IgMin Research STEM ISSN: 2995-8067 A Multidisciplinary Open Access Journal

TECHNOLOGY SUBJECT
Network Security | Data Security | Internet Security TOPIC(S)

Article Information

**Submitted:** January 03, 2024
**Approved:** February 09, 2024
**Published:** February 12, 2024

**How to cite this article:** Leshem G, Domb M. Strengthening IoT Network Protocols: A Model Resilient Against Cyber Attacks. IgMin Res. Feb 12, 2024; 2(2): 084-096. IgMin ID: igmin149; DOI: 10.61927/igmin149; Available at: www.igminresearch.com/articles/pdf/igmin149.pdf



Review Article

# Strengthening IoT Network Protocols: A Model Resilient Against Cyber Attacks

**Guy Leshem* and Menachem Domb***

Department of Computer Science, Ashkelon Academic College (AAC), Ashkelon, Israel

***Correspondence:** Guy Leshem, Department of Computer Science, Ashkelon Academic College (AAC), Ashkelon, Israel, Email: gialsm@edu.aac.ac.il

Menachem Domb, Department of Computer Science, Ashkelon Academic College (AAC), Ashkelon, Israel, Email: dombmnc@edu.aac.ac.il

## Abstract

The pervasive Internet of Things (IoT) integration has revolutionized industries such as medicine, environmental care, and urban development. The synergy between IoT devices and 5G cellular networks has further accelerated this transformation, providing ultra-high data rates and ultra-low latency. This connectivity enables various applications, including remote surgery, autonomous driving, virtual reality gaming, and AI-driven smart manufacturing. However, IoT devices' real-time and high-volume messaging nature exposes them to potential malicious attacks. The implementation of encryption in such networks is challenging due to the constraints of IoT devices, including limited memory, storage, and processing bandwidth. In a previous work [1], we proposed an ongoing key construction process, introducing a pivotal pool to enhance network security. The protocol is designed with a probability analysis to ensure the existence of a shared key between any pair of IoT devices, with the predefined probability set by the system designer. However, our earlier model faced vulnerabilities such as the "parking lot attack" and physical attacks on devices, as highlighted in the conclusion section. We present a complementary solution to address these issues, fortifying our previous protocol against cyber threats. Our approach involves the implementation of an internal Certification Authority (CA) that issues certificates for each IoT device before joining the network.

Furthermore, all encryption keys are distributed by the primary IoT device using the Unix OS 'passwd' mechanism. If a device "disappears," all encryption keys are promptly replaced, ensuring continuous resilience against potential security breaches. This enhanced protocol establishes a robust security framework for IoT networks, safeguarding against internal and external threats.

## Introduction

The Internet of Things (IoT) encapsulates a diverse array of interconnected objects and devices utilizing sensors to gather environmental information. This information undergoes analysis, prompting devices to respond to the physical world through actuators [2]. The evolution of IoT technology has become imperative for modern society, facilitating seamless integration of people and things, thus forming intricate information systems through wireless sensor nodes and networks. While sharing similarities with contemporary cyber-physical systems, IoT extends its applications across various sectors, including smart energy grids, industrial control systems, healthcare, transportation, home appliances, and wearables [3]. Despite the operational advantages, integrating IoT technologies has introduced vulnerabilities, offering opportunities for remote adversaries. Real-world incidents and proof-of-concept attacks highlight the emergence of IoT-enabled attacks across diverse sectors [4]. The heightened interconnectivity of previously isolated systems creates novel attack paths for remote adversaries. Innovative communication protocols have been developed for IoT devices characterized by limited processing capacity, constrained memory, and rapid battery depletion to address these challenges. Our prior work [1] introduced a protocol to address key distribution challenges within networks of small, resource-constrained devices susceptible to three distinct attacks: the parking lot attack, exposure of the keys dictionary, and physical attacks. The first vulnerability, the parking lot attack occurs when an attacker infiltrates the parking lot network, gaining access to hosts within the internal network. In our context, the proposed protocol›s initial phase involves determining the Controller device, creating a vulnerability to an «in the parking lot» attacker assuming network control, as illustrated in Figure 1.

The second vulnerability involves the exposure of the keys dictionary. If an attacker successfully intrudes on the device, they can easily access the keys' file, necessitating protection for this critical file. The third vulnerability pertains to physical attacks on the IoT network, as depicted in Figure 2.

Therefore, this research complements the previous work and shows how the three vulnerabilities can be dealt with. The remainder of the paper is organized as follows. Section two outlines

ISSN 2995-8067 DOI: 10.61927/igmin149

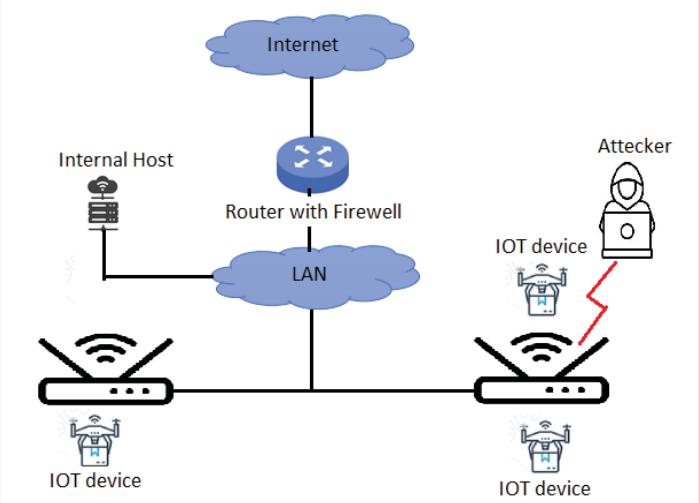



**Figure 1:** Demonstration of the parking lot attack.

**Figure 2:** Smart home concept based on IoT.

three new methods: integrating a local certificate authority method demonstrating how to configure a certificate for each device, a novel mechanism for securing the key dictionary, and a technique ensuring network security even if a device disappears. The implementation details are explained in Section 3. The upgraded and invulnerable protocol is presented in Section 4. Experiment results are detailed in Section 5. Finally, conclusions drawn from this research are presented in Section 6.

