National College of Ireland

Framework to secure IoT enable application

MSc Research Project
MSCCLOUD Research Project

Dinesh Kanojiya
Student ID: X21104174

School of Computing
National College of Ireland

Supervisor: Jitendra Sharma

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Dinesh Kanojiya
Student ID:	X21104174
Programme:	MSCCLOUD Research Project
Year:	2022
Module:	MSc Research Project
Supervisor:	Jitendra Sharma
Submission Due Date:	15/08/2022
Project Title:	Framework to secure IoT enable application
Word Count:	5836
Page Count:	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Framework to secure IoT enable application

Dinesh Kanojiya
X21104174

Abstract

Internet of Things (IoT) devices has increased significantly over the past ten years. Data collection from these devices is now outsourced to a different cloud platforms for processing and storage. However, several programs and frameworks have been created to assure data privacy. This article offered an end-to-end IoT data privacy architecture by merging hardware- and software-based solutions using Intel Software Extension (SGX) and Advanced Encryption Standard (AES) cryptographic encryption methods. This paper shows the implementation using the .Net framework with the Intel SGX enclave used with Azure IoT. We have user C console application to mimic the IoT device and Azure IoT hub service to exchange data. In our testing, we observed that after adding an enclave layer for authentication, the request network bandwidth has increased to 7-8%, which is minimal. However, we cannot thoroughly verify end-to-end data encryption due to a lack of tools. Furthermore, we also observed that the enclave processed the information in its memory as we didn't find any trace on our hard drive.

Keywords: IoT; Intel SGX, Intel SGX SDK, AES

1 Introduction

The Internet of Things (IoT) has made available a variety of technologies that entice academics to accelerate their study. Large amounts of data are shared globally due to the rapid growth of the Internet of Things. This promotes both the public and commercial sectors to use cloud-based services' compute capabilities for data processing and storage. However, it brings security issues regarding the integrity and privacy of data across the globe since it speeds up attempts to obtain sensitive data, making it extremely difficult to safeguard the data.

IoT devices typically communicate data with outside cloud service providers; this raises the risk of data corruption, abuse of private data, and unauthorised disclosure to the public, according to Sundareswaran et al. (2012). Various data protection strategies have been established by existing literature, particularly when data is sent to the public cloud for computing and storage, such as Secure Multi-party Computation, Zhou et al. (2011). This method has strict access guidelines and several anonymisation techniques. However, it has the drawback of being unable to do any computations to safeguard data. Even if they have computing capabilities, it will still be more work to give helpful information for analysis and decision-making.

The market for cloud services has grown enormously in today's competitive environment. Because of the improved data storage and development services offered by the cloud, which are affordable, most small and medium-sized businesses shift there. There is no doubt that the cloud ecosystem provides dependable resources and affordable computing. Grobauer et al. (2011) note that there is still a need to handle a certain amount of privacy and security in relation to the IoT. However, data privacy and secrecy have been severely hampered by the migration of industrial-based data to the cloud. Therefore, a framework for processing, exchanging, and protecting sensitive data while moving data to and from the cloud is urgently needed.

2 Related Work

It has been noticed that either a recognized algorithm or hardware component exploration is employed to safeguard data. The article describes a novel IoT end-to-end data analytics framework using a hybrid approach that uses a hardware and software-based paradigm in collaboration with a cloud service provider. The sections that follow will go through how hardware (cloud systems) may help to keep data safe, as well as the communication method from a hardware standpoint. Furthermore, it explains how software (algorithms) may contribute to data confidentiality.

The hybrid solution is used since AES only encrypts data arriving from IoT devices and is then outsourced to the cloud system. However, once data reaches the cloud, it is hacked and transferred to IoT devices owing to security flaws. Furthermore, the malicious actor can readily read this information when returning to remote devices. As a result, it is critical to develop a solution that combines the strength of hardware and software-based methodologies to establish an end-to-end data integrity architecture.

The following literature concerns IoT devices, data communication, networks, and security flaws. Following that is the hardware portion, which will examine what hardware (cloud systems) has to offer in terms of data security and how secure communication occurs between IoT devices. Finally, it delves into the software component that helps to ensure data integrity.

2.1 IoT System and Automation

The Internet of Things (IoT) system is a collection of devices that intelligently connect with humans to achieve shared goals, regardless of the surroundings, according to Sicari et al. (2015). For processing and storing data from IoT devices, an ecosystem that comprises cloud, fog, and edge computing is required. When it comes to IoT ecosystem security, these devices link for improved communication across the network. The security danger is exacerbated in this ecosystem, and it is encouraged to have a totally secure system, including network and data, against hostile assaults.

The most notable aspect of IoT devices is their ability to automate processes without user participation; they are more accurate and can carry out more tasks at a lower cost. Actuators and sensors are included in IoT devices to allow them to interact with their surroundings. Sensors detect and gather ambient data such as temperature and door lock condition before sending it to the cloud for analysis. Based on user-defined actions,

an appropriate command is given back to the device's actuators. Because the devices are low-powered, the manufacturer provides services for maintaining and controlling the machine; in addition, expose tools and API services for developers to construct numerous bespoke applications and automation Iulia Bastys (2018).

IoT devices are overly powerful, resulting in unknown losses in the absence of security. The data exposed by IoT devices falls into three categories:

- Data storage: This gathers user information, device identity, and trace logs.
- Sensor data: This collects information about the environment in and around the installed or configured devices.
- Activity data: Jingjing Ren (2019) gather all user interaction information with the devices.

The data listed above might be shared primarily with two parties: the device maker and the public cloud service. The three categories of data mentioned above are exposed to the cloud, making them an excellent option for privacy limits.

The IoT network trusts service providers to share users' plentiful data, but the problem is to balance service provider trust. Third-party cloud service providers, on the other hand, assure data confidentiality while processing and storing but fail to maintain security. Furthermore, sensitive information is exposed to a hostile actor as a result of a poor encryption policy. Users are exposed since their privacy has been violated; the malicious actor can now obtain financial information or learn about user activities in the house. Furthermore, due to a weak control system, enemies physically get root access to the machine placed at home and steal all data, making life even more difficult.

The attackers' only purpose is to eavesdrop on the network and access user private data in order to utilize it at whim; consequently, in order to minimize such instances, we must employ correct cryptographic encryption techniques in conjunction with a Trust Execution Environment (TEE) platform such as Intel SGX.

2.2 Trusted Execution Environments (TEE)

Before we get started with Intel SGX, let's first define Trusted Execution Environment (TEE).

The basic goal of TEE is to create trusted computing for secure computation, privacy, and data protection. This enables the system to safeguard data integrity and the encrypted cryptographic key within the unaltered hardware module. TEE is a memory and storage environment that provides a safe and integrity-protected processing environment within the processor. Because of its permanent memory, it ensures the secrecy of its programs and data saved in the runtime environment. Furthermore, it demonstrates trustworthiness in running important programs within the environment and protects the system against third-party software assaults and physical threats, according to Sabt et al. (2015).

Because of its growing popularity, several existing organizations began developing their own TEE for their unique needs. However, in this work, we will compare two TEE providers: ARM TrustZone and Intel Software Guard Extensions (SGX), both of which are well-known for securing remote IoT devices. In the following sections, we will investigate and compare ARM TrustZone and Intel SGX.

