Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 165 (2019) 328–335

Procedia Computer Science

www.elsevier.com/locate/procedia

INTERNATIONAL CONFERENCE ON RECENT TRENDS IN ADVANCED COMPUTING 2019, ICRTAC 2019

Secure Hash Authentication in IoT based Applications

Nishant Sharma*, Parveen Sultana H, Rahul Singh, Shriniwas Patil

Vellore Institute of Technology, Vellore and 632014, India

Abstract

Secure authentication while communicating data between sender and receiver nodes is a key concept when providing Internet of Things as a service. Hash algorithms can provide such an authentication mechanism for IoT based applications. The most recent secure hash algorithm technique standardized by the NIST is SHA-3. SHA-3 is fully apt in ensuring authentication for a sender and receiver transaction. A novel signature generating technique based on SHA-3 is presented in the paper. Two of the popular IoT communication models of publish subscribe and request response are examined under this authentication scheme. The sender generates a unique hash code for the data that it is about to send. This hash code serves as the authenticating token for the transaction. Mechanisms for the sender receiver interaction are built into the program codes in both the simulated communication models. The system architecture also manages interaction of the receiver with the cloud. This interaction ensures proper authentication through the mechanisms and api(s) provided by the cloud service providers, and these mechanisms are also integrated into the simulation. Both the cloud services used in the system architecture, i.e. cloud messaging as a service, and cloud database as a service ensure that the sender's identity is verified. In this way a blanket authentication system is set up for the aforementioned IoT architectures.

Peer-review under responsibility of the scientific committee of the INTERNATIONAL CONFERENCE ON RECENT TRENDS IN ADVANCED COMPUTING 2019.

Keywords: Cloud; Hash signatures; IoT; Publish-subscribe; Request-response; Remote messaging; Sensors; Web servers;

* Corresponding author. Tel.: +91-8894645564.
E-mail address: nishant.sharma2018@vitstudent.ac.in

Peer-review under responsibility of the scientific committee of the INTERNATIONAL CONFERENCE ON RECENT TRENDS IN ADVANCED COMPUTING 2019.
10.1016/j.procs.2020.01.042

1. Introduction

The number of active IoT devices is expected to be 10 billion in the year 2020 and is further expected to rise to a massive 22 billion by the year 2025 [20]. This illustrates the potential of the Internet of Things. One such IoT device, raspberry pi [2], is a low cost and versatile computer that has the potential to make Internet of Things more accessible to the developing countries. It is a device capable of acting as both the end node in an IoT network, and as an analytical machine and a central server for a small IoT network. It offers python programming language support.

There are two popular IoT communication models namely publish subscribe and request response. A publish subscribe model with an MQTT broker is considered in this paper. An MQTT broker acts as a server in this model and both publisher and subscriber are the clients. A broker maintains the list of topics to which various publishers publish information to. A subscriber subscribes to a topic of interest and can asynchronously obtain the messages from the broker. A publisher will be a sensor hardware that sends sensed data. A subscriber will be a machine that does further processing with the received data. A publish subscribe architecture is useful for remote communication where the network bandwidth is constrained and a light code footprint is desired [10]. It is also useful when there are many nodes sending and receiving data in a heavy traffic environment. Request response communication model, on the other hand, is a tried and tested model that works well in a low traffic setting. Publish subscribe model is asynchronous whereas request response model is synchronous. Although publish subscribe model is flexible and the message can stay in the message queue of the broker for some time while the receiver is busy, it is prone to memory buffer overflow for a constrained memory broker. On the plus side, for a publish subscribe model deployed in highly critical applications, critical data will be available in the broker even if the receiver is busy. However for a request response model in a critical application, there is a possibility of unproductive pinging between the receiver and the sender to continuously stay up to date. Therefore, request response model is a good choice for a low traffic, non-critical environment, and it is an easier setup.

