Proceedings of the 7th International Conference of Control Systems, and Robotics (CDSR'20)
~~Niagara Falls, Canada~~ Virtual Conference – November, 2020
Paper No. 135
DOI: 10.11159/cdsr20.135

Wireless Sensor Network Security for Smart Home IoT Systems

Aparicio Carranza[1], Xiaolin Chen[1], Heesang Kim[1], Casimer DeCusatis[2], and Harrison Carranza[3]
[1]NY City College of Technology
300 Jay Street, Brooklyn, NY, USA
acarranza@citytech.cuny.edu, xchen0904@gmail.com, hkim018@gmail.com
[2]Marist College
3399 North Road, Poughkeepsie, NY, USA
Casimer.decusatis@marist.edu
[3]Vaughn College of Aeronautics and Technology
8601 23rd Avenue, East Elmhurst, NY, USA
harrison.carranza@vaughn.edu

Abstract – In recent years, the so-called Internet of Things (IoT) has emerged as a transformative trend for both commercial and consumer-grade applications. The IoT includes a mesh of wirelessly networked devices with embedded sensors that monitor physical and environmental conditions, and which share the acquired data sets with other systems. Wireless Sensor Networks (WSN) have become ubiquitous components in modern IoT networks, employing protocols such as Wi-Fi and Bluetooth. Privacy is fundamental to the WSN, since it transmits sensitive personal information that can be misused in the wrong hands. In this paper, we explore WSN security mechanisms including cryptography and secure routing or data aggregation protocols. To demonstrate best practices, an experimental test bed is constructed, consisting of a 3D printed model building equipped with wireless sensors and controlled by an Arduino and a Raspberry Pi 3 model B. A wireless management application for smart phones, which interfaces with the WSN, is also developed. Security properties are investigated using Kali Linux tools.

***Keywords*:** Wireless, Sensor, Network, Secure, IoT

1. Introduction

Wireless networking technology and wireless sensors have advanced rapidly within the past decade. However, many existing commercial applications for the Internet of Things (IoT), such as smart homes and offices, continue to rely on a single Wi-Fi router broadcasting to all devices within range. In this conventional network architecture, all devices share a common IP address and are distinguished by their port numbers (features such as port address translation are often employed in an effort to provide increased wireless network security). Access to the control plane is often not encrypted, and access to the data plane may allow attackers to implant malware on the IoT devices or corrupt data input from multiple IoT sensors. In this paper, we develop a wireless network sensor test bed for a smart home, using low cost components such as the Arduino and Raspberry Pi platforms. Alternate network architectures using leaf nodes with restrictions on peer-to-peer leaf node communication privileges are studied. We also investigate secure routing and data aggregation protocols, and connect our test bed to a cloud service that enables remote monitoring using an Android smart phone. The testbed will be studied in an effort to better mitigate various types of cyber attacks, by deploying SSH, RSA keys, and other features.

For our wireless sensor network experiments, we employed a leaf node architecture with sensor nodes and a gateway router. A sensor node is a wireless device with on-board sensing, data processing, and network communications [1]. The gateway router enables communication between different network protocols in different segments of the network. Each network protocol is isolated on its own collision domain using the gateway routers. There are two common wireless network topologies used in IoT systems, namely "relay nodes", and "leaf nodes". The relay network topology involves interconnecting the source and destination through an intermediate node, such as a laptop with a wireless connection. The source and destination are separated by a distance greater than their wireless transmission range, hence they cannot communicate directly with each other and must depend on an intermittent relay node. This allows for wireless communication even when the transmitter and receiver are far apart. It also speeds up data transmission by selecting the

best available path between relay nodes to reach the destination device. However, a relay network design was not selected for our test bed due to a number of design considerations. Relay networks are not required if all destination nodes are within range of a common Wi-Fi network, such as when designing a smart home or office. Further, it is possible to intercept the wireless signal between the relay node and either endpoint; an attacker can spoof the relay node in order to gain access to sensitive wireless data, or to trick the end nodes into authenticating with a rogue relay node. By contrast, a leaf topology is a data structure consisting of a tree with nodes or vertices at the end of branches, as shown in Fig. 1- A tree with no nodes is called a null or empty tree. A non-empty tree consists of many levels of root nodes and additional nodes that potentially form a hierarchy. Leaf topologies offer several advantages, including ease of implementation, scalability, and straightforward network management. This is the approach we have adopted for our experimental test bed.