2.3 ARM TrustZone (ATZ)

ARM TrustZone is a hardware-based security enhancement introduced in Cortex-A processors with the ARM application that offers a trusted execution environment (TEE). The most recent version has the Cortex-M CPU and a new microcontroller. TrustZone explores a System on Chip and CPU-wide security solution. It is primarily concerned with two protection domain components: secure and normal worlds. These are independent methods that are executed by the processors. Both are completely hardware segregated and have distinct access restrictions, such as non-secure cannot access secure world resources. As a result, ARM TrustZone is an excellent candidate for a novel approach for securing programs and their data. In other terms, a client application operates in the normal world (NW), but a secure application operates in the secure world (SW).NW is unable to access the resources on SW. Programs in SW, on the other hand, have complete access to system resources. As a result, important applications may be installed in SW, and it is also protected from a NW, according to Pinto and Santos (2019).

The fundamental concept is to create an architecture with a programable environment that protects all assets from practically all assaults while also offering integrity, confidentiality, and building security solutions. It provides the Global Protection client API and the TEE internal core API, which allow developers to leverage these services and create safe encryption modules. ATZ is divided into two independent states: SW and NW. System buses are used to signal all peripheral devices in these states, and a median called monitor is in charge of sharing context between the two. The memory controller is responsible for running the program in SW and preventing access from an application running in NW by utilizing ARM architecture and assured memory isolation. Furthermore, an exception might be reported in the SW application, preventing a program operating in the SW environment from accessing the device.

Researchers expanded their experiment to exploit the potential of the new security systems after considering the data security provided by ATZ trustworthy environment. With an increase in the use of IoT devices and unauthorised access, ATZ is being used to safeguard user data and ensure integrity. The parts that follow will go through a few apps that were created with ARM TrustZone.

Guan et al. (2019) The article addressed TrustShadow, a novel program created to safeguard ARM IoT-based devices from OS compromise and unauthorized physical access. TrustShadow uses TEE to establish a secure environment for important apps, preventing illegal IoT access physically. Because it operates on segregated memory, the new memory encryption and decryption prevent unauthorized users from obtaining access to protected data and protects data even if the operating system is hacked.

Li et al. (2020) The research investigates the multi-threading technique in which IoT

devices leverage cutting-edge algorithms such as fingerprint and 3D facial recognition for mobile payment applications where security is crucial and processing power is limited. It recommends the following approaches for parallel execution of TEE, together with its benefits and drawbacks:

- Increase the number of hardware cores and spread TEE tasks using a global scheduler, albeit at a cost.
- Implement a partition policy to divide the physical core according to workloads. As the workload grows, allocate more core to TEE and decommission the core as the workload reduces; nevertheless, this strategy mainly advantages batch process applications. It stores the partition policy that will be used for later tasks.

In addition, the study suggests a novel technique in which a daemon thread interacts with the TEE by spawning a shadow thread for each TEE operation. To assure execution, they created a thread-compatible interface, libteethread, which aids in communication between the daemon and the TEE.

To summarize, ARM TrustZone is a Trusted Execution Environment that runs in an isolated environment within the physical location of the processor. It provides the Global Protection API and the internal core API service to developers in order for them to design data integrity modules for IoT devices. Intel SGX, an alternative to ATZ, provides a TEE environment and an Intel SDK framework for developing a security module for storing and processing important data.

2.4 Intel Software Guard Extensions (SGX)

Intel software guard extension (SGX) was introduced with the x86-64 instruction set architecture and was written in C/C++ to allow a program to reserve a section of memory space as a TEE known as an enclave container. There can be many enclaves that are only accessible by the hardware. Enclave containers, on the other hand, may access any region of the process memory. Intel presents a software development kit (SDK) and assures data safety even when all resources, including the operating system, BIOS, drivers, and so on, are compromised. Even if a hacker obtains access to the entire system, the section contained within the enclave will remain secure. Enclave employs signed cryptographic encryption for its data and code to maintain privacy and integrity. It also prevents other enclaves and programs from accessing the enclave data. Furthermore, enclave provides two important features: sealing and remote attestation. Sealing protects data from the outside world, while remote authentication ensures secure authentication with the distant device; if successful, the device can outsource material within the enclave. Wang et al. (2018).

- Enclaves: Figure 1 depicts enclaves the conventional user-based process has data, code, and an operating system to provide process bits, as well as an enclave integrated in the user processes. The Enclave contains independent code and data blocks that allow an application portion to be run within the enclave. It also supports multi-threading through the Thread Control Structure (TCS), which allows us to alter enclaves to execute simultaneous threads within a single enclave. Furthermore, the enclave has read/write access to all OS resources and memory. Other

OS resources, on the other hand, cannot access the enclave environment due to its signature encryption capabilities, according to Sinha et al. (2015).

- Attestation: Attestation is a type of authentication in which an enclave secures a portion of an application. Once loaded, it ensures that third-party programs securely exchange confidential data. To do this, a third-party must submit credentials that are identical to the enclave secure information and signature. Local and remote attestation are available from SGX. Local attestation is utilized for the same platform, but remote attestation allows any distant device to access the enclave security platform. Sardar et al. (2021) state that the enclave determines whether or not the distant device may be trusted based on the signature.

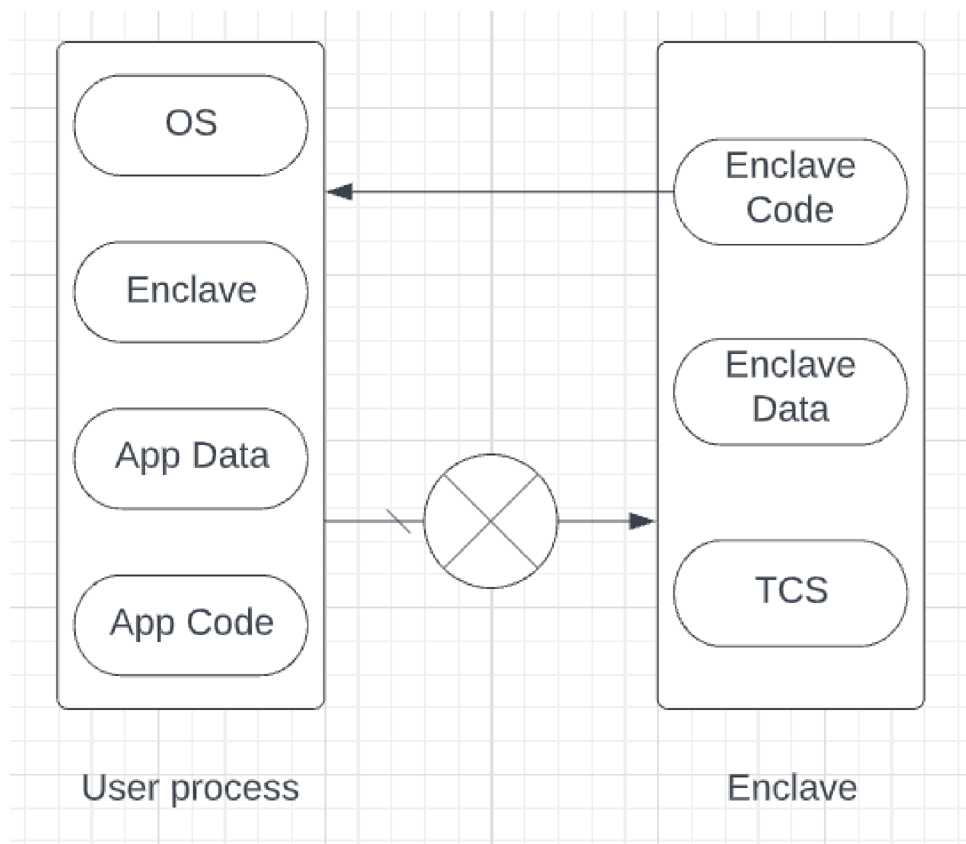

Figure 1: Embedded OS and Enclave

Furthermore, the Intel SGX features provide the use of computer memory and a private key encryption technique to safeguard code and data. Various studies on IoT-based applications for various areas have been undertaken over time. The parts that follow comprise a few literary works done with SGX that lay the way for a new encryption architecture for distant devices.

