figure 39: Machine Learning Model Management



Prediction

Build Prediction From Built Models

Select Component: Boiler
Select Sensor: Pressure
Prediction Name:
Prediction Description:
Start Date: mm/dd/yyyy
End Date: mm/dd/yyyy
Select Model: Temp Sensor Model
Select Algorithm: Select Algorithm

© 2022 - Secure Data Logging and Processing with Blockchain and Machine Learning FIU-ARC

Figure 40: Machine Learning Prediction Management

4. RESEARCH PUBLICATIONS:

This section outlines research publications published in peer reviewed journals and conference proceedings. All the research publications are based on the research conducted during this project.

4.1 Peer Reviewed Journal Publications:

[1] Aldyaflah, I.M.; Zhao, W.; Upadhyay, H.; Lagos, L. The Design and Implementation of a Secure Datastore Based on Ethereum Smart Contract. *Appl. Sci.* **2023**, *13*, 5282. https://doi.org/10.3390/app13095282

[2] W. Zhao, I. M. Aldyaflah, P. Gangwani, S. Joshi, H. Upadhyay and L. Lagos, "A Blockchain-Facilitated Secure Sensing Data Processing and Logging System," in IEEE Access, vol. 11, pp. 21712-21728, 2023, doi: 10.1109/ACCESS.2023.3252030.

[3] P Gangwani, S Joshi, H Upadhyay, L Lagos - International Journal of Computer Applications (0975 – 8887), “IoT Device Identity Management and Blockchain for Security and Data Integrity”, Volume 184 – No. 42, January 2023

[4] Gangwani, Pranav, Alexander Perez-Pons, Tushar Bhardwaj, Himanshu Upadhyay, Santosh Joshi, and Leonel Lagos. 2021. "Securing Environmental IoT Data Using Masked Authentication Messaging Protocol in a DAG-Based Blockchain: IOTA Tangle" Future Internet 13, no. 12: 312. https://doi.org/10.3390/fi13120312

4.2 Conference Publications:

[1] W. Zhao, "On Blockchain: Design Principle, Building Blocks, Core Innovations, and Misconceptions," in IEEE Systems, Man, and Cybernetics Magazine, vol. 8, no. 4, pp. 6-14, Oct. 2022, doi: 10.1109/MSMC.2022.3192658.

[2] W. Zhao, H. Upadhyay and L. Lagos, "Design and Implementation of a Blockchain-Enabled Secure Sensing Data Processing and Logging System," 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Melbourne, Australia, 2021, pp. 386-391, doi: 10.1109/SMC52423.2021.9658949.

[3] W. Zhao, S. Yang and X. Luo, "On Threat Analysis of IoT-Based Systems: A Survey," 2020 IEEE International Conference on Smart Internet of Things (SmartIoT), Beijing, China, 2020, pp. 205-212, doi: 10.1109/SmartIoT49966.2020.00038.

[4] Wenbing Zhao, Shunkun Yang, and Xiong Luo. 2020. Secure Hierarchical Processing and Logging of Sensing Data and IoT Events with Blockchain. In Proceedings of the 2020 The 2nd International Conference on Blockchain Technology (ICBCT'20). Association for Computing Machinery, New York, NY, USA, 52–56. https://doi.org/10.1145/3390566.3391672

[5] Elham Akbari, Wenbing Zhao, Shunkun Yang, and Xiong Luo. 2020. The Impact of Block Parameters on the Throughput and Security of Blockchains. In Proceedings of the 2020 The 2nd International Conference on Blockchain Technology (ICBCT'20). Association for Computing Machinery, New York, NY, USA, 13–18. https://doi.org/10.1145/3390566.3391673

5. CONCLUSION:

In conclusion, this project report presented a comprehensive approach to securing sensor data from fossil fuel power plants through the implementation of sensor identity management, data aggregation, and anomaly detection using machine learning, and secure logging and storage of sensor data using blockchain technology. Sensor identity management played a vital role in ensuring the authenticity and integrity of the sensor data by employing robust authentication and access control mechanisms. By implementing secure protocols and encryption techniques, unauthorized access and tampering with the sensor data were effectively mitigated.

Data aggregation and anomaly detection using machine learning algorithms provided a proactive approach to identifying abnormal patterns and potential threats within the sensor data. Through advanced analytics and pattern recognition, the system was able to detect anomalies, such as equipment malfunctions or security breaches, in real-time, enabling prompt action to be taken to mitigate risks and ensure the smooth operation of the power plants. Furthermore, the use of blockchain technology for secure logging and storage of sensor data added an additional layer of security and transparency. By leveraging the decentralized and immutable nature of blockchain, the integrity and auditability of the sensor data were guaranteed. The tamper-resistant nature of blockchain ensured that any unauthorized modification or tampering with the data could be easily detected, providing a trustworthy and reliable source of information.