## Related work

Eschenauer and Gligor [5] introduced a key-management scheme incorporating selective distribution and revocation of keys to sensor nodes. This scheme relies on probabilistic key sharing within a random graph, utilizing protocols for shared-key discovery, path-key establishment, key revocation, re-keying, and incremental addition of nodes. Notably, their work does not address IoT resource constraints, a focus of our research. Alagheband and Aref [6] analyzed the applicability of public key cryptography, pre-shared keys, and link-layer-oriented Key Management Systems (KMS) for sensor nodes within the IoT context. Their proposed solution employs predefined keys, while our work emphasizes the dynamic construction of keys, allowing for swift responses to unexpected events. In this context, we highlight several Key Management System (KMS) solutions tailored to the IoT domain that differ from our approach. Sciancalepore, et al. [7] proposed a Key Management Protocol for mobile and industrial IoT systems, emphasizing robust key negotiations, lightweight node authentication, fast re-keying, and efficient protection against replay attacks. Their solution leverages ECC constructions, key exchange, and implicit certificates, facilitating seamless integration into security protocol exchanges like 802.15.4. Roman, et al. [8] proposed key management mechanisms facilitating the negotiation of specific security credentials between two remote devices, providing shared keys for sensors within the same network. Wazid, et al. [9] designed a secure, lightweight three-factor remote user authentication scheme for IoT, featuring automated validation of Internet security protocols, offline sensing node registration, and sensing node anonymity. Benslimane and BenAhmed [10] introduced a lightweight key management protocol enabling constrained nodes to transmit captured data securely to an internet host. Mahmood and Ghafoor [11] proposed an Efficient Key Management (EKM) scheme tailored for multiparty communication-based scenarios. Their session key management protocol employs a symmetric polynomial for group members, with the polynomial generation method incorporating security credentials and a secure hash function. While these works contribute valuable insights to IoT key management, our approach distinguishes itself by addressing resource constraints through dynamic key construction, ensuring a responsive and secure network environment. In the subsequent sections, we detail three novel methods to fortify our protocol against vulnerabilities. An overview of new IoT protocols and security threats appears in [12-14].

## Our contribution

Our research contribution focuses on the development of three new methods aimed at addressing vulnerabilities within the local networks of IoT devices. Firstly, the “Local Certificate Authority” method helps prevent parking lot attacks. Secondly, the “Own Keywords Dictionary” method mitigates the risk of dictionary attacks on keys. Lastly, the “Detecting Missing Devices” method helps mitigate the risk of physical attacks on IoT devices within the local network.

## The new methods

Our proposed solution incorporates three novel methods to enhance the security of the IoT network: The Own/Local Certificate Authority method, the Own/Local Keywords Dictionary method, and the Detecting Missing Devices Method.

### Own/local certificate authority method

To counter the Parking Lot attack, where a malicious device poses as the Primary to seize control of key generation and distribution, we introduce the Own/Local Certificate Authority method (Figures 3,4). This method involves the following steps:

ISSN 2995-8067 DOI: 10.61927/igmin149


```
-----BEGIN CERTIFICATE-----
MIIC2TCCAcGgAwIBAgIUXn4msF6ONA8lWcehVqd1xxdRvYkwDQYJKoZIhvcNAQEL\nBQAwEjEQMA4GA1UEA
wwHQ29ycCBDQTAeFw0yMDA0MjcxODA0MjBaFw0yMjA4MDEx\nODA0MjBaMBIxEDAOBgNVBAMMB0NvcnAgQ0
EwggEiMA0GCSqGSIb3DQEBAQUAA4IB\nDwAwggEKAoIBAQC8JqeBHwVnmJkeOKLwqMcil/nY4QBLDsAg4LK
hhzFAB/SvJ16F\norqip2jLuRhpxrPNUYa9p8+ZPZziAL7ir68csnJI+UlLU7XV3+TghiaHVsd4lVz7\nHB
RhMLQcFQvnEyC5sfm84fptetlL4HN8jJUda/M26kxlHidJRCL221R9g+/RI113\n73tBX7iZSAcBTv/sOnd
EjVquYipOQXIZwRJ4ZXZ29K4UdoW+9iMCvhtVPCHz4FEl\nPBFn2vuqRg13EcZ6X3/83VJaO5TSh7Qzl87M
VmfBtGBWvib5gXxPEY1zOnhojfxc\nEPkffyHauwyORFkpaE00LkrkNjxNEQ5qhCKHAgMBAAGjJzAlMBIGA
1UdEQQLMAmC\nB0NvcnAgQ0EwDwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAZyMd\n5eu7
6geBT8yobTyovhPUq63+9BWvmUViNhukZSFX1zKI/8NG1QrAEwG1Rai2yTU/\n07s5XBRwGIcRuFC1tcT7o
qAjHYDQw+3RgYYd+isPUo3Mi7SSWQYpJWmk7ICmqYzy\nlS5uk4iZatPWFVwL4XcH9ssgTVTK3kIdG9LKPP
z/4KwlBQISxYi5u9pSwCum+gIS\nx2+Vc7jJGCUEP1iMLPuxpOHIns9FusfzPfRfApFQRqZfxBO2Hpewoj1
pbb6HckAJ\nVlOyV5KcAunC9UsUtliwN3eFef+U/tNakYtcZjzqn1R5hlLBfaENCwdG4pdvuFw7\na/a5r9
CF+SDw0tldZw
-----END CERTIFICATE-----
```