Correia et al. (2020)The paper discussed an Omega application that secures fog edge middleware using Intel SGX. Omega is made up of a module that operates on the critical key-value storage mechanism OmegaKV model, which reads and writes operations in a random sequence. OmegaKV processes the sequence of data demanded by fog nodes and

ensures the order of read and write activities. It makes use of Intel SGX since Omega creates all events with a digital signature using a secret key inside the SGX enclave. This maintains the data state and provides security while maintaining low latency and high throughput.

Elgamal and Nahrstedt (2020)The published research created a video analysis application for home security that uses fog computing and Deep Neural Network (DNN) computation inside the Intel SGX enclave to identify an intruder and safeguard the stream data from harmful users. The article also includes numerous enclaves for DNN processing to reach video stream performance. The research is the first attempt to partitioning memory's processing capability utilizing several enclaves and machine learning. This provides a parallel pipeline in which both enclaves process two separate video frames in concurrently, improving the application's overall performance. This test scenario includes a car break-in, an intruder in the garden, and package theft.

Gao et al. (2021)The study talked on blockchain-based IoT applications for the health care industry. As the number of mobile users grows tremendously, as is the case for online health and diagnostic specialists, privacy problems arise since a vast quantity of user personal information is transferred over the fog to cloud services, making it an excellent candidate for leakage. To protect the safety of user identification data, the author integrates the blockchain technology with the Intel SGX key encryption system. The blockchain authenticates users on cloud service providers by utilizing SGX and a well-defined access policy. This also prevents rogue edge nodes from accessing sensitive data from users.

Overall, Intel SGX provides a TEE environment call enclave as well as attestation functions for local and distant devices. In addition, Intel provides an SDK tool to assist developers in creating private key signed encryption modules for local and distant device authentication. As a result, much research has been performed to ensure the integrity and security of IoT device data. The part that follows analyzes the Advanced Encryption Standard (AES) cryptographic method, which is well recognized for its lightweight encryption architecture, and looks ahead to how IoT has used AES cryptography encryption. Furthermore, the next part for this literature compares ARM TrustZone with Intel SGX and highlights the merits and downsides for IoT devices.

2.5 Advanced Encryption Standard (AES)

Before we get started with the Advanced Encryption Standard (AES), let's define encryption. How can encryption aid with data security?

The primary benefit of using a cloud service is security and secrecy. Any security flaw, on the other hand, may reduce service performance. Despite the availability of numerous security techniques, security breaches nevertheless occur on occasion. Encryption algorithms are classified into two types: symmetric and asymmetric. Symmetric encryption encrypts and decrypts text using a single key and does not require a lot of computer power to run. Furthermore, the key is only used once; if another plain text is required, a new key is produced and the appropriate actions are performed. Asymmetric, on the other hand, employs two keys, one to encrypt data and the other to decode it. Gupta

et al. (2018) state that this sort of method always creates a public and private key for execution. AddRoundKey will encrypt the whole block at each level of repetition until the operation is completed.

2.6 AES Algorithm method

This section describes the algorithm flow for converting plain text to ciphertext. As indicated in section 2.6, AES enables several key sizes such as 128, 192, and 256 bits to safeguard data, with 10, 12, and 14 encryption cycles. As shown in Figure 2, these rounds contain round-key produced from the encryption key, and each band consists of four processing steps: SubBytes, ShiftRows, MixCloumns, and AddRoundKey.

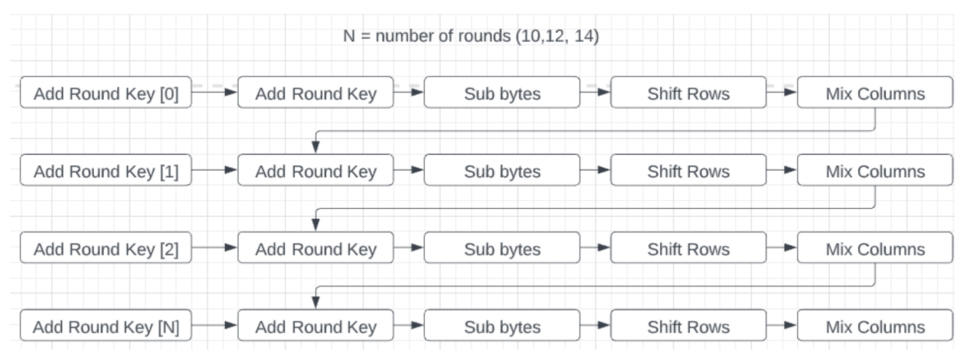

Figure 2: AES Encryption Alorigthm flow chart.

The following is a processing step:

- SubBytes: This changes a byte to a new value in 16 identical bytes for a 256-byte text length.
- ShiftRows: Shifts bytes from one block to the next by 1, 2, and 3, correspondingly.
- MixColumns multiplies each data block column by a modular polynomial. SubBytes and MixColumns can be integrated into huge Look-Up-Tables instead of being computed separately (LUT).
- Tsai et al. (2018) state that the AddRoundKey transformation adds the data block with a round-key generated from the initial secret key.

Because of its versatility and minimal computing power, most IoT devices use AES cryptography to encrypt and decode data. They can be used with another algorithm to improve the application's performance. The following are a few literary works that highlight how AES helps the IoT ecosystem retain security.

Zhang et al. (2018) The literature work mostly represents in-memory computing for IoT computing. It is the first time that AES cryptography has been employed with Safeguard Hash Algorithm 3 (SHA-3) to secure data, as the SHA algorithm gives one-way encryption to add another layer of protection to the data. The article also addressed Recryptor, a novel cryptographic processor that is programable, energy efficient, and improves the performance of IoT security applications.

Yu et al. (2018) The study focuses on IoT-based industrial applications such as transportation and manufacturing systems that require a secure framework with tight access control; this implies that only data owners may safely remove information. To accomplish this, the author encrypts the future user's (data owner's) access control data with AES encryption and performs secure resource destruction in fog computing.

Zhang et al. (2021) The study explores mobile edge computing (MEC) to offload work in an efficient and safe manner. Mobile device users (MDU) typically connect to the tiny station (Fog) to offload intense data from a restricted compute station. This increases latency and reduces application performance. It also causes worry since suitable secure communication is not properly defined. To solve this issue, a new security layer was built to reduce danger while offloading the data. To address the vulnerability of the data, these layers are built with AES cryptographic encryption and a load balancing mechanism.

To summarize, AES cryptographic encryption is a lightweight, safe, and trustworthy technique. It also allows the flexibility to be utilized with other algorithms to improve the application's performance and efficiency, as detailed in the previous study on how AES is used to handle various practical security challenges. As a result, the AES will be used in this article to ensure end-to-end data integrity. In the next part, the article compares ARM TrustZone with Intel SGX for better comprehension and makes an attempt to justify the choice of SGX with AES for this literary work.