Overall, the implementation of tasks in securing sensor data from fossil fuel power plants showcased a holistic and robust approach to safeguarding critical infrastructure. By combining strong identity management, advanced anomaly detection, and secure storage using blockchain, the system achieved enhanced data integrity, confidentiality, and availability. This not only helps protect the power plants from potential cyber threats but also ensures the reliability and efficiency of the energy generation process. The successful implementation of these measures underscores the importance of leveraging advanced technologies to address the ever-growing challenges of securing critical infrastructure in the digital age.

6. REFERENCES:

[1] V. Roman and J. Ordieres- Mere,IoT Blockchain Technologies for Smart Sensors Based on Raspberry Pi, *Proc. IEEE 11th Int. Conf. Serv. Comput. Appl. SOCA 2018*, pp.216–220,2019, doi: 10.1109/SOCA.2018.00038.

[2] L. Wang, et al., Cloud Computing: A Perspective Study, *New Gener. Comput.*, 28, 2, pp.137–146,2010, doi: 10.1007/s00354-008-0081-5.

[3] D. Gangwani and P. Gangwani, Applications of Machine Learning and Artificial Intelligence in Intelligent Transportation System: A Review, in *Lecture Notes in Electrical Engineering*, Springer, 2021, pp.203– 216.

[4] P. Gangwani, J. Soni, H. Upadhyay, and S. Joshi, A Deep Learning Approach for Modeling of Geothermal Energy Prediction, *Int. J. Comput. Sci. Inf. Secur.*, 18, 1, pp.62– 65,2020.

[5] B. P. Rimal, D. Pham Van, and M. Maier, Mobile- Edge Computing vs. Centralized Cloud computing in fiber- wireless access networks, in *2016 IEEE Conference on Computer Communications Workshops (INFOCOMWKSHPS)*, pp.991– 996, Apr.2016, doi: 10.1109/INFCOMW.2016.7562226.

[6] V. Parmar, H. A. Sanghvi, R. H. Patel and A. S. Pandya, A Comprehensive Study on Passwordless Authentication, *2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*,2022, pp.1266– 1275, doi:10.1109/ICSCDS53736.2022.9760934.

[7] Statista Research Department. Internet of Things (IoT)Connected Devices Installed Base Worldwide from 2015 to 2025, Aug. 03, 2016.

[8] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, DDoS in the IoT: Mirai and Other Botnets, *Computer*, 50, 7, pp.80– 84,2017, doi: 10.1109/MC.2017.201.

[9] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, Internet of Things (IoT):A vision, architectural elements, and future directions, *Futur. Gener. Comput. Syst.*, 29, 7,pp.1645–1660,2013, doi: 10.1016/j.future.2013.01.010.

[10] S. Sicari, A. Rizzardi, C. Cappiello, D. Miorandi, and A. Coen- Porisini, Toward Data Governance in the Internet of Things, in *New Advances in the Internet of Things*, R. R. Yagerand J. Pascual Espada, Eds. Cham: Springer International Publishing, 2018, pp.59– 74.

[11] F. Hawlitschek, B. Notheisen, and T. Teubner, The limits of trust- free systems: A literature review on blockchain technology and trust in the sharing economy, *Electron.Commer. Res. Appl.*, 29, pp.50– 63,May 2018, doi: 10.1016/j.elerap.2018.03.005.

[12] S. A. Al- Qaseemi,H. A. Almulhim, M. F. Almulhim, and S. R. Chaudhry, IoT architecture challenges and issues: Lack of standardization, *FTC 2016 – Proc.Futur. Technol.Conf.*, December, pp.731– 738,2017, doi: 10.1109/FTC.2016.7821686.

[13] B. Kehoe, Integrating the supply chain, *Mater. Manag. Heal. Care.*, 15, 8, pp.26– 29,2006, doi: 10.1108/eum0000000000329.

[14] D. Gangwani, Q. Liang, S. Wang, and X. Zhu, An Empirical Study of Deep Learning Frameworks for Melanoma Cancer Detection using Transfer Learning and Data Augmentation, in *2021 IEEE International Conference on Big Knowledge (ICBK)*, December 2021, pp.38– 45, doi: 10.1109/ICKG52313.2021.00015.

[15] A. Mukherjee, Physical- Layer Security on the Internet of Things: Sensing and Communication Confidentiality under Resource Constraints, *Proc. IEEE*, 103, 10, pp.1747–1761,2015, doi: 10.1109/JPROC.2015.2466548.