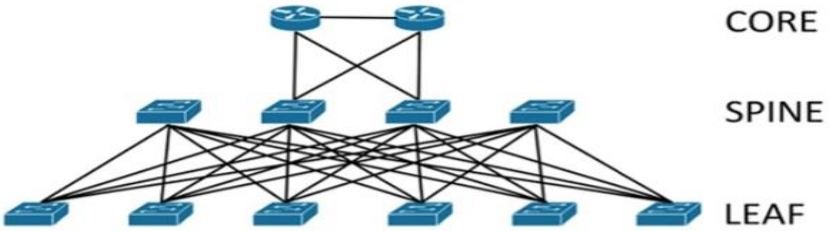


Fig. 1: Topology of leaf nodes in a wireless sensor network.

In our network model, data is first collected from sensors at the leaf nodes and transferred via serial data connection to the Arduino Nano. Next, the Arduino Nano transfers data to the Arduino Uno using a Bluetooth connection, and on to a Raspberry Pi 3 using a USB connection. The Raspberry Pi is also equipped with a Wi-Fi connection, allowing it to upload data to a cloud server. We can manage the entire network using a third party client, such as a mobile phone, with connectivity to the cloud server. This design should be scalable to large IoT installations, and we will demonstrate several security benefits of adopting this approach.

The remainder of this paper is organized as follows. Building on the introductory and background information presented in Section I previously, Section II describes the wireless network hardware components and Section III describes the system environment. Section IV discusses device communication and security features, followed by our conclusions and recommended steps for future research.

2. WSN Hardware Components

Our WSN is composed of the following subsystems: The Arduino board (Uno and Nano), a Raspberry Pi model 3, Bluetooth modules, and sensors. In the following sections, we describe each subsystem.

2.1. Arduino Uno/Nano

The Arduino platform is a single-board microcontroller based on open source software, which provides its own software development environment and libraries. For this project, we begin by using the Arduino Uno, a microcontroller that is based on the ATmega328 microprocessor. As shown in Fig. 2, it has 14 embedded input/output pins and a USB bus for serial communication. The board supports the integrated circuit standard known as I2C, and uses 5 V DC input as a power source, which may be provided from an external battery, power supply, or laptop computer. Programs written on a laptop using the Arduino IDE can be downloaded to RAM on the board for execution [2]. The Arduino Uno is powerful enough to process data from multiple inputs, including the smaller, lower cost Arduino Nano (also shown in Fig. 2). The Arduino Nano is based on an ATmega328p microcontroller, and also features USB connectivity and I2C design features. The Arduino Nano can be configured to directly receive IoT sensor data in our WSN.

Fig. 2: Arduino Uno and Arduino Nano boards.

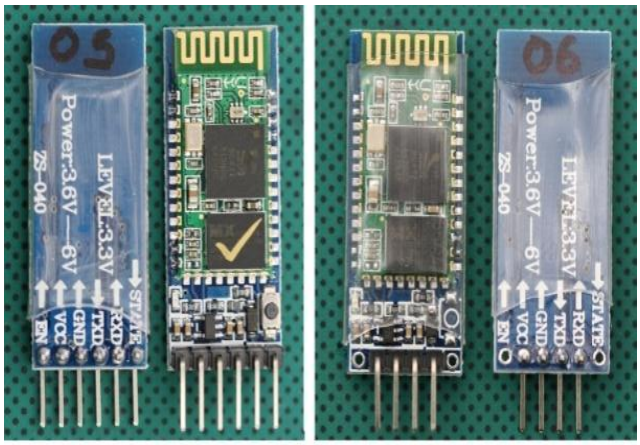
2.2. Raspberry Pi model B

The Raspberry Pi Model B (shown in Fig. 3) is a small single-board computer developed in the United Kingdom by the Raspberry Pi Foundation. It has a 1.2 GHz ARM Cortex-A53 processor with 1 GB DDR2 RAM. It supports wireless connectivity up to 100 Mbits/second Ethernet, 2.4 GHz 802.11n compatible Wi-Fi, and Bluetooth 4.1. It also supports serial connection with a built-in USB port and using the general-purpose input and output (GPIO) pins located on the board. Many different types of sensors can be networked to this device using either the serial ports or wireless connections. The Raspberry Pi uses the open source Linux operating system; many different distributions are available, we used the default "Raspbian" distro in our test bed.

Fig. 3: Raspberry Pi model B.