**Figure 3:** An example of the generated certificate for device #1.

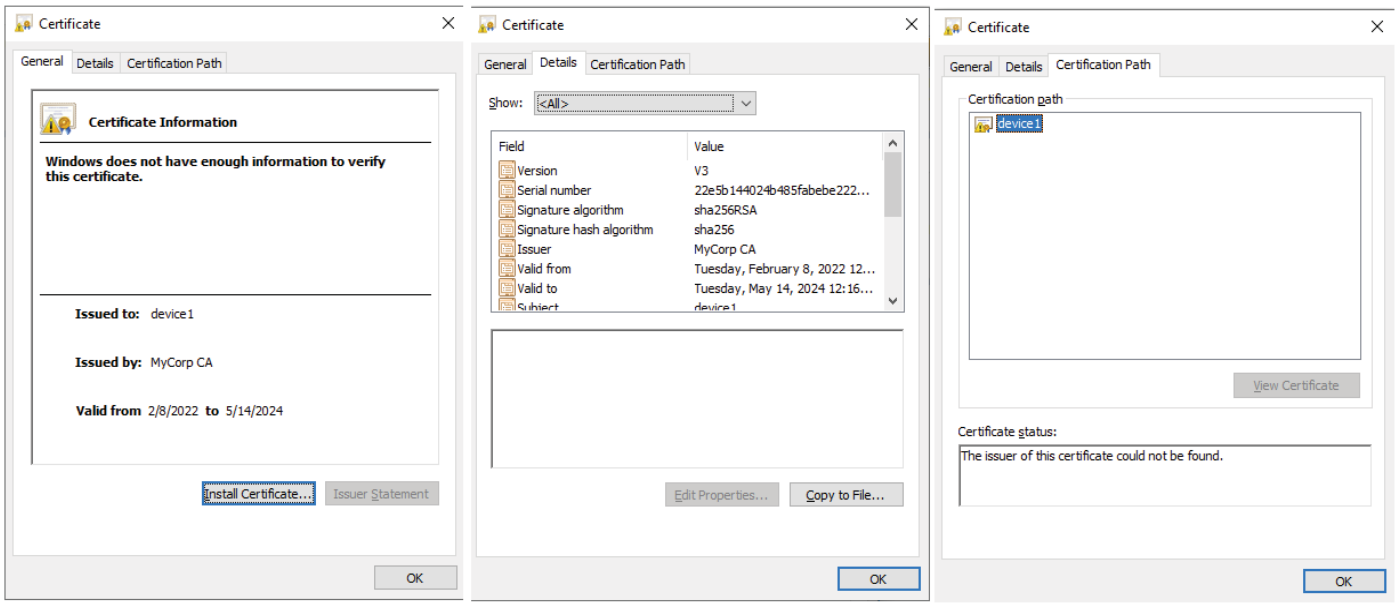



**Figure 4:** The Certificate of device#1 (for a sharp image, zoom in).

## Step 1: Certificate creation

Before a device enters the network, it undergoes an "initial installation," during which a unique encrypted code is installed. This code serves as a certificate, indicating the device's trustworthiness to enter the home network, define itself as a Controller or node, and communicate with other devices in the network [15].

## Step 2: Finding a controller in the home network

An identification step is initiated for each device before it connects to the network. Only after identification can the primary be determined (if one does not exist in the network) or identified as a device in the home network.

## Step 3: Creating a daemon

A daemon, a background process, is created to check the device's reliability. Once connected to the network, the daemon verifies the certificate file's existence, correctness, and reliability. Consider that the certificate file may not exist, is not valid, and/or is invalid (fake).

## Step 4: Creation of a key file and its encryption

During the initialization of the Controller, a "Key Pool" is created, encrypted, and saved in the device's memory. This "Key Pool" is the heart of our method, because the proposed protocol relies on probabilistic key sharing among the IoT devices of a random graph that ensures the existence of a common key between any pair of IoT devices in a predefined probability which is set by the system designer.

## Step 5: Creating a public key for the home network

The Controller generates a public key for the home network.

When a new device joins the home network, it receives, from the Controller, the shared public key and a group of keys with randomly selected indexes from the key pool. They ensure the home network's public key is shared in a conversation between two devices.

## Step 6: Communication between two devices on the home network

It checks to ensure that both devices are from the same home network and that the public shared key for the home network is equal. They find a shared key to encrypt messages and securely exchange information.

## Step 7: Sending encrypted messages

When a device wants to communicate with another, it sends an encrypted message using AES encryption. The encryption key is the shared key. The receiving device decrypts the message using the same key.

## Own keywords dictionary method

We introduce the Own Keywords Dictionary method to mitigate the risk of dictionary key attacks on an IoT device within the local network. This could lead to key discovery and compromise the network. This mechanism operates like the safeguarding mechanism for password and shadow files in the UNIX system. Access to the keyword file is restricted, mirroring the root's protection of critical system files (e.g., *cat/keyword: Permission denied*). The keyword file comprises the following data:

- Device name - visible data
- Key index number - encrypted data (SHA256)
- Keys - Encrypted data (SHA256).
- Last key change time - visible data
- Key Expiration Time - Visible data

An example of the records in the ***keyword*** file (Figure 5).

## Detecting missing devices method

To prevent a malicious opponent from "snatching" a device and extracting the information required to hack the network, a dedicated daemon in the Controller device checks all active devices. The test will be performed as follows:

1) For each time frame (e.g., 20 minutes), the Controller sends a ping to each IoT device and maintains a list of all those who returned an answer (Reply from) and those who did not (Request timed out).

2) Select those who did not return an answer from the list, wait a time (e.g., 3 minutes), and send a ping again.

3) After three unanswered attempts, the Controller exchanges keys for all IoT devices in the local network (Repeat steps 3 through 6).