2.7 Compare ARM TrustZone and Intel SGX

In general, the fundamental worry while processing and storing massive amounts of created data is security. TEE has been used to reduce the danger of IoT devices such as mobile, intelligent, and health monitoring devices relaying personal information to fog and cloud computing. ARM TrustZone and Intel SGX TEEs are used to safeguard remote device data. The sections that follow compare and contrast both TEEs:

The Intel SGX, as indicated in section 2.4, offers an isolated area on the physical CPU known as an enclave. This enclave ensures data and code security even if the operating system is hacked. The Intel SDK may be used by developers to setup the enclave and implement encryption and decryption routines. This ensures that no bad actor has access to the data held within the enclave. Furthermore, SGX provides attestation, which is an authentication mechanism for validating devices, both local and remote. The attestation capability is the centerpiece of the Intel SGX; as a result, every IoT device based on the attestation implementation must authenticate; failure to do so results in the enclave refusing access. This provides another degree of protection.

As noted in section 2.3, ARM TrustZone provides a secure world (SW) and an ordinary world (NW) in a separated memory region where the program within is more safe in SW and shielded from outside access, even if NW cannot access SW resources. Furthermore, SW leverages the GlobalPlatform TEE client API and the Internal Core API services to protect sensitive data like as login and payment information. The ARM TrustZone enables IoT devices to process data in the cloud while retaining secrecy. The ARM TrustZone's fundamental flaw was that it did not enable remote attestation.

Before delivering encrypted data to cloud servers, the remote device needs be authorized. As a result, the article offered an end-to-end data integrity architecture for IoT devices based on Intel SGX and AES cryptographic encryption.

3 Methodology

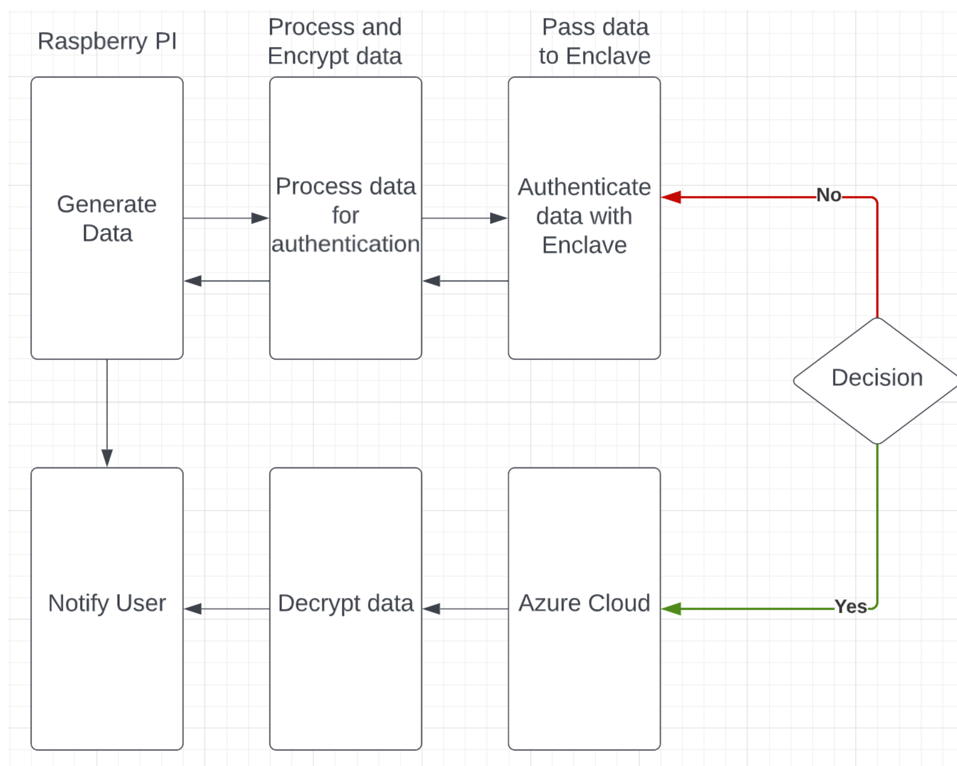

Figure 3: Methodology

The research methods consist of various stages as follows 3:

1. **Stage 1:** Raspberry PI code will generate data in a JSON format. However, in this project, we created a .net console application which will mimic a raspberry PI to generate data and pass it to another layer for processing.

2. **Stage2:** The generated data contains deviceId and user-related data such as room temperature and humidity. This information will get a pass to the cloud to monitor the room temperature. Before data pass to the cloud, it gets encrypted using the AES encryption algorithm, especially critical data such as deviceId, username and password, if any. For the AES cryptography algorithm, we have used a single key for encryption and decryption.

   First, it will check whether the device is registered or not with the given username and password. If yes, it will authenticate by the enclave; upon success, it passes to the cloud for storage and further operations. In this stage, the user can register

and remove devices based on the credential. Furthermore, the user can update the old password by providing device information.

3. **Stage 3:** All authentication takes place inside the enclave. The enclave is the trusted zone where all crucial execution occurs, such as authentication. In this project, we have used the enclave feature to perform secure operations primarily related to user personal information. Enclave execution takes place inside its own allocated memory.

   We have created a wallet inside the enclave to perform CRUD operations for users' critical data. Add method will register a device associated with a username and password. Remove will remove the device registration. The display will search and locate the device with user credentials, and at last, we have an update which will update the password with the registered device.

4. **Stage 4:** Once the enclave returns successful authentication, device data will pass to could for storage and further process such as sending a command back to the device. If the enclave cannot authenticate the device, the action will redirect to the Raspberry Pi device, followed by a notification to the user.

5. **Stage 5:** Once data is received from the cloud, it will decrypt it using the AES cryptography algorithm and notify the user.

4 Design Specification

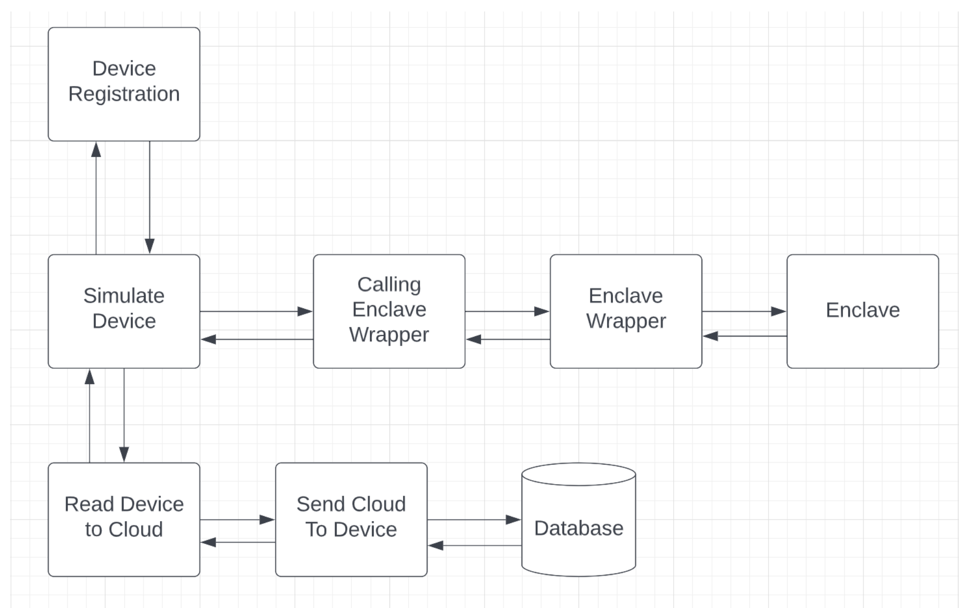

Figure 4: Architecture

The system architecture consists of seven projects that work collectively in a single solution to provide a hybrid-based encryption solution for end-to-end IoT device encryption. Foremost, we need to have the Azure account, and inside the Azure portal, we need to

create the Azure IoT service. Once the service gets created, it generates the azure connection string, which will be used to connect the C application with the azure.