[16] M. Conti, R. Di Pietro, L. V. Mancini, and A. Mei, Emergent properties: Detection of the node- capture attack in mobile wireless sensor networks, *WiSec'08 Proc. 1st ACMConf. Wirel. Netw. Secur.*, pp.214– 219,2008, doi: 10.1145/1352533.1352568.

[17] L. Xie, Y. Mo, and B. Sinopoli, False Data Injection Attacks in Electricity Markets, in *2010 First IEEE International Conference on Smart Grid Communications*, 2010,pp.226– 231,doi: 10.1109/SMARTGRID.2010.5622048.

[18] D. Swathigavaishnave and R. Sarala, Detection of Malicious Code- Injection Attack Using Two Phase Analysis Technique, *Int. J. Comput. Appl.*, 45, 18, pp.8– 14, May 2012.

[19] M. Pirretti, S. Zhu, N. Vijaykrishnan, P. McDaniel, M. Kandemir, and R. Brooks, The sleep deprivation attack in sensor networks: Analysis and methods of defense, *Int. J.Distrib. Sens. Networks*, 2, 3, pp.267– 287,2006, doi: 10.1080/15501320600642718.

[20] F.- X.Standaert, Introduction to Side- Channel Attacks, in *Secure Integrated Circuits and Systems*, I. M. R. Verbauwhede, Ed. Boston, MA: Springer US, 2010, pp.27– 42.

[21] B. Collier, R. Clayton, D. R. Thomas, and A. Hutchings, Booting the booters: Evaluating the effects of police interventions in the market for denial- of-service attacks, *Proc. ACMSIGCOMM Internet Meas. Conf. IMC*, pp.50– 64,2019, doi: 10.1145/3355369.3355592.

[22] Y. Zou and G. Wang, Intercept Behavior Analysis of Industrial Wireless Sensor Networks in the Presence of Eavesdropping Attack, *IEEE Trans. Ind. Informatics*, 12,2, pp.780– 787,2016, doi: 10.1109/TII.2015.2399691.

[23] H. Liu, A new form of dos attack in a cloud and its avoidance mechanism, *Proc. ACMConf. Comput. Commun. Secur.*, pp.65– 75,2010, doi: 10.1145/1866835.1866849.

[24] S. Gupta, A. Singhal, and A. Kapoor, A literature survey on social engineering attacks: Phishing attack, *Proc. IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2016*,pp.537– 540,2017, doi: 10.1109/CCAA.2016.7813778.

[25] S. Andy, B. Rahardjo, and B. Hanindhito, Attack scenarios and security analysis of MQTT communication protocol in IoT system, *Int. Conf. Electr. Eng. Comput. Sci.Informatics*, 2017, September, pp.19– 21,2017, doi: 10.1109/EECSI.2017.8239179.

[26] M. S. Islam, M. Kuzu, and M. Kantarcioglu, Access pattern disclosure on searchable encryption: Ramification, attack and mitigation, *Ndss*, 20, pp.1– 15,2012.

[27] B. Kannhavong, H. Nakayama, Y. Nemoto, N. Kato, and A. Jamalipour, A survey of routing attacks in mobile Ad Hoc networks, *Wirel. Commun. IEEE*, 14, pp.85– 91, 2007, doi: 10.1109/MWC.2007.4396947.

[28] M. N. Ismail, A. Aborujilah, S. Musa, and A. Shahzad, Detecting flooding-based Dos attack in cloud computing environment using covariance matrix approach, *Proc.7th Int. Conf. Ubiquitous Inf. Manag. Commun. ICUIMC 2013*, pp.4– 9,2013, doi:10.1145/2448556.2448592.

[29] N. Sayegh, A. Chehab, I. H. Elhajj, and A. Kayssi, Internal security attacks on SCADA systems, *2013 3rd Int. Conf. Commun. Inf. Technol. ICCIT 2013*, pp.22– 27,2013, doi:10.1109/ICCITechnology.2013.6579516.

[30] A. Barua, H. Shahriar, and M. Zulkernine, Server-side detection of content sniffing attacks, *Proc. Int. Symp. Softw. Reliab. Eng. ISSRE*, pp.20– 29,2011, doi:10.1109/ISSRE.2011.27.

[31] M. Nawir, A. Amir, N. Yaakob, and O. B. Lynn, Internet of Things (IoT): Taxonomy of security attacks, *2016 3rd Int. Conf. Electron. Des. ICED 2016*, pp.321– 326,2017, doi:10.1109/ICED.2016.7804660.