Nevertheless, in both the publisher subscriber and request response communication models, authentication of the received data is a primary task. SHA-3 (Keccak) provides a secure mechanism for creating a unique signature for data transaction, a hash code that is unique to this transaction. SHA-3 is the most recent among the hash algorithms standardized by National Institute of Standards and Technology (NIST). Consider X1 and X2 as two messages that on application of a hash function H produce H (X_1) and H (X_2). With the birthday paradox, for an n bit hash code, it is possible to generate a hash collision (theoretically) in $O(2^{n/2})$ attempts [24], [31]. However, in practice, it is desirable that the hash algorithm makes it extremely hard to get such a hash collision (which will invalidate the hash algorithm). Hence the choice of SHA-3, which is currently the most secure hash algorithm. SHA-3 has a sponge construction [15]. It has an absorbing phase where the input is read in and processed. It then has a squeezing phase in which the hash code output is generated.

An integration of the IoT system with the cloud is indispensable to the current times. If some critical information needs to be communicated by the receiver machine to some remote client, say an android device, a cloud messaging service can assist in sending an appropriate notification. Simultaneously, the system can upload necessary data to a cloud database. This data can then be further processed to find patterns that provide insights in the relevant area of study. Cloud integration and secure authentication in cloud interaction is considered in the system architecture discussed in the paper.

2. Related Work

The authors of [27] have demonstrated the use of raspberry pi as a video server, showing the machine's computational abilities. The publish subscribe model is discussed in [33]. The authors present an adaptive river monitoring system using publish subscribe model. Here, even though monitoring of huge amount of video data is done, the IoT communication is still resource efficient. Hashing and its properties are discussed in detail in [9]. Computational complexity of universal hashing has been discussed in [21]. The problem of sensitive data exposure in IoT has been addressed by the authors of [23]. A hardware serialization based authentication scheme for IoT based applications is discussed by the authors of [16]. The authors of [8] propose to use physically unclonable functions to provide mutual authentication in IoT communication. Physically unclonable functions are also used in [32] and [11] for authentication purposes. Elliptic curves have been used for one time authentication in [29]. An important

keynote talk on SHA-3 and its comparison with other (hash algorithm) standards is transcribed in [30]. The talk focusses on the hardware aspects of various hashing algorithms, particularly their system level efficiency. A good resource for further study on SHA-3 is [24] and [15]. A detailed survey of Cloud computing systems has been done by the authors of [26]. The use of a real time cloud database for IoT has been discussed in [18]. A home security systemusing IoT, cloud, and remote messaging on an android device is proposed by the authors of [28].

3. System Architecture

The general system architecture considered for simulation in this paper is shown in Fig. 1. Here, the sending device is an IoT node that has a sensor built on it. A sensor is an electronic device that is used to detect various measurements such as temperature, moisture, and distance etc. in the real world. The receiver is an IoT device that receives such measurements and is capable of processing and manipulating this data. It is a relatively (computationally) powerful Linux based machine in contrast to the sender which may be constrained. Any programmable device (such as arduino) may serve as a sender. Providing authenticity to the data using the SHA-3 based signatures is the responsibility of the sender. As such, verifying the authenticity of the data received is the responsibility of the receiver. The receiver is also responsible for evaluating if critical data has been received and send appropriate notification to a remote client device. Cloud messaging service is used to send these critical notifications. The authentic data received is also published to a cloud database. Data may be received from one or more senders and can be stored in the cloud database for further analysis. A communication architecture facilitates the interaction between the sender and the receiver. Two communication architectures of publish subscribe and request response are examined in the paper under the proposed blanket authentication scheme.