Once the Azure portal is in place, then on the application front, we need to install nugget packages so that it will provide the necessary libraries to talk to Azure directly from the application. Nugget packages are as follows: Microsoft.Azure.Device.Client, Microsoft.Azure.Devices.Shared and Newtonsoft.Json are used for communication between the application and Azure.

This solution has been implemented using C.Net Core 3.1 version, which uses a console application to mimic a Raspberry PI simulation to send and receive data from the Azure cloud.

The architecture 3 is as follows:

```
private static byte[] EncryptStringToBytes(string data)
{
    byte[] encryptedAuditTrail;

    using (System.Security.Cryptography.Aes newAes = Aes.Create())
    {
        newAes.Key = Key;
        newAes.IV = IV;

        ICryptoTransform encryptor = newAes.CreateEncryptor(Key, IV);

        using (MemoryStream msEncrypt = new MemoryStream())
        {
            using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
            {
                using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                {
                    swEncrypt.Write(data);
                }
                encryptedAuditTrail = msEncrypt.ToArray();
            }
        }
    }

    return encryptedAuditTrail;
}
```

```
private static string DecryptStringFromBytes(byte[] data)
{
    string decryptText;

    using (Aes newAes = Aes.Create())
    {
        newAes.Key = Key;
        newAes.IV = IV;

        ICryptoTransform decryptor = newAes.CreateDecryptor(Key, IV);

        using (MemoryStream msDecrypt = new MemoryStream(data))
        {
            using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))
            {
                using (StreamReader srDecrypt = new StreamReader(csDecrypt))
                {
                    decryptText = srDecrypt.ReadToEnd();
                }
            }
        }
    }


    return decryptText;
}
```

Figure 5: AES

1. **Simulate Device:** This is a C console-based application, which is the application's entry point and mimics a Raspberry PI simulator. The console application is chosen because the Raspberry PI simulator is unavailable for .Net-based applications. However, we can deploy this application on the device with the additional configuration, which is not in this project's scope. Furthermore, this project also notifies the user of their respected request.
   The simulated device needs registration on the Azure cloud. Once done, it will return the device id to the application. Furthermore, this device id and user credentials will be appended in JSON for authentication. Before authentication takes place, it gets encrypted using Advanced Encryption Standard (AES) cryptographic encryption.
   As mentioned in section 2, AES uses key for encryption and decryption 5 to make life easier .Net framework provides an inbuilt AES algorithm. We have inherited that function and perform encryption logic with a single key. Refer to the figure. Once the JSON data gets encrypted; then it will pass to the Calling Enclave Wrapper project.

2. **Device Registration:** It is a simple project that contains the Azure connection string to connect the cloud, register a device with a unique ID, and return a symmetric key as a device id registered in Azure IoT.

3. **Enclave:** The enclave project is where we have to define the wallet for secure authentication. However, configuring enclaves requires intel SGX SDK, which provides

C library and API services to initialise and implement enclaves. It consists of two parts trusted and untrusted zone. The enclave contains two important files, enclave.edl and enclave.cpp.

(a) **Enclave.edl:** This edl file contains trusted and untrusted zone. The trusted zone execute inside the enclave for secure operation. As the enclave comes with limited memory, we only put critical tasks inside it. As in the project, we declare wallet and add CRUD operation to the enclave.

(b) **Enclave.cpp:** This is C based file where we have implemented enclave functions such as create_wallet, show_wallet, change_master_password, add_item, and remove_item. All these functions use a C struct wallet, which contains a list to store deviceid, username and password.

Intel SGX SDK also offers sealing and unsealing, which helps to store and retrieve critical data into the enclave memory. The sealing concept ensures that all data are one-way encryption, and only the enclave can retrieve and update the data into the memory.

4. **Enclave Wrapper:** This is C++ based project which wraps the enclave project to expose a function to the outside world as a DLL. The Wrappercalling.cpp function is the main file where we created the CallingEnclave class that contains CallWallet's public method. This CallWallet function inherits the Intel SGX SDK API service to initialise the enclave and call all the methods defined in the trusted and untrusted zone for the enclave.
In this project, we are passing users' data to initialise the C struct wallet and associated items so that they can store inside the enclave. All CRUD operations performed based on the command passed as an integer value as follows:
1. If argc == 1, then create a new wallet, 2. If argc == 2, find and locate the wallet. 3. If argc == 3, then change the master password. 4. If argc == 4 then add item (user's data). 5. If argc == 5, then remove item (user's data).
If any of argc the value passed which is not mentioned in the above list, it will notify "invalid command". Once the above commands are executed successfully, the enclave exists safely. If any of the commands fails, it will return the appropriate message to notify the user.

5. **Calling Enclave Wrapper:** This project links the Enclave wrapper and C project, which help two applications interact with the data. We have used the .Net interop namespace to call the C++ object a COM object especially to support data negotiations as C variables require different compiler and C++ need different. To work these two projects in sync, we required the InteropService library, which will read the C++ DLL and expose its methods.
Once the enclave has done its part, and all data is saved/ retrieved, the action passes to Simulate Device project. Upon success, the data is given to the Azure IoT cloud for further process and storage.

6. **SendCloudToDevice:** Once the cloud process the data and this data needs to send back to the device to notify the user. This project connects to the cloud and picks messages that would need to be delivered back to the user. Once the device receives the news, it will notify on the screen.

7. **ReceivedDeviceToCloudMessage:** This project sends a delivery notification for the message delivered from cloud to device.

4.1 Video Presentation

we have created video presentation which detailed out the architecture and touch base of other components that we have used for the research. Follow the link for Video Presentation or copy and paste the URl on the browser `https://web.microsoftstream.com/video/e42f2c38-a135-4b48-8c95-aa9702842fc8?list=studio` for a Video Presentation.