2.3 Bluetooth Modules

To connect the Arduino and Raspberry Pi, we used the wireless Bluetooth modules HC-05 and HC-06 shown in Fig. 4. The HC-05 is a Bluetooth Serial Port Protocol (SPP) module designed for the Arduino. It supports Bluetooth V2.0+EDR (Enhanced Data Rate) that uses the 2.4 GHz frequency band with data rates up to 3 Mbits/second. The module has CSR Bluecore 04-External single chip Bluetooth systems based on CMOS technology. HC-05 support both master and slave modes of operation, while the HC-06 module supports similar features but only in slave mode. Thus, the HC-05 master can control one or more HC-05 or HC-06 slaves.



HC-05 HC-06

Fig. 4: HC-05 and HC-06 modules.

3. System Environment

First, we need to install Kali Linux on the Raspberry Pi platform. Kali Linux is based on the Debian Linux distribution and is mainly used for computer forensics and penetration testing. Kali Linux has advanced networking features in contrast to other Linux distributions. To install this distribution on the Raspberry Pi, an image file of Kali Linux for ARM processors needs to be downloaded from www.Kali.org and burned into a micro SD card. The hash of this distro can be verified prior to download from the kali.org website. We created the micro SD card image using open source Etcher software, available from https://etcher.io/. By default, Kali Linux uses a limited portion of a virtual memory partition. We recommend expanding the partition size in order to maximize available storage space that can be used for security applications such as RSA keys. From a terminal window type "**apt-get install gparted**." to invoke the GParted partition size editing program.

We also use Virtual Network Computing (VNC), a graphical desktop sharing system that enables control of the I/O for a Raspberry Pi remotely from another device such as a PC or smartphone. To perform a VNC connection, we can download VNC server on a Raspberry Pi (Host) and VNC viewer on an Android smartphone (client). We invoke a terminal window on the Raspberry Pi host and type "**apt-get install x11vnc**". This command will install the x11vnc server on Kali Linux. Then by invoking **"x11vnc -cache 10 -auth guess -nap -forever-loop -repeat -rfbauth /root/.vnc/passwd -report 5900**" we can enable the x11vnc server, which runs automatically when Kali Linux is booting up. On the client we install the RealVNC viewer. Upon launching the x11vnc server, the default port is 5900 which is used to access the Real VNC viewer. Each sensor is serially connected to an individual Arduino Nano in the system, which runs a control program to manage the sensor data. In our test bed, we are using sensors from Adafruit [3] including the temperature and humidity sensor (Si7021), the light sensor (TSL2591), and the PIR motion sensor (PIR189). All the sensors require libraries from Adafruit that are installed in the Arduino IDE library in order for the Arduino to detect the sensors.

4. WSN Security

The Bluetooth component of our design requires Bluetooth modules to connect with all the devices in our testbed, as shown in Fig. 5. The status of the Bluetooth module needs to be configured before setting up the circuit to secure a successful connection. To configure the HC-05, enter the AT command mode in serial monitor in the Arduino IDE. After the main code has been compiled to the Arduino Nano, the connection between the Arduino Nano and the Arduino Uno is complete. To run the Arduino on a Raspberry Pi, the Arduino IDE must be installed by typing "**apt-get install Arduino**" in the terminal window. The Arduino Uno and a Raspberry Pi can be connected by a serial connection using general

purpose input/output (GPIO) from a Raspberry Pi to the Arduino I2C or using the USB cable, which may be more secure and offer faster transfer speeds.

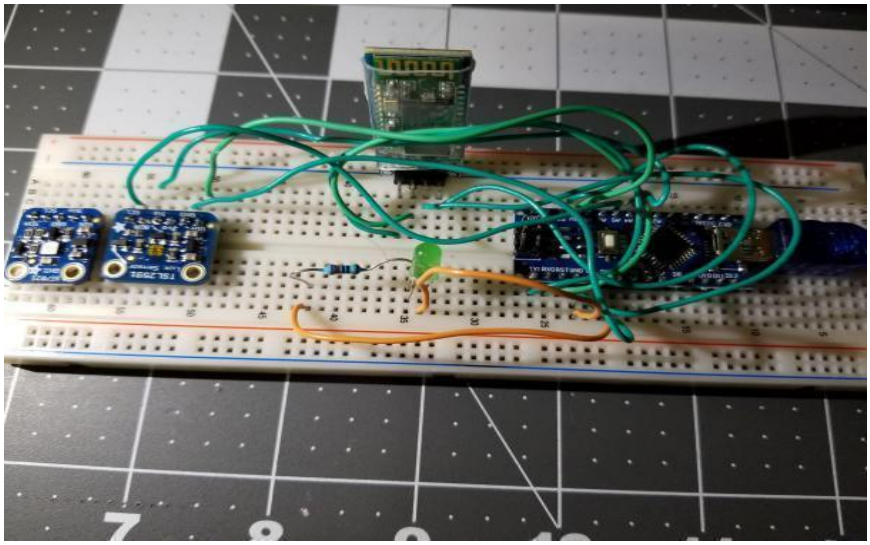


Fig. 5: Bluetooth communication established with sensors attached.

