# **MASTERS PROJECT DOCUMENTATION**

# **USER PRIVACY IN SMART HOME DEVICES – IoT SECURITY**

# By

# Moulika Bollinadi

# **Advisor: Professor Dr. Steve Tate**

# (Term: Spring 2019 - Fall 2019)

Table of Contents

[**MASTERS PROJECT DOCUMENTATION** 1](#_Toc25321031)

[I. INTRODUCTION 3](#_Toc25321037)

[II. SYSTEM MODEL 4](#_Toc25321038)

[III. THREAT MODEL 5](#_Toc25321039)

[a. What does the attacker know? 5](#_Toc25321040)

[b. What can the attacker access? 5](#_Toc25321041)

[c. What can the attacker do? 5](#_Toc25321042)

[IV. DESIGN DECISIONS 6](#_Toc25321043)

[V. DESIGN FOR BUILDING A SMART THERMOSTAT 7](#_Toc25321044)

[a. Components used 7](#_Toc25321045)

[b. Programming Language used to build the Thermostat 7](#_Toc25321046)

[c. Server used to store the data 7](#_Toc25321047)

[d. Application used to display the content to the user 7](#_Toc25321048)

[VI. DETAILED DESCRIPTION ABOUT THE HARDWARE COMPONENTS USED 7](#_Toc25321049)

[a. DHT 11 Temperature and Humidity Sensor: [8] 7](#_Toc25321050)

[b. Raspberry pi model 3 B+: [9] 7](#_Toc25321051)

[c. Resistor: 8](#_Toc25321052)

[VII. CIRCUIT DIAGRAM 8](#_Toc25321053)

[VIII. WORKING OF A SMART THERMOSTAT 9](#_Toc25321054)

[IX. BLUETOOTH INTERFACE 11](#_Toc25321055)

[a. IN SMART THERMOSTAT 11](#_Toc25321056)

[b. IN ANDROID APP 12](#_Toc25321057)

[X. USERS DAILY INTERFACE 14](#_Toc25321058)

[XI. MULTIPLE DEVICES OPTION 16](#_Toc25321059)

[XII. CONCLUSION AND FUTURE SCOPE 16](#_Toc25321060)

[XIV. REFERENCES 17](#_Toc25321061)

**USER PRIVACY IN THE SMART HOME DEVICES – IoT SECURITY**

# INTRODUCTION

Smart homes and smart home devices are a recent trend in the new technology. Performing and operating the different tasks remotely using an Android or IOS app by authorized user has decreased physical work for the humans at home. Introducing these features into our homes and applying connectivity to everyday appliances and home features has been a huge success, and the concept of Internet of Things (IoT) has been a major research field in both hardware and software engineering. The industry is on the rise and this growth has attracted many companies to jump in. The thermostats from Nest home, Honeywell, Ecobee, and others are top sellers of smart home devices especially smart thermostat. Though the technology has an enormous positive outcome in the corporate world, we need to also understand the security risks and privacy concerns caused by smart technology in our lives. Unless and until customers demand strong privacy controls, their smart home device may become a source of information for intruders.

The OWASP top 10 IoT vulnerabilities are: Weak Guessable or Hardcoded Passwords, Insecure Network Services, Insecure Ecosystem Interfaces, Lack of Secure Update Mechanism, Use of Insecure of outdated Components, Insufficient Privacy Protection, Insecure Data Transfer and Storage, Lack of Device Management, Insecure Default Settings, Lack of Physical Hardening. [1]

This research project concentrates on the user privacy issue in smart home devices. Privacy is one of the major concerns in IoT devices. We considered a smart home device, i.e., Smart Thermostat for testing purposes. Smart thermostats are responsible for controlling a home's [heating](https://en.wikipedia.org/wiki/Heating) and/or [air conditioning](https://en.wikipedia.org/wiki/Air_conditioning). They allow the user to control the temperature of their home throughout the day using a schedule. They also contain additional features, such as sensors and Wi-Fi connectivity and are connected to the Internet. They allow users to adjust heating settings from other Internet-connected devices, such as a laptop or [smartphones](https://en.wikipedia.org/wiki/Smartphones). This allows users to control the thermostat remotely.