An example of the dedicated daemon (just in the Controller) that starts running immediately after distributing the keys to all devices:

1) Every 20 minutes, ***ping*** to all devices in the local network

***>> ping device-1***

*Pinging device-1 with 32 bytes of data:*

*Reply from device-1: bytes=32 time=222ms TTL=115*

…

*Ping statistics for device-1:*

*Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),*

…

***>> ping device-2***

*Pinging device-2 with 32 bytes of data:*

*Request timed out.*

…

*Ping statistics for device-2:*

…

…

***>>ping device-n***

…

2) Since there is no response from device 2, after 3 minutes

***>> ping device-2***

*Pinging device-2 with 32 bytes of data:*

*Request timed out.*

…

*Ping statistics for device-2:*

…

3) Since there is no response from device 2, after 3 minutes

***>> ping device-2***

*Pinging device-2 with 32 bytes of data:*

*Request timed out.*

…

*Ping statistics for device-2:*

…

4) Immediately after the third failure, the Controller sends

ISSN 2995-8067 DOI: 10.61927/igmin149

| Device name | | | | | | | |
|---|---|---|---|---|---|---|---|
| Index | Index of Keys Not Part of the file | Keys Not Part of the file | Salt | SHA256(Index of Key) | SHA256(Keys) | Last key change time | Key Expiration Time |
| 1 | 107 | 123 | 1100110 01100 | 3346f2bbf6c34bd2dbe 28bd1bb657d0e9c3739 2a1d5ec9929e6a5df47 63ddc2d | fd2da12cc87478c1f d503ce40b11e77160 9163eee279ba4db71 64c628ff6f2ee | 18:00 01/01/22 | 18:00 01/02/22 |
| 2 | 1056 | 456 | 0000111 10000 | e8c5e943ad4fd9d115c 2baacd110acddee7f66e c24aa177efa6780f5641 ce277 | 92847366066d8b97 3414ccd211d932c25 5b37282e7428f2b43 067ed355eae72d | 18:00 01/01/22 | 18:00 01/02/22 |

**Figure 5:** The structure of the keyword file.

new keys to all devices and sends a cancellation message on the previous keys.

## System implementation

1. **Certificate authority method: Creating a certificate:** Before a device enters the network, it undergoes an "initial installation," during which a file is installed. This file contains a unique encrypted code certifying the device's trustworthiness to enter the home network, define itself as a Controller or node, and communicate with other devices. The certificate creation process occurs as follows: An installer runs an installation file during the initial installation on the device. At the end of the installation, a file is saved in the device›s memory containing a unique code of approximately 1600 characters. When the device connects to the network, it is required to ensure the existence of the certificate file. A daemon continuously runs in the background to verify the file›s presence and integrity, preventing hacking attempts. If the certificate file is not found, the system identifies an intruder, issues a warning, and halts all processes on the device.

2. **Reliability check:** When the device is connected to the network, a unique daemon operates in the background to verify the certificate file's correctness and existence. If the file is incorrect or missing, the system deactivates the device from the home network and encrypts and blocks the key file to prevent hacking attempts.

3. **Creating a shared public key for the home network:** The Controller generates a public key saved in the certificate file. When a device connects to the network, the Controller sends the shared public keys to the home network. When two devices initiate a conversation, a check ensures that both devices share the same home network's public key. Only then can the devices communicate and exchange messages.

4. **Creating and encrypting the key file:** When a Controller device connects to the network, it generates a file of keys shared with the devices. Each device receives a list to find a shared key for encryption. After the file is created, it is encrypted to protect its contents. Devices use the shared key to encrypt messages passing through the network. Implementing the Certificate Authority Method establishes a robust framework for secure key management and communication within the IoT network, safeguarding against unauthorized access and potential attacks.

**Own keywords dictionary method:**

**Pseudo code for the key security mechanism**

***For every device:***

- *Get a list of key indices from the device interested in communicating.*
- *Encrypt all indexes received and get a list of encrypted index results.*

***Loop:***

*Compare the encryption result (of each index) to all results in the "Index of Key" column*

o *If a match was found:*

*Move to the "Keys" column in the same row, decrypt the encryption result, and return it*

o *Otherwise:*

*Return "match was not found."*

***end loop;***

The root also protects this.

**Detecting missing devices method:** The following Python code implements the "Detecting Missing Devices" Method (Figure 6).

**The new immune protocol**

Following, we describe the protocol in further detail:

1) **Preliminary step**

- The local network administrator, as Certificate Authority, must update a local certificate for each IoT device connected to the network; without the certificate, the device will not belong to the network under any conditions.

ISSN 2995-8067 DOI: 10.61927/igmin149

```
import time
import os

hostnames = [
   '10.40.161.2',
   '10.40.161.3',
   '10.40.161.4',
   '10.40.161.5',
]

For hostname in hostnames:
   time.sleep(1200)
   response = os.system('ping -c 1 ' + hostname)
   if response == 0:
      print(hostname, 'is UP')
   Else:
      time.sleep(180)
      response1 = os.system('ping -c 1 ' + hostname)
      if response1 != 0:
        time.sleep(180)
        response2 = os.system('ping -c 1 ' + hostname)
        if response2 != 0:
           print(hostname, 'is down, Controller sends new keys')
           Fernet(key)
        else:
                print(hostname, 'is UP')
      Else:
          print(hostname, 'is UP')
```

**Figure 6:** Python code for detecting missing devices.

- A daemon will run according to the certificate's expiration date, which will create a new certificate for each device, make sure that the file exists, is a valid and reliable certificate, and makes sure that the file exists, is a valid and reliable certificate.

2) **Determining the controller device** as soon as an IoT device enters the network, it broadcasts two messages on the network: "IS THERE Controller & myCertificate" and waits for some time for a response:

- If it receives an "I AM the Controller" message, it switches to "Controller FOUND" and stops the search for the Controller. From then on, it will only listen to messages from the Controller (to get the keys and perform the rest of the protocol stages) until a secure network is established.
- If it does not receive any answer (twice or there is a failure in the network), it will declare itself a Controller and move to "Controller FOUND." It broadcasts two messages: "I AM the Controller" and "myCertificate." If it receives an "IS THERE Controller" message, it will send a private message to that "I AM the Controller & myCertificate" device.

3) **The Controller defines the key pool size**

- The Controller calculates the number of keys it must produce and the number of keys it must divide to each node according to physical data such as memory size and the number of nodes in the network. By using mathematical calculations, conclude the size of the pool.
- The Controller generates a public key, which he saves in the certificate file.

4) **The Controller requests the manufacturing of distributed keys**

- The Controller sends "Controller NEEDS KEY &

myCertificate" to each node in the network about the number of keys it must generate.