4. SHA-3 based unique signatures for a transaction

The message that is sent by the sender is of the form (hash code, data) where the 'hash code' serves as the unique signature for the data transaction, and the 'data' in the simulation is the distance measured by the ultrasonic sensor. This 512 bit hash code is generated by using the SHA-3 algorithm. The hash code is a function of the (distance) data to be sent and the unique hardware serial id of the receiver machine. In the simulation, the unique serial id of the receiver (raspberry pi) is hard-coded into the sender's program code. Pi's unique serial id can be obtained from '/proc/cpuinfo' on the machine. This ensures the usage of the unique hardware characteristics of any chosen IoT node (and previously known to the other nodes in the network) for authentication in addition to the security provided by the SHA-3 algorithm. The unique serial id of the receiver machine is appended to the distance data and the SHA-3 hash is calculated for this string. This hash acts as a unique signature for the authentication of the transaction. Naturally, different transactions will have different signatures.

5. Setup for Simulation

5.1. Raspberry Pi

A raspberry pi client (with raspbian OS) is set up on Ubuntu 18.04 (bionic beaver) with a static ip configured in the /boot/cmdline.txt file and default gateway added for lanaccess.Wifi details are configured in wpasupplicant.conf file in /etc/wpa supplicant directory and ssh is enabled. Using VNC server on pi and remmina on 18.04, a gui login into pi is made.

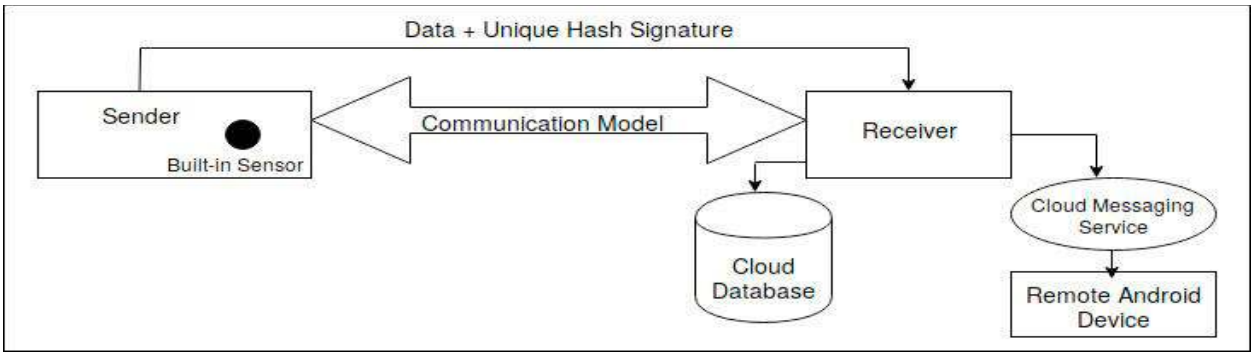

Fig. 1. Secure authentication architecture for IoT based applications.

5.2. Ultrasonic Sensor (HC-SR04)

Ultrasonic sensor is set up on raspberry pi as shown in [25]. Pi conveniently offers python as a programming language to communicate with the ultrasonic sensor. A pre-installed python library in pi called RPi.GPIO assists in this interaction. It is then possible using some basic programming to record the distance measure of a nearby object using the sensor.

5.3. Broker (for Publish Subscribe Architecture)

Mosquitto-mqtt as a broker is set up on the raspberry pi [22]. The broker listens to incoming client requests on port number 1883.

5.4. Firebase Cloud Messaging for Android

Firebase Cloud Messaging can be used to deliver messages across cross platform applications. For the simulation it is desirable to create an android application that will run on a remote android device, that is able to receive push notification from the receiver (raspberry pi) if some critical condition is detected. The setup for firebase cloud messaging for android client application is done by following the instructions in [13]. After the setup, the application is ready to be deployed on an android device. An android emulator with Android 9.+ (Google Play), API 29, 1080 × 1920 xxhdi resolution, x86 CPU with Play Store available, and that supports at least Android Jelly Beans is used for the testing purpose. After the successful launch of the android application on the device, a new token is generated for allowing push notification to be sent to the remote device. This token could be retrieved from the logcat of the running activity. Once the token has been retrieved it can then be used across platforms to send push notifications to this android device.