We introduced a few methods to secure user privacy by protecting the confidentiality and integrity of the system. Concepts such as how to securely connect the authorized user to the thermostat, how to securely transfer keys between Android app and thermostat, how to securely transfer the data to the Android app and decrypt, how to store the data securely in the server and thermostat are studied and practically implemented. Secure methods such as data encryption, usage of random encryption/ decryption keys, encrypting the data in the server, storing the keys securely in the android app using Android key store object, and random PIN Bluetooth Authentication are implemented in this project.

In our system, we encrypted the whole data in the thermostat using a randomly generated 128 – bit content key and transferred the data to the server. Technically, the data stored in the server is encrypted. We generated a random 128-bit device key in thermostat whenever the user requests the “get key” in the Android app. The content key is encrypted by device key and content key is stored on the server along with the device ID. Each device has a separate device key that allows them to decrypt the content key data after verifying the device ID.

# SYSTEM MODEL

A screenshot of a cell phone

Description automatically generated

**Fig-1: System Model**

The above figure describes the architecture design of the system. The three components used in this project for interaction are: Smart Thermostat, Cloud Server, Android Phone. The system model explains different modes of the data such as Data at Rest, Data in Use, Data in Motion.

1. Data at Rest: It is defined as data that is not actively moving from one device to another or from one network to another [2]. Examples include, hard drive, laptop, flash drive, or archived, database, server, etc. For our project data at rest consists of following locations:

* Thermostat: Plain text of temperature and humidity values, content key.
* Server: Encrypted content key & device ID, encrypted temperature & humidity values along with time.
* Android Phone: Plain text of temperature and humidity values, device key.

1. Data in Use: It is defined as active data which is stored in a non-persistent digital state [4]. For our project data in use is,

* Thermostat & Android Phone: Encrypted temperature and humidity values.

1. Data in Motion: It is defined asdata actively moving from one location to another across the internet or through a private network [3]. For our project data in motion is,

* Temperature and Humidity values, encrypted content key & device ID, device key.

**Note:**

**Content Key** – is used to encrypt the Temperature and Humidity values.

**Device Key** – is used to encrypt the content key.

# THREAT MODEL

IoT devices have many potential threats such as structural vulnerabilities and absence of safe - guards or proper protection mechanisms. The threat model is a process where we can identify these threats, enumerate and prioritize our mitigations. The threat model can be implemented from understanding: What does the attacker know? What can the attacker access? What can the attacker do? [5] The following are the threats that we have predicted for the smart home device – smart thermostat depending upon the above understanding.

## What does the attacker know?

If we think from the attacker point of view, he/she might know the encryption & decryption algorithms, and typical user behavior. According to Kirckhoff’s principle, it is impossible to hide the algorithms. So, we need to always think about how to restrict the attacker from accessing confidential information.

## What can the attacker access?

1. **Thermostat:**

* The attacker might physically access the thermostat if he breaks into the house.
* Once he/she gains the physical access, they have a possibility to turn on the Bluetooth and start pairing.
* An attacker can access the device through the internet if he hacks the Wi-Fi of the house.

1. **Data Transfer:**

* The attacker can view the traffic sent from one place to another. If the data in motion is not secured then, he/she can access the information such as session keys, temperature and humidity values etc.

1. **Server:**

* The data uploaded to the server is a plain text.

1. **Android Phone:**

* Storing keys in a plain text in the Android app.

## What can the attacker do?

1. **Thermostat:**

* If an attacker gains access to the device, the attacker also gains access to device memory, which could be protected by hardware controls, but those attacks and protections are beyond the scope of this project.
* Fixed keys are always dangerous and do not provide confidentiality. If an attacker gains the access to the device, he/she can get the key information and change it.
* If an attacker gets physical access to the device i.e.; for thermostat, he/she would be able to get the stored plain text key information and eventually get the data.

1. **Data Transfer:**

* If we don’t transfer keys and data over a secure channel, the attacker can spoof it.

1. **Server:**

* If an attacker gains access to the server, he/she can read the data stored on the server.

1. **Android App:**

* Consider, if the user mobile is stolen by attacker. If the keys are stored in plain text, then the user can get the key and access the data.

# DESIGN DECISIONS

Based on the Threat Model, we prioritized the privileges and made the decisions. These decisions helped in mitigating the issues to some extent and enhanced the security in the design model.