- The Controller generates a private and public RSA key and sends its public key to each node in a "Controller PUBLIC KEY" message. The safety of the proposed method requires the transfer of secure data (for example by RSA) between the devices because this data includes encryption/ decryption keys and certificate files.

5) **Creation of distributed keys by each node in the network**

- Each node in the network creates keys, encrypts them with a public RSA key (of the Controller), and sends them to the Controller.
- The Controller accepts the generated keys from each node and consolidates them into a key pool.

6) **Distribution of keys**

- Each client (IoT device) generates a private and public RSA key and sends the public key to the Controller in a CLIENT PUBLIC KEY & "myCertificate" messages so that the Controller can encrypt the subset of keys it sends to the client.
- The Controller raffles k keys from the pool and sends the client each key in the CLIENT RING KEYS message while it is encrypted by the client's public key with the index of the key.
- Once all keys are sent, the Controller sends a CLIENT RING END message, and the client switches to CLIENT GOT KEYS and decrypts each key sent to it with its private key.
- A unique daemon starts to run in the background, and if the file is incorrect or does not exist, the system will deactivate this device from the home networking (to which it was connected), encrypt, and block the key file against hacking.

7) **Finding a shared key**

- Each node that receives keys sends an "I AM ON THE NETWORK & myCertificate" message so that the other nodes on the network recognize it.
- When two devices start a conversation, a check is made that verifies that both devices have the same home network's public key,
- Node#1 will send a "verify certificate" message to node#2, and node#2 will return a public key in response. Node#1 will check that the public key returned by Node#2 is the same as the one it received from the Controller.
- When a node wants to exchange encrypted messages with another node in the network, it sends a "CLIENT START SESSION & myCertificate" message with its list of indexes (of keys). The node that receives the message checks for overlaps between its indexes and the indexes (of keys) it received. If it finds a shared key (which will happen with a probability of 80% - 90%), it sends the CLIENT COMMON INDEX message with the shared key, and in the absence of a shared key, it returns a message of -1. The two nodes maintain the index of the key with which they will exchange encrypted messages.

8) **Secure network**

- When a node (IoT device) wants to communicate with another node, it sends a MESSAGE ENC DATA message in which AES encrypts the information it wants to transmit, and the encryption key is the shared key. The receiving node decrypts with the same key.
- A new node that enters the network goes through the protocol steps and can communicate securely with any node on the network.
- When a node disconnects from the network, it does not interfere with the communication between the other nodes, nor does it reveal all the encryption keys.

9) **Missing devices detected**

- For each time frame (e.g., 20 minutes), the Controller sends a ping to each IoT device and maintains a list of all those who returned an answer (Reply from) and those who did not (Request timed out).
- Selects from the list those who did not return an answer, wait a time (e.g., 3 minutes), and send a ping again.
- After three unanswered attempts, the Controller exchanges keys for all IoT devices in the local network (Repeat steps 3 through 6).

## Experiment results

In this section, we demonstrate the operation of the innovative protocol (appears in section 3) we developed in this research with the new approaches to dealing with the vulnerabilities of the previous protocol. We performed experiments in our laboratory with 3 Raspberry Pi 3 Model devices (#1: 10.0.0.26, #2: 10.0.0.10, #3: 10.0.0.5).

**Step 1: Preliminary step** as described in **section 4, step 1.**

Updating a local certificate for each IoT device by the network manager**,** as follows (Figure 7).

**Step 2: Determine who the Controller device is** as described in **section 4, step 2.**

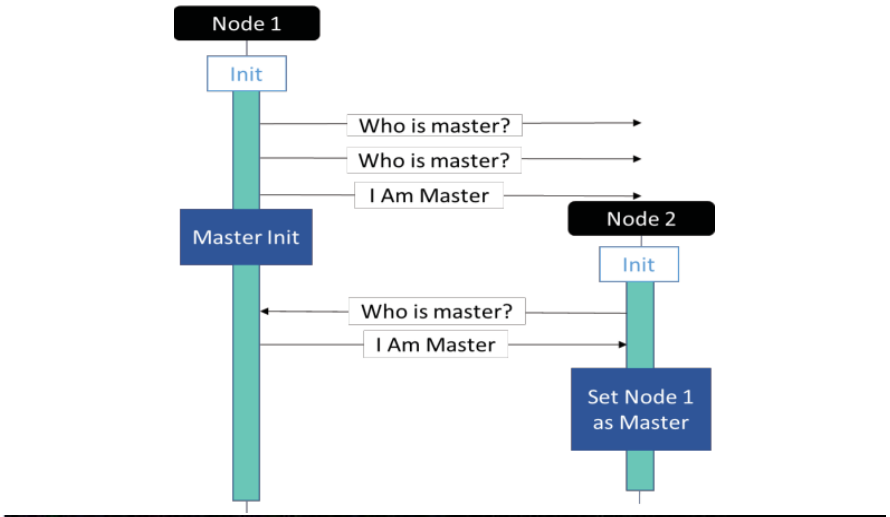
This operation is done by device #1, which looks for the Controller device on the network, sends a message in broadcast with its certificate to other devices, and declares itself a Controller. The other instruments would ignore this message if the device were without a certificate (Figure 8).

**Step 3:** The Controller defines the key-pool size as described in **section 4, step 3.**

ISSN 2995-8067 DOI: 10.61927/igmin149

```python
def create_certificate():
    file_exist = os.path.exists('../encrypted_files/certificate.crt')
    if not file_exist:
        path = '../encrypted_files'
        os.mkdir(path)
        with open('../encrypted_files/certificate.crt', 'wb') as wf:
            wf.write(certificate.public_bytes(
                encoding=serialization.Encoding.PEM,
            ))
        with open('../encrypted_files/ca.key', 'wb') as wf:
            wf.write(private_key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.TraditionalOpenSSL,
                encryption_algorithm=serialization.BestAvailableEncryption(b"openstack-ansible")
            ))
        print('Certificate successfully created')
    else:
        print('Existing Certificate')
```

**Figure 7:** Python code for updating a local certificate for each IoT device.




```
pi@raspberrypi03:~/src $ python main.py
My IP is: 10.0.0.26
**Node startup**
Creating New thread..
Looking for the Master on the network...
Trying for the #1 time...
Sending message IS_THERE_MASTER to: <broadcast>
 Created socket. Listening for messages...
Trying for the #2 time...
Sending message IS_THERE_MASTER to: <broadcast>
Sending message I_AM_MASTER to: <broadcast>
No master is found! Setting myself as master
Master is found! master ip is: 10.0.0.26
```

**Figure 8:** Python code results for determining who is the Controller device.

ISSN 2995-8067 DOI: 10.61927/igmin149

Now, Device #1 calculates the required pool of keys and the number of keys each device will receive, ensuring an overlap of at least one key between any two devices on this network with a 90% probability. In our case (3 devices), a pool of 152 and 16 keys per device is required (Figure 9).