5.5. Firebase Cloud Database

Firebase Cloud Database can be used to store important updates from the various senders. It can be set up as follows. From project settings in project overview of the firebase web console for a registered firebase project, under service accounts, a private key (a json file) is generated and can be used as a Certificate (security credentials) for authentication. To store data, a collection and a nested document is created in the cloud database for this registered project.

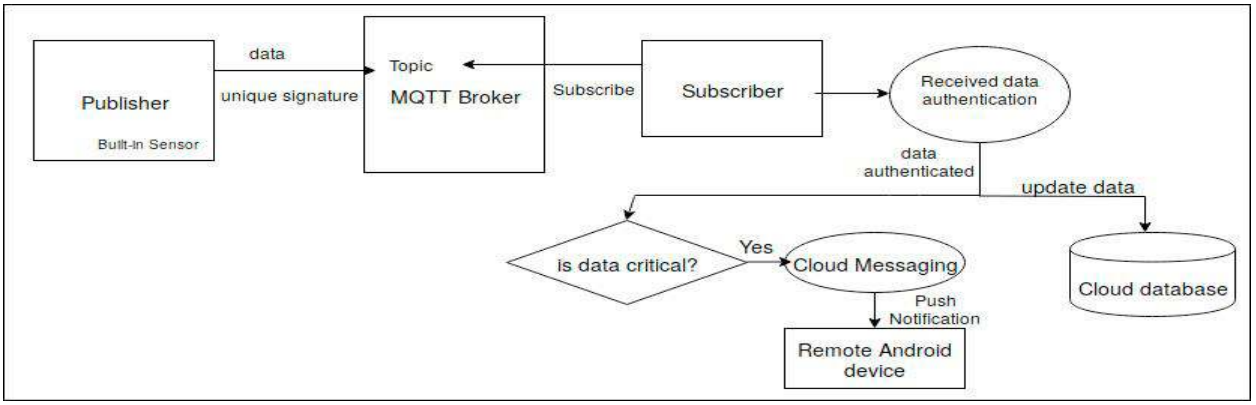

Fig. 2. Publish subscribe communication model extending the general architecture.

6. Results and Discussion

6.1. Publish Subscribe Communication Model

6.1.1. Flowchart

The detailed data flow and processes in the publish subscribe communication model is given in Fig. 2. The working implementations of publisher and subscriber are presented and explained in the following subsections.

6.1.2. Publisher

In the simulation, the publisher and the subscriber is raspberry pi. Hence, the Eclipse Paho MQTT python client library [19] assists with publishing. The python library pysha3 [17] and its member function keccak_512() assist in the creation of the unique transaction signature (a hexadecimal hash code, 'hexcode'). If an arduino with wifi shield is used, then we can use the UDP protocol based string sending as shown in [3] for publishing sensor data to the broker. The C++ implementations for SHA-3 are available at [4] which may be integrated into the arduino's publisher program code.

6.1.3. Subscriber

Subscriber does a few things: it validates the authenticity of the received data, and if the received data is critical it sends an appropriate notification to the cloud registered remote client device. It also updates the distance data and the current time to a cloud database. Since a hash code has a one-way property, it is practically infeasible to generate the data back from the hash code. Therefore the only way to authenticate the received data is to recreate the hexcode at the subscriber's end. If the recreated hexcode matches with the received one, then the authenticity of the received data is verified. It is then that the data (or message) is considered for further processing. For example, if the received message from the (ultrasonic) publisher states a distance that is below a certain predefined threshold, then a critical notification may be sent to a remote client such as an android device. The python library pyfcm [7] assists us in the integration of the subscriber program code and the firebase cloud messaging for android. With the use of server api key for cloud messaging and the unique token generated from the android activity, a successful critical notification is sent to the android device. Fig. 3. shows an instance. Here, the received distance of 1.38 cm is less than the chosen threshold of 5 cm and hence a notification is received by the registered android device. An update to the firebase cloud database can be performed by using the firebase-admin library [12] of python. The major tasks performed during the experiment have been summarized in Table 1.