We will protect our WSN using Secure Shell (SSH), a UNIX-based command interface and protocol for secure access to a remote computer. The user can execute commands on the remote system and copy files to another system. SSH is used by network administrators to remotely control various types of servers, and is suitable for the Arduino and Raspberry Pi platforms. SSH features strong authentication methods and the ability to communicate securely over unsecured networks. With SSH, any kind of authentication, including password authentication, is completely encrypted. However, when password-based logins are allowed, malicious users can repeatedly attempt to access the server [4-7]. Users can disable password-based authentication by setting SSH key authentication. SSH keys follow the best practice of using more bits than passwords, increasing their resistance to brute-force attacks; it has been shown [7] that most SSH key algorithms cannot be broken in a reasonable amount of time using commercial or enterprise grade computing equipment. To start SSH in Kali Linux, we type "**service --status-all**" command on a terminal command line interface. This command checks installation of SSH on Kali Linux. To start SSH and check running status we use the commands "**service sshstart**" and "**service ssh status**" respectively. We can modify the configuration by navigating to the SSH directory and editing the GNU sshd_config file using the command "**nanosshd_config**" . We can also change the port number for increased security (the default is port 22). We can further generate an RSA key pair which is saved to a location we specify, using the command "**ssh-keygen-t rsa**" . To apply these setting and restart SSH, we use the command "**service ssh restart**".

Our system design must also include secure data aggregation technology. Data aggregation is used in the wireless sensor network for reducing power consumption and extending battery life [8-9]. The raw data gathered by the sensors will be processed to reduce the amount of repeated data, thereby decreasing the workload on the microprocessors. Data aggregation can offer additional security features, as well. For example, in the WSN test bed, many different sensor nodes are connected to the central controller. If an attacker is able to gain control of one of the Arduino Nanos, they could contaminate the sensor data being transmitted across the network. Alternately, an attacker could upload malware from one Nano to another in a lateral privilege escalation attack, since all the Nanos have similar security privileges enabled. This might allow running malware applications such as crypto currency miners (which are too large to operate on a single Arduino Nano, but could be made to function on a Nano cluster or an Arduino Uno). As shown in Fig. 6, we have designed the network so that the central Arduino Uno controller is the only location that stores sensor data. Further, the Nano controllers that attaches directly to the sensors can only communicate with the central controller; they do not have

privileges to communicate directly with each other, or to permanently store sensor data. This provides an increased level of security for the data aggregation process. Our test bed also includes the concept of secure routing protocols to protect the system from attacks, including mode capture, eavesdropping, denial of service, and more. Secure routing is the foundation of a secure data aggregation protocol.

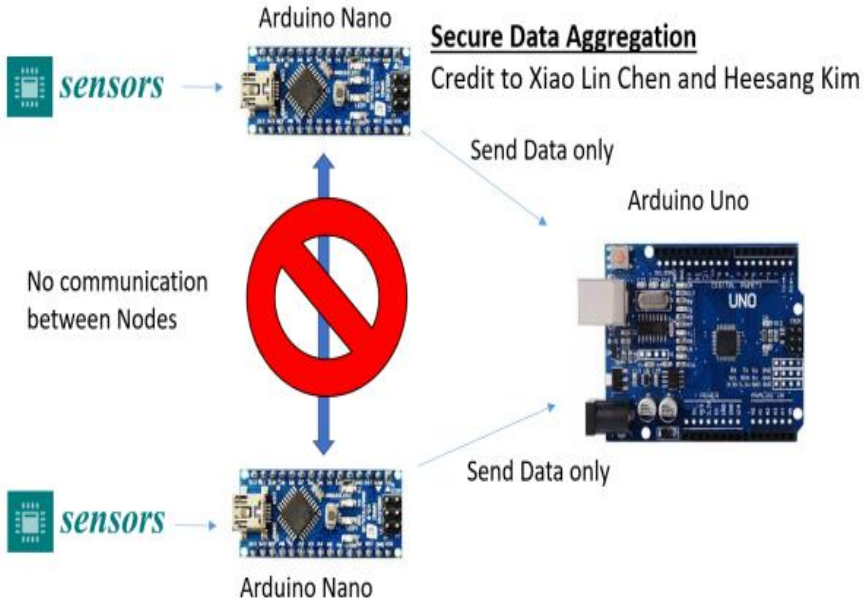


Fig. 6: Secure data aggregation system design.