5 Implementation

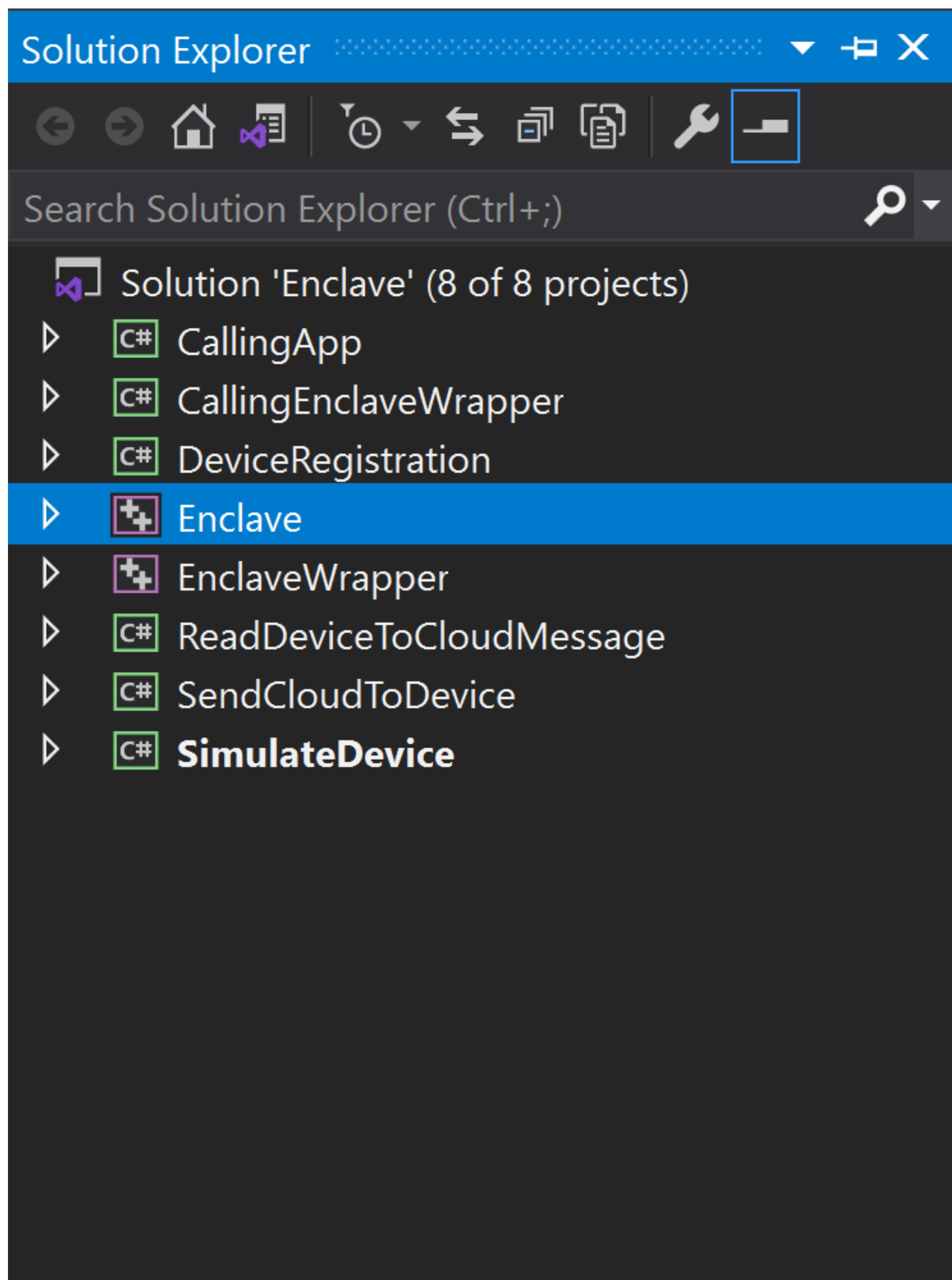

Figure 6: Project Solutions

We created this project using the .Net Core framework 6 to develop hybrid hardware and software-based encryption. However, we also use the Intel SGX software development kit. Furthermore, we have various nugget packages to provide the required libraries to interact with the Azure cloud.

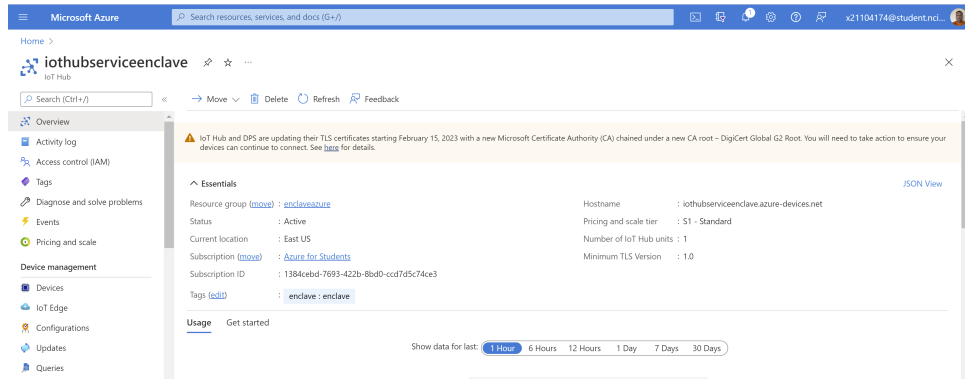

Figure 7: Azure IoT Hub

First, we have set up the Azure. In the Azure portal we have setup the Azure IoT hub service 7. Once the IoT hub is created it generates a primary and secondary connection string which help any application to connect and call that service inside the code. For development We use the visual studio professional 2019 IDE (trail version), valid for a month. Then we created a blank solution, added a console list of projects to the solutions, and chose C language and .Net Core 3.1 framework for complication and execution.

```
public class SimulateDevice
{
    static string IotHubUri = "iothubserviceenclave.azure-devices.net";
    static DeviceClient deviceClient;
    public string deviceId = string.Empty;
    private readonly static byte[] Key = Convert.FromBase64String("AsISxq9OwdZag1163OJqwovXfSWG98m+sPjVwJecfe4=");

    private readonly static byte[] IV = Convert.FromBase64String("Aq0UThtJhjbuyWXtmZs1rw==");
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
        SimulateDevice simulateDevice = new SimulateDevice();
    }

    1 reference
    public SimulateDevice()
    {

        Console.WriteLine("Simulating Device\n");
        deviceId = new RegisterDevice().AddDevice(); //"AsISxq9OwdZag1163OJqwo";//
        deviceClient = DeviceClient.Create(IotHubUri, new DeviceAuthenticationWithRegistrySymmetricKey("hometemperature", deviceId), TransportType.Mqtt);
        for (int i = 0; i < 30; i++)
        {
            SendMessageToCloudAsync(60, 20, deviceId, "dinesh", "password", "");
        }
    }
```

Figure 8: IoT Device Client

Second, we have creates a C console application (Simulate Device) which will mimic the IoT device ability to interact with the cloud. For the scope of the project this application will generate room temperature data only. We need to initialize the device with the help of nugget package library. Microsoft.Azure.Device.Client which provides DeviceClient class to act as the IoT device like features to exchange data with the cloud 8. Furthermore, with the help of Device.Client library we can create multiple devices for various purpose such as sensors.

```
public class RegisterDevice
{
    static RegistryManager registryManager;
    static string strConnectionString = "HostName=iothubserviceenclave.azure-devices.net;SharedAccessKe
    1 reference
    private static async Task<string> AddDeviceAsync()
    {
        string deviceId = "hometemperature";
        Device device;
        try
        {
            device = await registryManager.AddDeviceAsync(new Device(deviceId));
        }
        catch (DeviceAlreadyExistsException)
        {
            device = await registryManager.GetDeviceAsync(deviceId);

        }

        Console.WriteLine("Generated Device key:{0}", device.Authentication.SymmetricKey.PrimaryKey);

        return device.Authentication.SymmetricKey.PrimaryKey;
    }

    1 reference
    public string AddDevice()
    {
        registryManager = RegistryManager.CreateFromConnectionString(strConnectionString);
        return AddDeviceAsync().Result;

    }
}
```