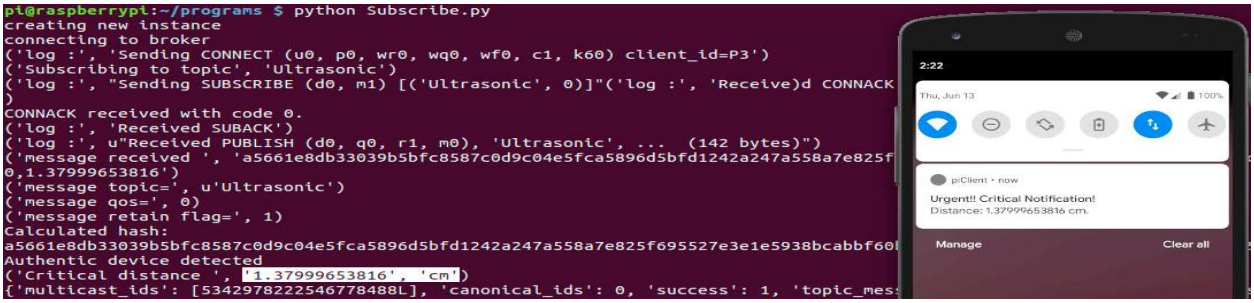

Fig. 3. Subscriber process processing received data.

Table 1. Summary of the tasks performed in publish subscribe communication.

Tasks Performed	Description
Generating the registration token for the android client application.	A token string of the form "e4sIn8qEtNs:APA91bG4pF..." is generated. This registers the android device (activity) to the cloud.
Publishing to the topic "Ultrasonic".	Publisher creates a new instance. Connects to broker. Sends a CONNECT to client. Publishes the (SHA3-secured) message (143 bytes long hexcode + distance) to the topic "Ultrasounic". Receives CONNACK.
Subscribing to the topic "Ultrasonic".	Creates a new instance. Connects to broker. Subscribes to topic "Ultrasonic". Receives message (143 bytes long hexcode + distance). Recalculates hexcode of the received distance data using the pre-known serial id of pi. Verifies with the received hexcode and authenticates data. Evaluates criticality of the data, and (if necessary) sends critical notification to the registered android device using firebase cloud messaging service. Updates the current time and the received (authentic) data to the firebase cloud database.

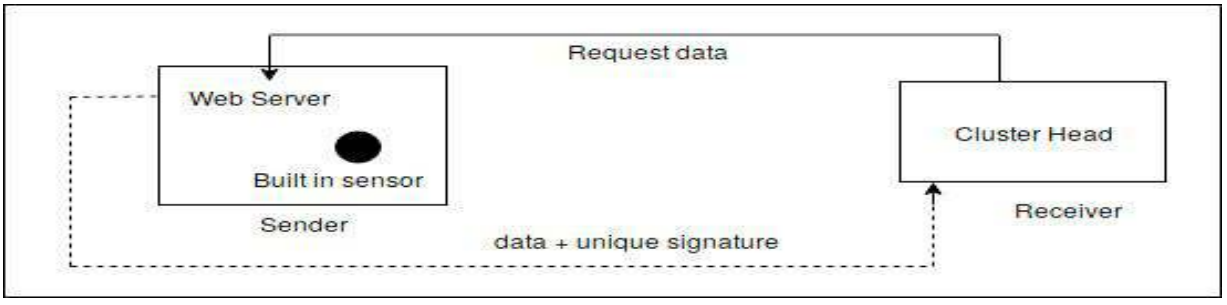

Fig. 4. Request response communication.

6.2. Request Response Communication Model

The request response communication is diagrammatically shown in Fig. 4. The cluster head is responsible for maintaining one-to-one connections with one or more senders.