1. **Thermostat:**

* The data in the thermostat is a plain text. We used AES encryption algorithm for encrypting data & content key to mitigate the plain text issue. Reason for choosing AES over RSA, RSA is computationally expensive and has slower encryption time. AES-128bit key is equal to 3072 RSA keys [7]. This system has a trusted key sharing setup, so, public key crypto is unnecessary.
* AES encryption algorithm is used to protect the confidentiality and GCM mode to protect the integrity of the system. When data at rest is encrypted using AES and GCM, the stored data in thermostat will be a cipher text rather than a plain text and protects the integrity.
* Using fixed symmetric keys or hashing the keys is not a secure mechanism and does not protect the confidentiality. So, we used 128-bit random generation of symmetric content key.
* When a user requests “get key” in the app, we generate a random 128-bit device key and encrypt the content key. We store the encrypted content key along with device ID in thermostat.
* According to principles of least of privilege [6], there is no requirement for the thermostat to store device key information. Therefore, we did not keep any records about device keys in the thermostat.

1. **Data & key Transfer:**

* Not transferring the data and content key over a secure channel would give access to the confidential user data. We decided to send encrypted data & content key from the thermostat to the server. We haven’t used specific secure channel to transfer as the information is already in encrypted form. Even though the attacker can read the captured traffic, it will be a cipher text and they cannot tamper with because of GCM mode.
* Device key should be transferred securely. We decided to transfer device key via Bluetooth as it can range up-to 100 meters. If an attacker wants to get device key information, they need to be within 100 meters range and physically access the device which is a rare scenario.

1. **Server:**

* The data stored in the server is encrypted and no one can decrypt it unless and until they have both content key and device key which is highly impossible.

1. **Android App:**

* We used an Android asymmetric key store object in the Android app for device key storage. The device key obtained from Bluetooth, is converted to "key" object, and the object is stored in the KeyStore. When we need the device key, we obtain it from the KeyStore as a key object, which is used to call the cipher text. Even if attacker gains the access to the user mobile, the Android system provides additional protections to make key recovery difficult.
* According to principles of least privilege [6], it is least priority to store content key in app because it is securely stored in server. So, there is no record about content key in the app.

1. **Other:**

* If user wants to turn on Bluetooth in the thermostat/ app, it can turn be on manually. The attacker needs to first get physical access to the thermostat to turn it on.
* We used random PIN authentication method for the initial Bluetooth pairing. After initial pairing only, the device key is transferred from thermostat to Android app.
* The Bluetooth is turned off automatically in thermostat after 3 minutes for restricting the Bluetooth pairing and key transfer to unauthorized users.

# DESIGN FOR BUILDING A SMART THERMOSTAT

We initially couldn’t get into the devices that are available in the market for practical research. So, for testing purposes, we decided to build our own sample smart thermostat which is similar- to the thermostats available in the market. The components and software used to build thermostat are:

## Components used

* Raspberry pi model 3 B+
* DHT11 Temperature and Humidity Sensor
* Resistor – 10ohm
* Connecting wires
* Bread Board

## Programming Language used to build the Thermostat

* Python 3

## Server used to store the data

* Webdav

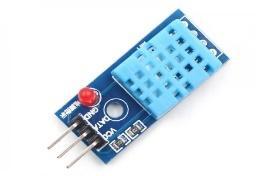
## Application used to display the content to the user