Our test bed connects the Raspberry Pi to an external cloud server, which can access a mobile phone management application. We used the Google Firebase cloud server with a real time database option. The Raspberry Pi runs Node.js, and open source JavaScript platform. Firebase also requires us to install the NPM packet manager in JavaScript on the Raspberry Pi. In our wireless sensor network testbed, we constructed a simple model of how IoT sensors might be deployed in a smart home to monitor and control lighting, temperature, humidity, and various consumer electronic devices. A 3D printed scale model of a living room with sensors and electronics installed in the ceiling is shown in Figs. 7 and 8. This smart house is a complex multi-function WSN system that can monitor the status of the house from a secure, centralized controller. We have successfully demonstrated resistance to brute force password attacks, denial of service, and data stream corruption (compromising the Arduino Nanos) using this architecture. The real time sensor feed is monitored using Kali Linux as shown in Fig. 9.

Fig. 7: Prototype for smart home model using secure WSN.

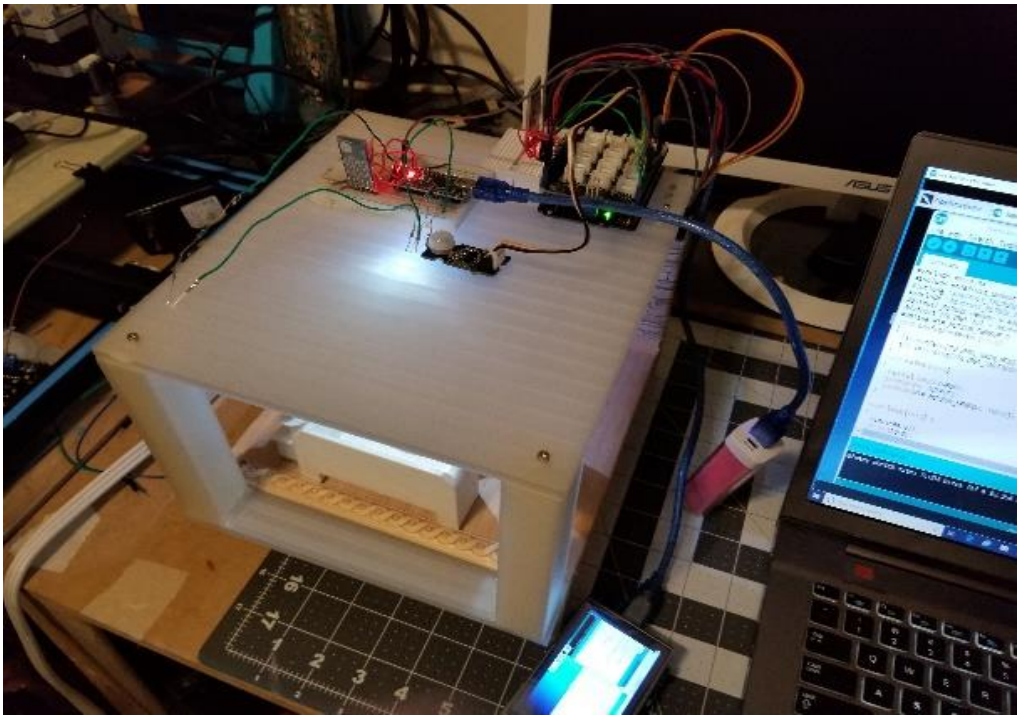


Fig. 8: Placement of electronic circuits in smart home model.