6.2.1. Sender Process

The term "sender" is used loosely to conform to the general system architecture described above. A sender in this model is basically a web server. A raspberry pi sender is considered for simulation. An arduino based wifi server may also be set up as shown in [6]. This satisfies our condition that the sender may be computationally constrained. Now, the distance measure is obtained from the ultrasonic sensor, and a unique hexcode using this distance data and the serial id of the receiver machine is created as discussed above in the publish subscribe communication model. A web server is then set up on the sender. Flask [1], which is microframework that supports request response RESTfulcommunication is used to set up this web server. The message that is served by the sender is available for retrieval at the ip:port (default port for flask is 5000) combination on the sender machine. A 'Get' request by the receiver will make the 'unique SHA-3 based signature + data' accessible.

6.2.2. Receiver Process

The 'hexcode + data' pair is received at the receiver machine by using the urllibpackage [5] for python. The receiver then verifies the authenticity of the received data by comparing the received hexcode and the recreated one, and manages the interaction with the cloud similar to the discussion in the publish subscribe communication model.

7. Conclusion

The paper discusses a blanket authentication scheme for publish subscribe and request response based IoT architectures that extend to cloud based services. The authentication of data in the interaction between the sender and the receiver in both these communication models is ensured by the usage of the state of the art SHA-3 algorithm. This mechanism creates a unique signature for each data transaction between the sender and the receiver. Verified authentic data received by the receiver is further processed while the inauthentic data is discarded. The interaction between the receiver and the cloud is secured via the mechanisms provided by the cloud service providers. A push notification is successfully delivered to the remote client device via a cloud messaging service by using a unique token for authentication generated during the initial handshake between the device and the service. Updates done to the cloud database by the receiver for the verified authentic data received from the sender are also authenticated by the usage of a private key generated during the initial cloud database setup. This blanket authentication systemis efficient and state of the art, and can be deployed into real world.

Appendix A. Request response model with local PostgreSQL database

A running instance of the simulation of the receiver in request response model with a local postgreSQL database set up at the receiver is shown in Fig. 5. Using the urllib [5] library package in python, hexcode and the data are received from the sender. The received and the recreated hexcodes are matched, and if the data authenticity is verified, it is updated to the postgreSQL database using the psycopg2 [14] python library.

```
/usr/bin/python /home/nishant/nishantsharma5592@gmail.com/MtechV
Received hash:
fd88dbf8816f6261ea1ac202fd933015245b6306110e679dda8a27784cf4cabt
Received data:
16.1183595657
Connected to database!
Calculated hash:
fd88dbf8816f6261ea1ac202fd933015245b6306110e679dda8a27784cf4cabt
Hashes match!
Upload successful!

Process finished with exit code 0
```

Fig. 5. Received data authentication in request response model with a local database.

References

[1] Flask [internet]. Available from: https://palletsprojects.com/p/flask/.

[2] Raspberry pi [internet]. Available from: https://www.raspberrypi.org/.

[3] Send and receive udp string [internet]. Available from: https://www.arduino.cc/en/Tutorial/WiFiSendReceiveUDPString.

[4] Software resources [internet]. Available from: https://keccak.team/software.html.

[5] urllib url handling modules [internet]. Available from: https://docs.python.org/3/library/urllib.html.

[6] Wifi web server [internet]. Available from: https://www.arduino.cc/en/Tutorial/WiFiWebServer.

[7] pyfcm 1.4.7, python client for fcm - firebase cloud messaging (android, ios and web) [internet]. Available from: https://pypi.org/project/pyfcm/.

[8] Aman, Muhammad Naveed, KeeChaing Chua, and Biplab Sikdar. (2017) "Physically secure mutual authentication for iot." *Conference On Dependable and Secure Computing: IEEE*: 310–317.

[9] Babka, Martin. (2013) "Properties of universal hashing."

[10] Banks, Andrew, and Rahul Gupta. (2014) "Mqtt version 3.1.1." *OASIS standard* **29**(89).

[11] Frikken, Keith B., Marina Blanton, and Mikhail J. Atallah. (2009) "Robust authentication using physically unclonable functions." *International Conference on Information Security: Springer*: 262–277.