**Step 4:** Controller requests manufacturing of distributed keys as described in **section 4, step 4.**

Device #1 **(**the Controller**)** sends a message in broadcast for all devices to generate keys (Figure 10).

**Step 5:** Create distributed keys by each node in the network **as described in section 4, step 5.**

Each node in the network generates keys as required. These keys are sent to the Controller encrypted with the Controller's public key (Figure 11).

**Step 6:** Distribution of keys **as described in section 4, step 6.**

Each device generates a private and public RSA key and sends the public key to the Controller with his certificate. The Controller (#1) can now encrypt the subset of keys and send it to the other devices (#2 and #3) (Figure 12).

**Step 7**: Finding a shared key **as described in section 4, step 7**

Each node that receives keys sends an "I AM ON THE NETWORK & myCertificate" message in the broadcast. When another node wants to exchange encrypted messages, it sends CLIENT START SESSION & myCertificate messages with its list of indexes (of keys). Here, device #2 got a message from device #1 to find a shared key (148) (Figure 13).

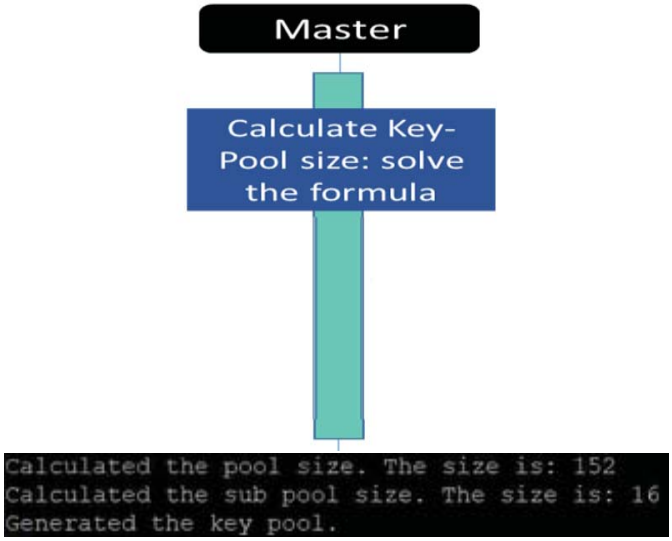



**Figure 9:** Python code results for defining the key-pool size by the Controller device.

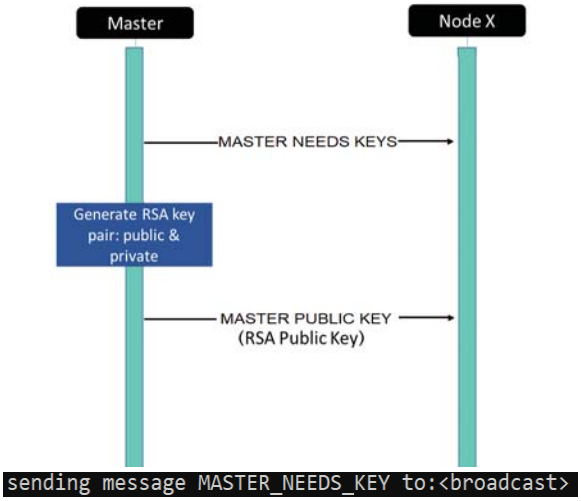



**Figure 10:** Python code results for distributing keys by the Controller device.

ISSN 2995-8067 DOI: 10.61927/igmin149

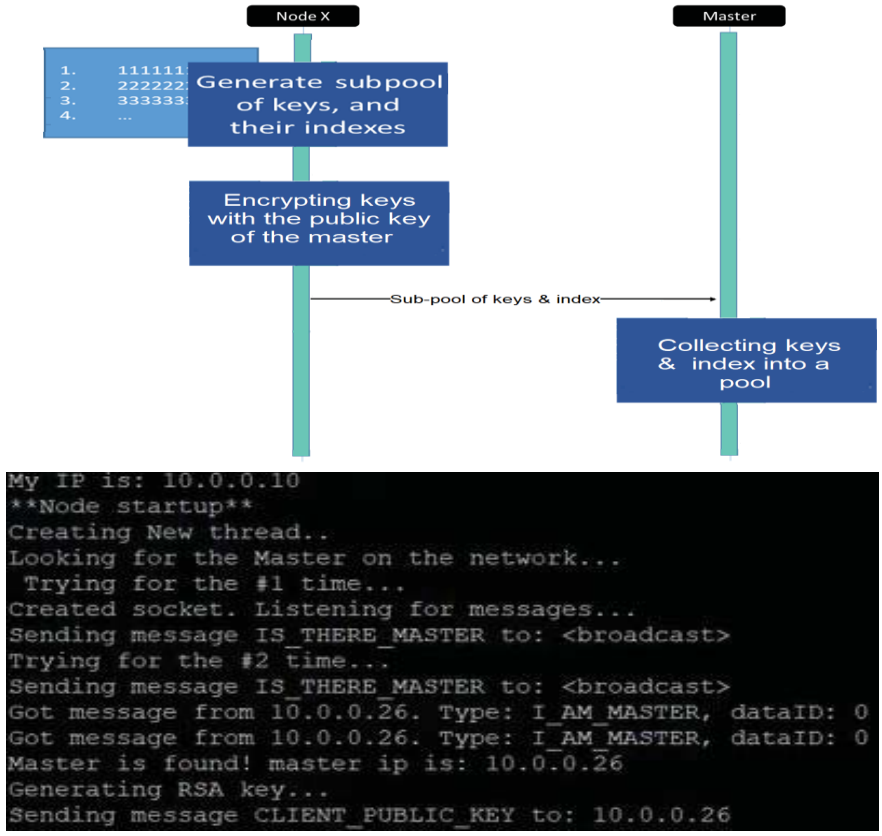



**Figure 11:** Python code results for the creation of distributed keys by each node.

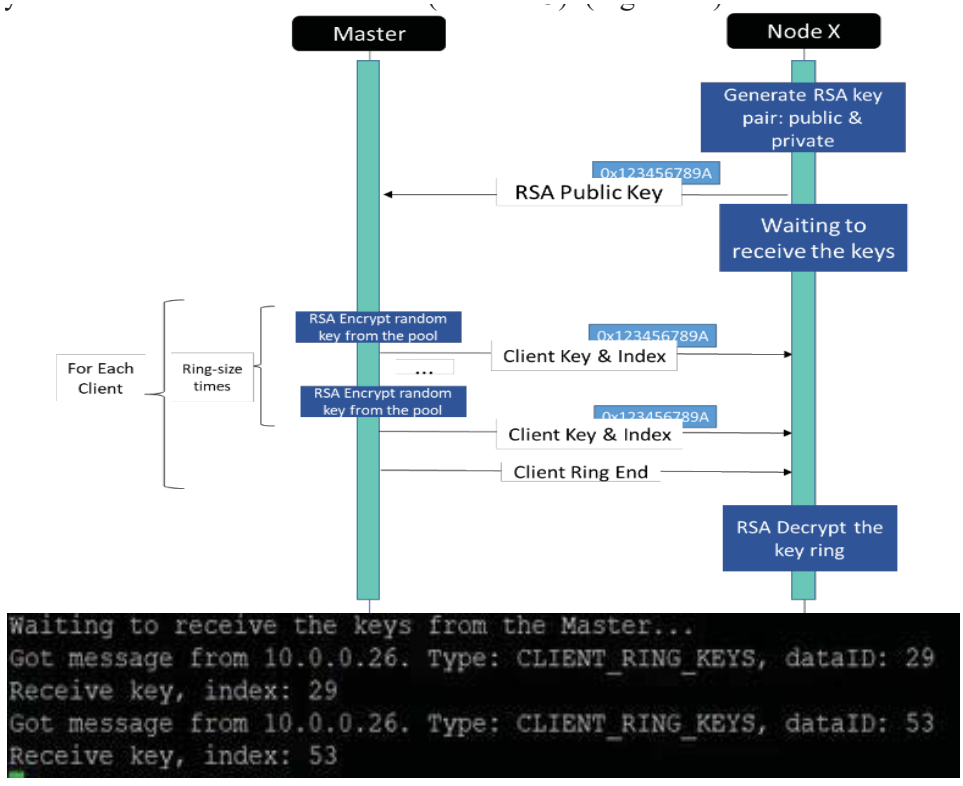



**Figure 12:** Python code results for generating a private and public RSA key.

ISSN 2995-8067 DOI: 10.61927/igmin149

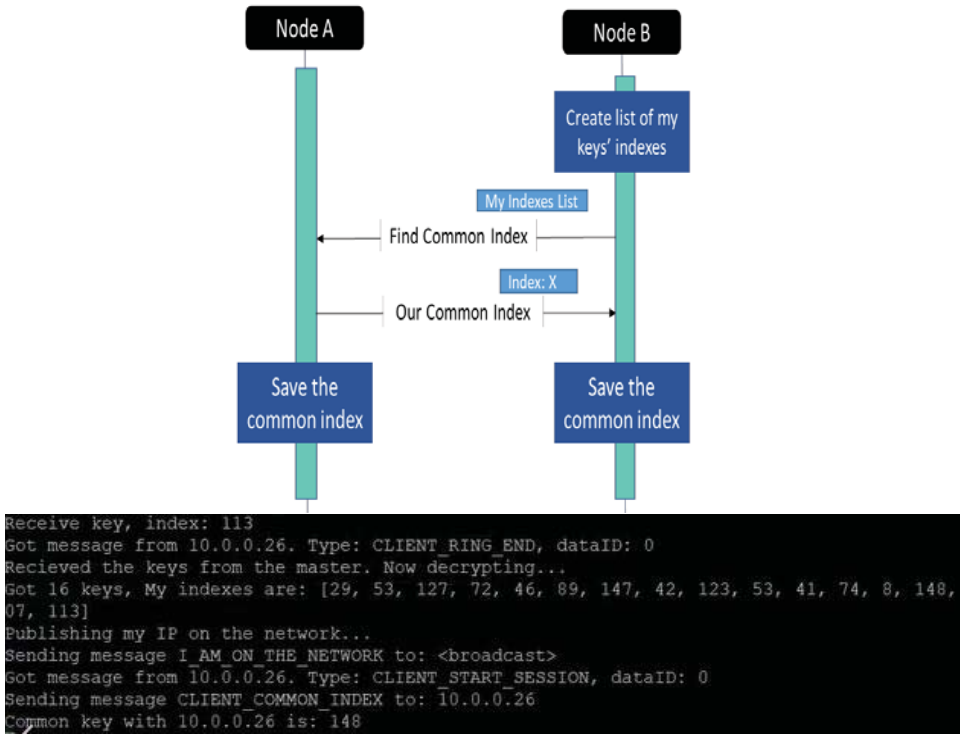



**Figure 13:** Python code results for finding a shared key.

**Step 8:** Secure network as described in **section 4, step 8**

Here, node #2 wants to communicate with node #3 (none of them is the Controller), and the shared key between them is **42**. node #2 sends a MESSAGE ENC DATA message in which AES encrypts the information it wants to transmit, and the encryption key is the shared key. Node #3 decrypts with the same key (Figure 14).