Figure 9: IoT Device Registration

Once we initialize the device we need to register this with the Azure IoT hub for this we need to use the Azure IoT connection string to establish a connection between the device and the cloud 9. For this, we have a use nugget package such as Microsoft.Azure.Devices.Shared this package used to send data to the cloud and Newtonsoft.Json for formatting data. We have also empowered the .Net cryptography library to implement AES encryption and decryption to ensure confidentiality of data should be maintained over the network.

```
Enclave.edl
1  enclave {
2      from "sgx_tstdc.edl" import *;
3      include "Wallet.h"
4
5      trusted {
6          /* define ECALLs here. */
7          public void enclaveChangeBuffer([out, size=len] char* buf, size_t len);
8
9          public int create_wallet([in, string]const char* master_password);
10
11         public int show_wallet([in, string]const char* master_password,  [out, size=wallet_size] wallet_t* wallet, size_t wallet_size);
12
13         public int change_master_password([in, string]const char* old_password, [in, string]const char* new_password);
14
15         public int add_item([in, string]const char* master_password, [in, size=item_size]const item_t* item, size_t item_size);
16
17         public int remove_item([in, string]const char* master_password, int index);
18
19     };
20
21     untrusted {
22         /* define OCALLs here. */
23         int save_wallet([in, size=sealed_size]const uint8_t* sealed_data, size_t sealed_size);
24
25         int ocall_load_wallet([out, size=sealed_size]uint8_t* sealed_data, size_t sealed_size);
26
27         int ocall_is_wallet(void);
28     };
29 };
30
```

```
int create_wallet(const char* master_password){ ... }

/**
 * @brief      Provides the wallet content. The sizes/length of
 *             pointers need to be specified, otherwise SGX will
 *             assume a count of 1 for all pointers.
 *
 */
int show_wallet(const char* master_password, wallet_t* wallet, size_t wallet_size){ ... }


/**
 * @brief      Changes the wallet's master-password.
 *
 */
int change_master_password(const char* old_password, const char* new_password){ ... }


/**
 * @brief      Adds an item to the wallet. The sizes/length of
 *             pointers need to be specified, otherwise SGX will
 *             assume a count of 1 for all pointers.
 *
 */
int add_item(const char* master_password, const item_t* item, const size_t item_size){ ... }


/**
 * @brief      Removes an item from the wallet. The sizes/length of
 *             pointers need to be specified, otherwise SGX will
 *             assume a count of 1 for all pointers.
 *
 */
int remove_item(const char* master_password, const int index){ ... }
```

Figure 10: Enclave.edl and Enclave.cpp

Then we download and install Intel SGX SDK from the intel portal for windows, adding an extension to visual studio to implement the enclave. Once we select that project it will create two file Enclave.edl and Enclave.cpp 10. The both files are based on C

language, whereas enclave.edl is where we define the methods that needs to be execute inside the trusted and untrusted zones and the .Net framework provides a suitable compiler for execution.

Furthermore, to call C methods to the C layer, we first created a COM object in C++, which wrapped C API services and that C++ public method gets exposed to the outside world. The advantage of using COM object is that any programming language that understands DLL will quickly build to make use of enclave encryption.

Intel SGX SDK can be installed and executed on Windows and Linux platforms, but the downside is that it will work only on those processors that support enclave refer ayeks (n.d.). To set up and activate the enclave on the processor, we first need to configure BIOS to enable it. Once the system is rebooted, navigate to service manager and start Intel SGX service. Upon starting the service, the Intel SGX SDK can use the enclave feature in the code.

5.1 Demo Presentation

we have also created a demo videos which will give a detailed walkover on the entire code flow followed by the output. Please follows the video link: Video Presentation or copy paste the URL on the browser `https://web.microsoftstream.com/video/65296dab-c2a4-445c-a464-2a6540e00a2c?list=studio` for a Video Presentation.

6 Evaluation

In our assessment, we have observed that the user critical data such as device, username and password have been encrypted inside the enclave in the form of sealing and only the enclave trusted zone can unseal the data and decrypt it for the system to read. The sealing happened inside the enclave memory, driven by an Intel microprocessor. That ensures hardware-based encryption. For software-based encryption, AES used symmetric encryption, which used a single key for encryption and decryption.

6.1 Experiment / Case Study 1

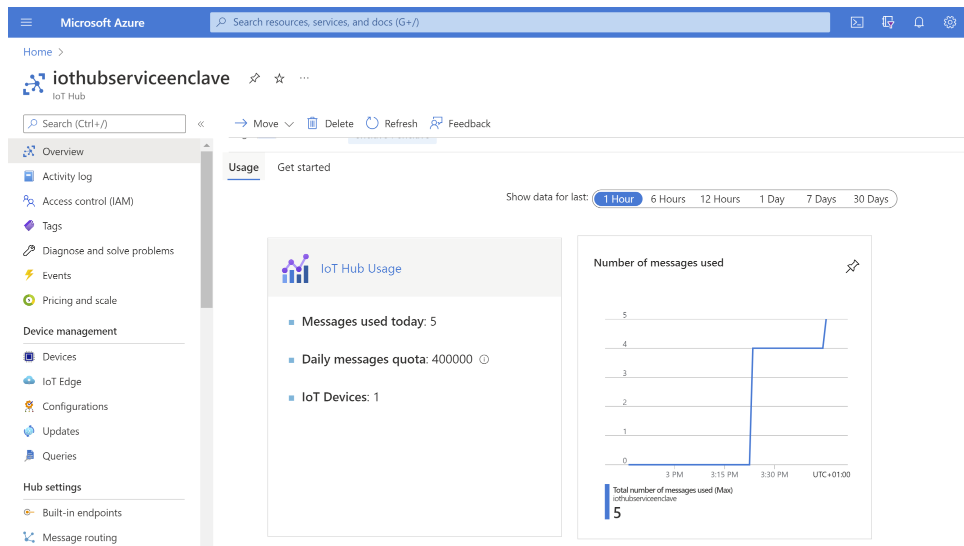

Figure 11: Request served by Azure

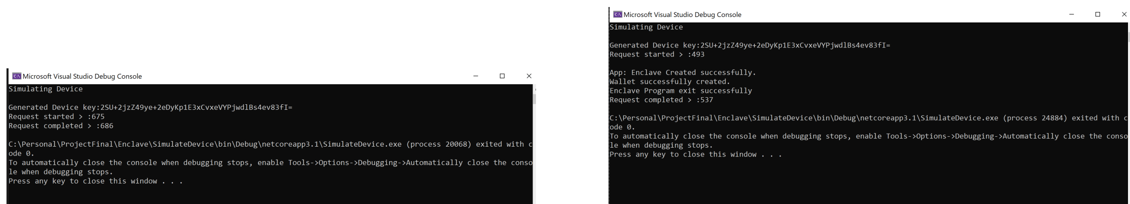

Figure 12: Request time before and after Enclave layer.

We have run the unit test case on our console application. We have noted the total request serve time before the execution and after the completion of processes. We have sent a package of five messages to the cloud 11, logged the time when the application is directly exchanging data with the cloud and logged the time for another request where the enclave authentication layer was a part of the entire execution 12. By comparing both the execution we observed that the request completed with the enclave layer cost 7-8% of the network bandwidth which is acceptable. We have also sent different message package to azure which we have illustrated in below graph.

6.2 Discussion

We assumed that during the transit we have minimised the attack on data as we are sending data using AES encryption and user's personal data are being processed inside the enclave with help of sealing and unsealing functionality and in no way any malicious can get the data access from the enclave. However, we don't have proper tools to verify the enclave but we observed that once the data is passed to enclave it process that information in its own memory as we didn't find any trace of data on our hard drive.