[12] firebase-admin 2.17.0 [internet]. Available from: https://pypi.org/project/firebase-admin/.

[13] Set up a firebase cloud messaging client app on android [internet]. Available from: https://firebase.google.com/docs/cloud-messaging/android/client.

[14] psycopg2 - python-postgresql database adapter [internet]. Available from: https://pypi.org/project/psycopg2/.

[15] Bertoni, Guido, Joan Daemen, MichaëlPeeters, and Gilles Van Assche. Keccak [internet]. Available from: https://keccak.team/keccak.html.

[16] Hasan, Anum, and KashifNaseer Qureshi. (2018) "Internet of things device authentication scheme using hardware serialization." *International Conference on Applied and Engineering Mathematics (ICAEM): IEEE*: 109–114.

[17] pysha3 1.0.2, sha-3 wrapper (keccak) for python [internet]. Available from: https://pypi.org/project/pysha3/.

[18] Li, Wu-Jeng, Chiaming Yen, You-Sheng Lin, Shu-Chu Tung, and ShihMiao Huang. (2018) "Justiot internet of things based on the firebase real-time database." *International Conference on Smart Manufacturing, Industrial & Logistics Engineering (SMILE): IEEE*: 43–47.

[19] paho-mqtt 1.4.0 [internet]. Available from: https://pypi.org/project/paho-mqtt/.

[20] Lueth, Knud L.(2018) State of the iot 2018: Number of iot devices now at 7b market accelerating [internet]. Available from: https://iot- analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/.

[21] Mansour, Yishay, Noam Nisan, and Prasoon Tiwari. (1993) "The computational complexity of universal hashing." *Theoretical Computer Science* **107**: 121–133.

[22] Eclipse mosquito - an open source mqtt broker [internet]. Available from: https://mosquitto.org/.

[23] Naik, Swapnil, and Vikas Maral. (2017) "Cyber security iot." *International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT): IEEE* **2**: 764–767.

[24] Paar, Christof, and Jan Pelzi. *Understanding Cryptography- A Textbook for Students and Practitioners*, Springer.

[25] Using a raspberry pi distance sensor (ultrasonic sensor hc-sr04) [internet]. Available from: https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/.

[26] Rimal, Bhaskar Prasad, Eunmi Choi, and Ian Lumb. (2009) "A taxonomy and survey of cloud computing systems." *Fifth International Joint Conference on INC, IMS and IDC: IEEE*: 44–51.

[27] Salih, Fatma, and SA Mysoon Omer. (2018) "Raspberry pi as a video server." *International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE): IEEE*: 1–4.

[28] Sarkar, Sourabh, Srijita Gayen, and Saurabh Bilgaiyan. (2018) "Android based home security systems using internet of things (iot) and firebase." *International Conference on Inventive Research in Computing Applications (ICIRCA): IEEE*: 102–105.

[29] Shivraj, V. L., M. A. Rajan, Meena Singh, and P. Balamuralidhar. (2015) "One time password authentication scheme based on elliptic curves for internet of things (iot)." *5th National Symposium on Information Technology: Towards New Smart World (NSITNSW): IEEE*: 1–6.

[30] Sklavos, Nicolas. (2012) "Towards to sha-3 hashing standard for secure communications: On the hardware evaluation development." *IEEE Latin America Transactions* **10**: 1433–1434.

[31] Stallings, William. (2006) *Cryptography and Network Security*, Pearson Education India.

[32] Suh G. Edward, and Srinivas Devadas. (2007) "Physical unclonable functions for device authentication and secret key generation." *ACM/IEEE Design Automation Conference: IEEE* **44**: 9–14.

[33] Wirawan, I. Made, Irawan Dwi Wahyono, Gilang Idfi, and Gradiyanto Radityo Kusumo. (2018) "IoT Communication System Using Publish-Subscribe." *International Seminar on Application for Technology of Information and Communication: IEEE*: 61–65.