* Android Application
* Programming Language: JAVA

# DETAILED DESCRIPTION ABOUT THE HARDWARE COMPONENTS USED

## DHT 11 Temperature and Humidity Sensor: [8]

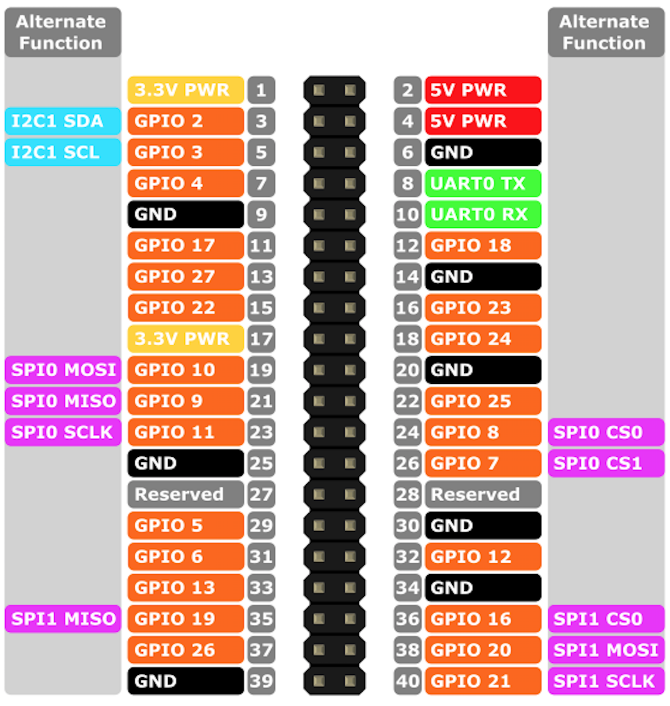
The Sensor has 3 pins. Namely,

* GND – connect to Raspberry pi Pin 6 – GND
* DATA – connect to Raspberry pi Pin 7 – GPIO 4
* VCC – connect to Raspberry pi Pin 2 – 5V Power

**Fig-1: DHT 11 Temperature and Humidity Sensor [8]**

## Raspberry pi model 3 B+: [9]

Raspberry pi consists of in total 40 pins and has separate GPIO description for each pin.



**Fig-2: Pin diagram of Raspberry pi model 3 B+ [9]**

## Resistor:

* Used – 10ohm Resistor
* Color Code: Brown Black Black Brown Brown
* Connected between Data and VCC of the Temperature Sensor to limit the power supply.

# CIRCUIT DIAGRAM

A circuit board

Description automatically generated

**Fig-3: Circuit diagram for building smart thermostat.**

# WORKING OF A SMART THERMOSTAT

A screenshot of a social media post

Description automatically generated

**Fig-4: Working of a Thermostat**

The above figures explain the working of smart thermostat. When the user turns on the thermostat, it reads the temperature and humidity values every minute from DHT11 sensor and displays them on UI interface of thermostat. The obtained data is encrypted using the AES encryption algorithm using the 128-bit random symmetric content key. The AES encryption algorithm is used for protecting confidentiality and the GCM mode for protecting the integrity of the system. The generated cipher text is uploaded to the server and stored in encrypted form. The encrypted data consists of Temperature and Humidity values obtained for every minute. This data is stored with the today’s date file name and uploaded to the server. When the user requests the data for the specific date, the data is decrypted by verifying the file name and download it.

A screenshot of a cell phone

Description automatically generated

**Fig-5: UI interface in Smart Thermostat.**

**A screenshot of a social media post

Description automatically generated**

**Fig-6: Encrypted data stored on the server with the today’s date file**

A screenshot of a social media post

Description automatically generated

**Fig-7: Encrypted data on the server**

A screenshot of a social media post

Description automatically generated

**Fig-8: Sequence diagram of thermostat working**

The sequence diagram of thermostat working is represented above. We can see that, the read\_sensor\_upload.py file sets job to schedular.py to read the generated values for every minute. The SensorData.py file helps to compute the data filename with today’s data and create a folder if there is no existing folder. We then generate an AES key and create a keys.txt file if it is not existing. Whenever the data is generated, we read the AES key from keys.txt file and encrypt the data using the key. The encrypted data is stored in the data file which has today’s date as a filename and the content key is encrypted using the device key upon user’s “get key” request. We upload the encrypted sensor data to the cloud server using the WebdavUpload.py module.

The AESGCMCrypter.py module helps is protecting the confidentiality and integrity of the system. All the keys generated here a random 128-bit symmetric keys. We use the same key for both encryption and decryption, so, the key needs to be secret.

# BLUETOOTH INTERFACE

## IN SMART THERMOSTAT

A screenshot of a social media post

Description automatically generated

**Fig-9: Bluetooth Interface in Smart Thermostat**

A screenshot of a cell phone

Description automatically generated

**Fig-10: Bluetooth UI interface in Smart Thermostat.**

Initially, when the thermostat is switched on, if the user wishes to turn on Bluetooth, he can turn it on in the UI interface of the thermostat. The thermostat Bluetooth is on discoverable mode. It is automatically turned off after three minutes. We chose to transfer the device key to the Android app via Bluetooth. The reason for choosing Bluetooth is because; Bluetooth can be discovered only till a range of 100 meters. For a smart home device to protect the physical security issue by unauthorized users, Bluetooth pairing, and key transfer is a safer option. In this scenario, unless and until the attacker barges into the house, he/she cannot access the device. Even if the attacker gets the access to the device, they cannot get any information as the whole data is encrypted.