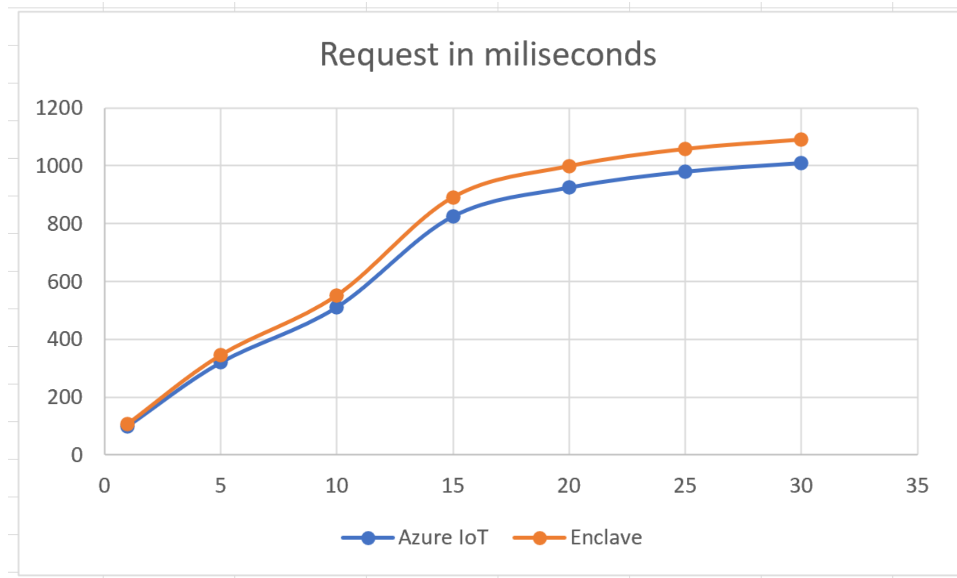

Figure 13: Request comparison between the cloud and enclave

The graph illustrates 13 that the x-axis is the number of requests sent for processing directly to the Azure cloud and request server via the enclave. The y-axis represents the request time in milliseconds. By adding a security layer, the serving time of appeal had been increased by 7-8%, which is marginal head over on the network as we add a layer of security to ensure user's privacy and data confidentiality, Furthermore, this assessment is conducted based on a single device design, but as the device grows, it might increase the request and response time a bit.

7 Conclusion and Future Work

In this research, we are transferring data processing to the cloud; we suggested a novel framework that uses a hybrid computer hardware and software-based solutions model to build a privacy-based IoT data analytic framework for cloud service providers. We have successfully implemented the project in which IoT devices can securely send and receive data from the cloud.

We have to use AES with Intel SGX, which ensures data privacy during the exchange. The enclave provides the additional layer for user authentication, which minimise the attack of a malicious user; the enclave also helps the cloud so that even if the cloud OS get compromised, even then, the hacker wouldn't be able to pull information from the enclave. However, this implementation observed a slight 7-8% overhead on the network while processing the request from the device to the cloud and back. The other drawback of using enclaves is that it is supported by selected intel microprocessors only. The entire implementation can be improved by adding multiple device requests and responses in a parallel thread where we embrace all the features of the enclave. This might again take a bit overhead while serving requests.

Future work can also be suggested as we haven't used the attestation feature of the enclave, which is a stringent way to authenticate the device and then allow the request to

process as it was out of scope. But applying attestation will ensure devices are registered with the enclave. Only those device requests can be served rest will be ignored.

References

ayeks (n.d.). Sgx-hardware.
URL: *https://github.com/ayeks/SGX-hardware*

Correia, C., Correia, M. and Rodrigues, L. (2020). Omega: a secure event ordering service for the edge, *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 489–501. CORE2021 Rank: A.

Elgamal, T. and Nahrstedt, K. (2020). Serdab: An iot framework for partitioning neural networks computation across multiple enclaves, *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pp. 519–528. CORE2021 Rank: A.

Gao, Y., Lin, H., Chen, Y. and Liu, Y. (2021). Blockchain and sgx-enabled edge-computing-empowered secure iomt data analysis, *IEEE Internet of Things Journal* **8**(21): 15785–15795. JCR Impact Factor 2021: 9.471.

Grobauer, B., Walloschek, T. and Stocker, E. (2011). Understanding cloud computing vulnerabilities, *IEEE Security Privacy* **9**(2): 50–57. JCR Impact Factor 2021: 3.573.

Guan, L., Cao, C., Liu, P., Xing, X., Ge, X., Zhang, S., Yu, M. and Jaeger, T. (2019). Building a trustworthy execution environment to defeat exploits from both cyber space and physical space for arm, *IEEE Transactions on Dependable and Secure Computing* **16**(3): 438–453. JCR Impact Factor 2021: 7.329.

Gupta, U., Saluja, M. S. and Tiwari, M. T. (2018). Enhancement of cloud security and removal of anti-patterns using multilevel encryption algorithms, *International Journal of Recent Research Aspects* **5**(1): 55–61.

Iulia Bastys, Musard Balliu, A. S. (2018). If this then what?: Controlling flows in iot apps, *in Proceedings of the 201, ACM SIGSAC Conference on Computer and Communications Security, 2018.*

Jingjing Ren, Daniel J. Dubois, D. C. (2019). Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach, *in Proceedings of the Internet Measurement Conference, 2019*, ACM, pp. 267–279. ERA2010 Rank: A.

Li, Z., Li, W., Xia, Y. and Zang, B. (2020). Teep: Supporting secure parallel processing in arm trustzone, *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 544–553. CORE2021 Rank: B.

Pinto, S. and Santos, N. (2019). Demystifying arm trustzone: A comprehensive survey, *ACM Comput. Surv.* **51**(6).
URL: *https://doi.org/10.1145/3291047*

Sabt, M., Achemlal, M. and Bouabdallah, A. (2015). Trusted execution environment: What it is, and what it is not, *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1, pp. 57–64. CORE2021 Rank: B.

Sardar, M. U., Musaev, S. and Fetzer, C. (2021). Demystifying attestation in intel trust domain extensions via formal verification, *IEEE Access* **9**: 83067–83079. JCR Impact Factor 2021: 3.367.

Sicari, S., Rizzardi, A., Grieco, L. and Coen-Porisini, A. (2015). Security, privacy and trust in internet of things: The road ahead, *Computer Networks* **76**: 146–164.
URL: *https://www.sciencedirect.com/science/article/pii/S1389128614003971*

Sinha, R., Rajamani, S., Seshia, S. and Vaswani, K. (2015). Moat: Verifying confidentiality of enclave programs, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Association for Computing Machinery, New York, NY, USA, p. 1169–1184. CORE2021 Rank: A*.
URL: *https://doi.org/10.1145/2810103.2813608*

Sundareswaran, S., Squicciarini, A. and Lin, D. (2012). Ensuring distributed accountability for data sharing in the cloud, *IEEE Transactions on Dependable and Secure Computing* **9**(4): 556–568. JCR Impact Factor 2021: 7.329.

Tsai, K.-L., Huang, Y.-L., Leu, F.-Y., You, I., Huang, Y.-L. and Tsai, C.-H. (2018). Aes-128 based secure low power communication for lorawan iot environments, *IEEE Access* **6**: 45325–45334. JCR Impact Factor 2021: 3.367.

Wang, J., Hong, Z., Zhang, Y. and Jin, Y. (2018). Enabling security-enhanced attestation with intel sgx for remote terminal and iot, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **37**(1): 88–96. JCR Impact Factor 2021: 2.807.

Yu, Y., Xue, L., Li, Y., Du, X., Guizani, M. and Yang, B. (2018). Assured data deletion with fine-grained access control for fog-based industrial applications, *IEEE Transactions on Industrial Informatics* **14**(10): 4538–4547. JCR Impact Factor 2021: 10.215.

Zhang, W.-Z., Elgendy, I. A., Hammad, M., Iliyasu, A. M., Du, X., Guizani, M. and El-Latif, A. A. A. (2021). Secure and optimized load balancing for multitier iot and edge-cloud computing systems, *IEEE Internet of Things Journal* **8**(10): 8119–8132. JCR Impact Factor 2021: 9.471.

Zhang, Y., Xu, L., Dong, Q., Wang, J., Blaauw, D. and Sylvester, D. (2018). Recryptor: A reconfigurable cryptographic cortex-m0 processor with in-memory and near-memory computing for iot security, *IEEE Journal of Solid-State Circuits* **53**(4): 995–1005. JCR Impact Factor 2021: 5.013.

Zhou, M., Mu, Y., Susilo, W., Au, M. H. and Yan, J. (2011). Privacy-preserved access control for cloud computing, *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 83–90. CORE2021 Rank: B.