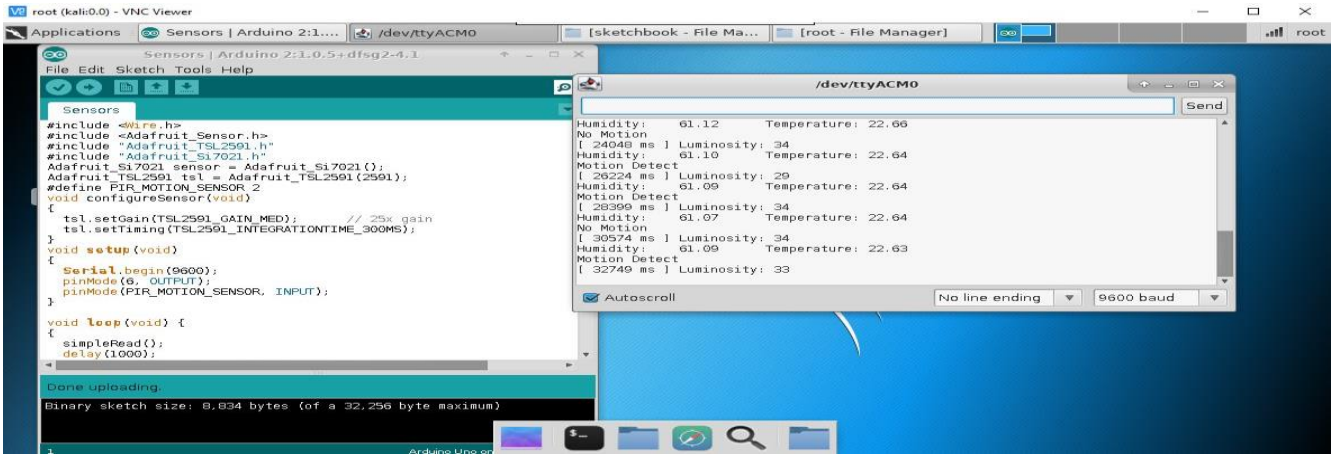


Fig. 9: Sensor data displayed in Arduino IDE running under Kali Linux.

5. Conclusions

Security and privacy are growing concerns for wireless sensor networks, such as those used in smart homes and offices. We have described the construction of a wireless smart home test bed using IoT sensors for lighting, temperature, and other control functions. Rather than simply broadcast to all devices using a single Wi-Fi access point, we implemented a leaf model in which raw sensor data is collected and aggregated using a combination of Ardunio Nano and Arduino Uno platforms. Secure routing and secure data aggregation are implemented on these platforms to mitigate the risk of cyber-attacks against the Arduino platform. Sensor data is uploaded in real time to a Raspberry Pi centralized controller. For security reasons the Nanos are not allowed to directly communicate with each other, and the Raspberry Pi utilized SSH and RSA encryption. The resulting system is connected to a Google Firebase cloud server, which allows us to remotely monitor and control systems using an Android mobile phone application. We can also monitor sensor data in real time using Kali Linux. We have successfully demonstrated resistance to brute force password attacks, denial of service, and data stream corruption (compromising the Arduino Nanos) using this architecture. Future work includes extension of the WSN design to larger distributed sensor networks and hardening the system against different types of cyber attacks.

References

[1] I.Akyildiz, “A survey on sensor networks”. IEEE Communications Magazine, 2002, pp. 102–114

[2] (2019, June). Arduino open source system manual.[Online]. Available: https://www.arduino.cc/en/Guide/Introduction

[3] (2019, July). Adafruit Industries, Unique & fun DIY electronics & kits.[Online]. Available: https://www.adafruit.com/

[4] (2019, July). Instructables.com. AT Command Mode of HC-05 and HC-06 Bluetooth Module. [Online]. Available: http://www.instructables.com/id/AT-command-mode-of-HC-05-Bluetooth-module/

[5] S.Horbaty. (2018, December). ”Security in distributed systems”. [Online]. Available: . https://www.slideshare.net/search/slideshow?searchfrom=header&q=security+in+distributed+systems

[6] P. Oppenheimer, P. (2018, December). “Security Mechanisms Developing Network Security Strategies”. [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=1626588&seqNum=2

[7] J. Ellingwood, J. (2018, December). “ 7 Security Measures to Protect Your Servers by DigitalOcean”. [Online]. Available: https://www.digitalocean.com/community/tutorials/7-security-measures-to-protect-your-servers

[8] H. Alzaid, E. Foo,and J. Nieto, (20180 “ Secure data aggregation in wireless sensor network: a survey”. [Online]. Available: https://dl.acm.org/citation.cfm?id=1385127

[9] J. Sen, and A. Ukil “A Secure Routing Protocol for Wireless Sensor Networks” , Proc. International Conference on Computer Science Applications p. 277-290 (2010).