A screenshot of a cell phone

Description automatically generated

**Fig-11: Sequence Diagram of Bluetooth interface in thermostat.**

The above figure explains the sequence diagram for Bluetooth interface in smart thermostat. When the thermostat is switched on, it runs the subprocesses internally and displays the temperature and humidity values on UI which is refreshed for every 15 seconds. As the user clicks on “Turn - On Bluetooth”, the bluetoothctl turns on Bluetooth and sets the discoverable on. When the user tries to pair using Bluetooth, the bt\_simple\_agent.py starts a sub process, generates agent with bluez stack to suppress prompt for a random authentication PIN before pairing and then pair the device upon the confirmation. All of this happens when we execute the rfcomm\_server.py.

## IN ANDROID APP

A screenshot of a computer

Description automatically generated

**Fig-12: Bluetooth interface in Android app.**

The user can turn on the Bluetooth in the Android app and look for discoverable devices nearby. Once they find their thermostat name, they can click for pairing options. We used a random pin authentication method for the pairing process. If the device is paired successfully, we click on the “get key” button in the thermostat. When the user requests the “get key” button, the request is transferred to the thermostat via Bluetooth and it generates a “device key”. The “Content key” is encrypted using the “Device key” and the encrypted content key is transferred to the server along with the device ID. After this process is successfully established, we get a “device key” via Bluetooth and it is stored in the Android app. Later, we turn off the Bluetooth or else the Bluetooth is turned off automatically from the thermostat side after 3 minutes. This mechanism is used to secure the device key and safely transfer it.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

**Fig-13: Bluetooth pairing in smart thermostat**

A screenshot of a social media post

Description automatically generated

**Fig-14: Class diagram for Android app.**

The above figure represents the class diagram for each class used in the Android app programming. Whenever the user turns on Bluetooth and pairs with the thermostat, the Android app displays the paired devices using BluetoothActivity.java program. The program displays the bonded devices and obtains the device key from the thermostat via Bluetooth which is decoded decodes by base64. After decoding, the device key is again encoded using base64 and wrapped to store securely in the app using Android KeyStore. The BluetoothDevicesAdapter.java program helps the BluetoothActivity.java to display the list by binding each entry of the data. The KeyStoreUtil.java is a utility class program which generates an asymmetric KeyStore object by implementing RSA encryption algorithm for the device key and stores the device key in the Android KeyStore. The Constants.java program stores the encrypted device key by naming it as decryption key. Whenever the user clicks on “get key”, the BluetoothUtil.java connects to the socket, gets input and output streams and invokes device using Bluetooth socket. After connecting to the server, the FetchWebdavTask,java uses the webdav login credentials to access the server and looks for the keys file to get the encrypted content key and device ID. As soon as the device ID is verified the encrypted content key is decoded and returned. In the background, the encrypted content is decrypted using the Android device key KeyStore object. The device name is displayed rather than the decryption key as it needs to be confidential. When the user clicks on “get data” the MainActivity.java program goes to the given webdav URL and looks for the filename with the corresponding date given by the user. As soon as the file is found, it is downloaded, the content key decrypts the data file and displays it to the user. To obtain the data multiple times, we use FetchwebdavTaskcallback.java program, which calls back the MainActivity.java program multiple times. The FetchBTTaskcallback.java program helps to call the BluetoothAcitivity.java program multiple times which decodes the device key whenever a new one is received.

# USERS DAILY INTERFACE

A screenshot of a social media post

Description automatically generated

**Fig-15: Users daily interface in Android app.**

The user first turns on the Bluetooth in their respective mobile phones and pair the Bluetooth using random PIN authentication method to get the device key. As the thermostat is already turned on, it automatically generates one content key, encrypts the data using the content key and stores it in the device. The thermostat will be readily waiting for the “get key” request from the Android app via Bluetooth. As soon as it receives the request, it generates a random 128-bit device key, encrypts the content key and sends back the device key via Bluetooth to the app. The encrypted content key and the device ID are stored in the thermostat and uploaded to the server. The device key obtained from the thermostat is stored as the Android KeyStore object. When user clicks on “get key”, the Android app first verifies the device ID and downloads the encrypted content key. Later, the data file is downloaded. The device key KeyStore object decrypts the encrypted content key and the content key decrypts the data and displays it to the user upon “get data” request.