[32] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, Blockchain, *Bus. Inf. Syst. Eng.*, 59, 3,pp.183– 187,2017, doi: 10.1007/s12599-017-0467-3.

[33] M. Cash and M. Bassiouni, Two- tier permissioned and permission- less block chain for secure data sharing, *Proc. - 3rdIEEE Int. Conf. Smart Cloud, Smart Cloud 2018*, pp.138– 144,2018, doi: 10.1109/SmartCloud.2018.00031.

[34] H. Sukhwani, J. M. Martinez, X. Chang, K. S. Trivedi, and A. Rindos, Performance modeling of PBFT consensus process for permissioned blockchain network(Hyperledger fabric), *Proc. IEEE Symp. Reliab. Distrib. Syst.*, 2017, pp.253– 255,2017, doi: 10.1109/SRDS.2017.36.

[35] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke, Blockchain technologies for the internet of things: Research issues and challenges, *IEEE Internet Things J.*, 6, 2, pp.2188– 2204,2019, doi: 10.1109/JIOT.2018.2882794.

[36] P. Gangwani, A. Perez- Pons, T. Bhardwaj, H. Upadhyay, S. Joshi, and L. Lagos, Securing Environmental IoT Data Using Masked Authentication Messaging Protocol in a DAG- Based-Blockchain: IOTA Tangle, *Futur. Internet*, 13, 12, p.312, December2021, doi: 10.3390/fi13120312.

[37] P. Cui, U. Guin, A. Skjellum, and D. Umphress, Blockchain in IoT: Current Trends, Challenges, and Future Roadmap, *J. Hardw. Syst. Secur.*, 3, 4, pp.338– 364,2019, doi:10.1007/s41635-019-00079-5.

[38] S. Roy, M. Ashaduzzaman, M. Hassan, and A. R. Chowdhury, Blockchain for IoT security and management: Current prospects, challenges and future directions, *Proc. 2018 5thInt. Conf. Networking, Syst. Secur. NSysS 2018*, 2019, doi: 10.1109/NSysS.2018.8631365.

[39] A. Aniso, A. Chaudhary, and S. Sahana, A Review of Defense against Distributed DoS attacks based on Artificial Intelligence Approaches. In *2021 IEEE 6th International Conference on Computing, Communication and Automation (ICCCA)*, pp.32– 38, IEEE, 2021.

[40] M. Kumar Singh, K. Deep Mishra, S. Sahana An Intelligent Real Time Traffic Control Based on Vehicle Density, in *International Journal of Engineering Technology and Management Sciences*, 5, 3, 2021, doi: 10.46647/ijetms. 2021.v05i03.004. ISSN:2581-4621.

[41] P. K. Sharma, N. Kumar, and J. H. Park, Blockchain technology toward green IoT: Opportunities and challenges, *IEEE Netw.*, 34, 4, pp.263– 269,2020, doi:10.1109/MNET.001.1900526.3.

[42] A. Abiri-Jahromi, M. Parvania, F. Bouffard and M. Fotuhi-Firuzabad, "A two-stage framework for power transformer asset maintenance management—Part I: Models and formulations", *IEEE Trans. Power Syst.*, vol. 28, no. 2, pp. 1395-1403, May 2013.

[43] V. Chandola, A. Banerjee and V. Kumar, "Anomaly detection: A survey", *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1-58, Jul. 2009.

[44] S. D. J. McArthur, C. D. Booth, J. R. McDonald and I. T. McFadyen, "An agent-based anomaly detection architecture for condition monitoring", *IEEE Trans. Power Syst.*, vol. 20, no. 4, pp. 1675-1682, Nov. 2005.

[45] D. P. Filev, R. B. Chinnam, F. Tseng and P. Baruah, "An industrial strength novelty detection framework for autonomous equipment monitoring and diagnostics", *IEEE Trans. Ind. Informat.*, vol. 6, no. 4, pp. 767-779, Nov. 2010.

[46] A. D. Kenyon, V. M. Catterson, S. D. J. McArthur and J. Twiddle, "An agent-based implementation of hidden Markov models for gas turbine condition monitoring", *IEEE Trans. Syst. Man Cybern.: Syst.*, vol. 44, no. 2, pp. 186-195, Feb. 2014.

[47] Z. Hameed, "Condition monitoring and fault detection of wind turbines and related algorithms: A review", *Renewable Sustain. Energy Reviews*, vol. 13, no. 1, pp. 1-39, 2009.

Acknowledgment: "This material is based upon work supported by the Department of Energy Award Number DE- FE0031745."