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

A screenshot of a computer

Description automatically generated

**Fig-16: Daily user interface in Android app.**

# MULTIPLE DEVICES OPTION

In our design, we can connect multiple devices. Different users can pair to the Bluetooth and get the data simultaneously. Each device has a different device key as they are random 128-bit generations. So, if there is another user “user-2” who wants to connect to the thermostat, he/she will be repeating the same process as above, but the thermostat generates new content key and assigns it for device ID as soon as it sees a new device connection. It generates a new device key for user-2 and encrypts the new content key with it. The encrypted content key and device ID of user are stored together for the verification and uploaded to the server. The rest of the process is repeated for user-2 also.

A screenshot of a social media post

Description automatically generated

**Fig-17: Two different device ID’s and encrypted content key details on the server.**

# CONCLUSION AND FUTURE SCOPE

This project concentrates on user privacy in smart home devices. This report explains about the threat model in the smart thermostat and design decisions we’ve decided to implement in the project. We protect the confidentiality of keys in system by AES encryption algorithm and integrity by applying GCM mode. We generated two different symmetric keys in project i.e.; content key and device key. The data in the thermostat is encrypted using a randomly generated 128-bit symmetric content key and uploaded to the server. Technically, the data is in the encrypted form on the server. We turn on Bluetooth manually on both devices and make a random PIN authentication for pairing. Once the devices are paired, whenever the user requests for “get key”, the request is transferred to the thermostat via Bluetooth and thermostat generates a random 128-bit device key to encrypt the content key. The encrypted content key along with the device ID is uploaded to the server and the device key is again transferred to Android app via Bluetooth. We stored device key in the app using Android KeyStore object which decrypts generates a reference for the device key. The device ID is verified, and the encrypted content key is decrypted using the device key. The content key then decrypts the data and makes it available to the user.

We protected the confidentiality of the system by implementing random key generations, so, that the device key would be separate for every device. The data is encrypted on the server, the keys are securely transferred over the channel and securely stored in the Android app. The integrity is protected using the GCM mode where it adds a verification for the key, so, that it won’t be tampered. Bluetooth pairing is secured by random PIN authentication. Bluetooth is turned off after three minutes to mitigate the unauthorized pairing. We implemented multiple user interface for the system, so, that different authorized users can access the data at the same time by using different device keys. The future scope of this project is to enhance the usability for the user by implementing more techniques. We hardcoded the Webdav login credentials in the system which needs to be removed and automatically verified. This would secure user privacy a bit more. We just used temperature and humidity values for testing purposes but in future we can implement this method for securing the user information such as login ID, password, home address, contact info etc. We did not implement an easy process of controlling the temperature from the Android phone. We can do it as an extension for this project to enhance the usability.

# REFERENCES

[1] OWASP Top 10 IoT Vulnerabilities of 2019, link: <https://www.techwell.com/techwell-insights/2019/01/owasp-releases-latest-top-10-iot-vulnerabilities>

[2] Data at rest, link: <http://aspg.com/three-states-digital-data/#.XdcPC-hKjIU>

[3] Data in motion, link: <http://aspg.com/three-states-digital-data/#.XdcPC-hKjIU>

[4] Data in use, link: <http://aspg.com/three-states-digital-data/#.XdcPC-hKjIU>

[5] Threat model, Dr. Professor Steve Tate’s class notes CSC 681.

[6] Michael T. Goodrich and Roberto Tamassia. Introduction to Computer Security, Pearson, 2011. ISBN-13 978-0-321-51294-9.

[7] NIST Special Publication 800-57 Part 1 Revision 4, January 2016, page no: 53, link: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf>

[8] DHT11 Temperature and Humidity Sensor, link: <https://www.amazon.com/AM2302B-Digital-Temperature-Humidity-Electronic/dp/B07MZS14BQ>

[9] Raspberry pi model 3 B+, link: <https://roboticsbackend.com/raspberry-pi-3-pins/>