**Step 9:** Detection of missing devices as described in **section 4, step 8**

To ensure that all devices in the network are alive, the Controller (#1) sends a ping message every 20 minutes to each node (e.g., #3). If there is no answer from a particular device, the current batch of keys is canceled and immediately replaced (Figure 15).

## Conclusion and future work

In conclusion, our endeavors to enhance the security of IoT networks have led to significant strides, building upon the foundation laid in our previous work as mentioned previously. The initial protocol addressed key construction and distribution challenges for IoT devices with limited resources. As a complementary extension, this paper introduces a fortified protocol designed to withstand potential cyberattacks.

### Key upgrades

1) **Device-specific certificates:** We implemented unique certificates for each IoT device, enhancing trust and preventing unauthorized access to the network during the initial installation phase.

2) **Robust key dictionary security:** Recognizing the vulnerability of the key dictionary, we implemented stringent security measures, restricting unauthorized access and fortifying the encryption keys against potential breaches.

3) **Swift key replacement mechanism:** In response to the disappearance of an IoT device, we introduced a mechanism for immediate key replacement. This proactive measure ensures continuous network security, mitigating the risk of compromised keys.

### Future work

While the enhancements show promise in controlled laboratory conditions with three devices, future work involves expanding the scope of testing to a larger scale. Real-world deployment and evaluation will be pivotal in understanding the practicality and effectiveness of the enhanced protocol in diverse IoT environments. The following aspects will be explored in future work:

1) **Large-scale testing:** Rigorous testing on a larger scale will provide insights into the protocol's performance and resilience under dynamic and extensive IoT deployments.

2) **Real-world deployment:** Practical implementation of the enhanced protocol in real-world scenarios will be crucial in validating its effectiveness and identifying any unforeseen challenges.

ISSN 2995-8067 DOI: 10.61927/igmin149

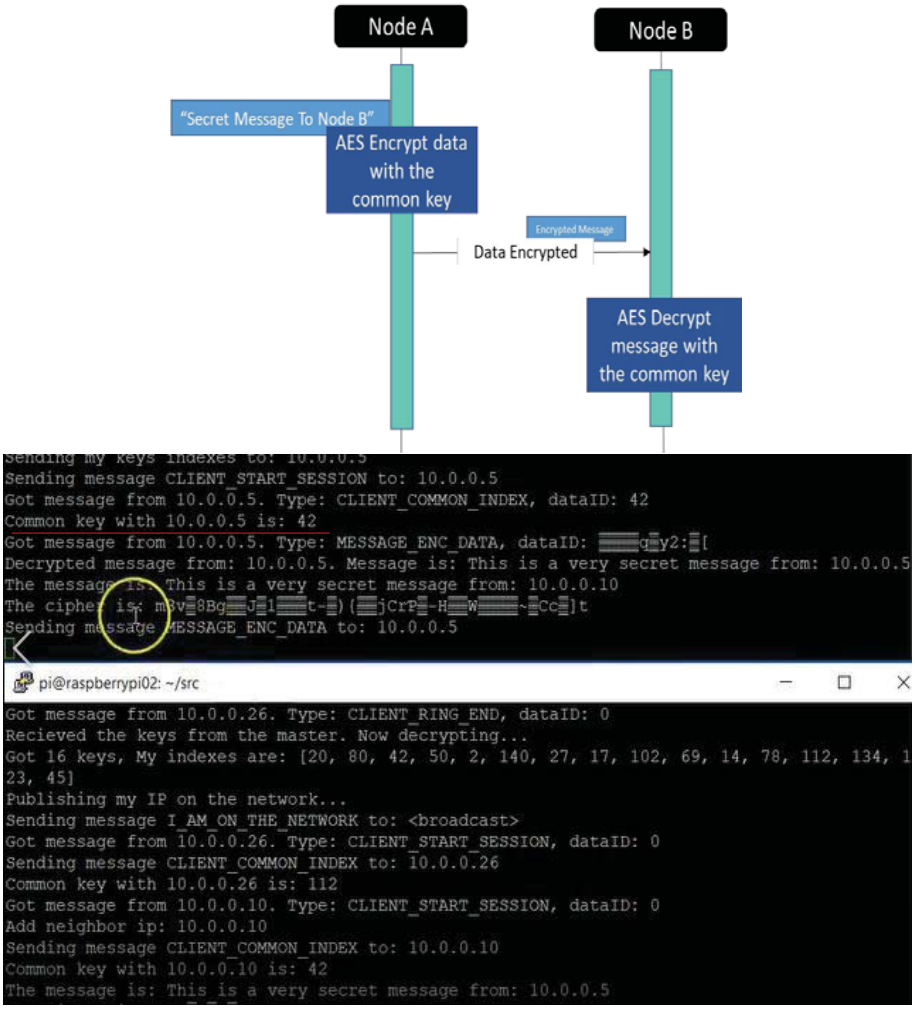



**Figure 14:** Python code results for communication between node #2 and node #3.

```
ping 10.0.0.5
Pinging 10.0.0.5 with 32 bytes of data:
Reply from 10.0.0.5: bytes=32 time=60ms TTL=116
Reply from 10.0.0.5: bytes=32 time=61ms TTL=116
Reply from 10.0.0.5: bytes=32 time=61ms TTL=116
Reply from 10.0.0.5: bytes=32 time=61ms TTL=116

Ping statistics for 10.0.0.5:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 60ms, Maximum = 61ms, Average = 60ms
```

**Figure 15:** Python code results for detection of missing devices.

3) **Dynamic IoT environments:** Future work will consider the adaptability of the protocol to various IoT environments, ensuring its effectiveness in diverse and dynamic settings.

4) **Security audits:** Continuous security audits and assessments will be conducted to identify and address any emerging threats or vulnerabilities in the evolving landscape of IoT security.

In summary, our enhanced protocol represents a crucial step towards creating a robust and secure IoT ecosystem. Future endeavors will focus on translating these advancements into practical solutions, addressing the ever-evolving challenges in IoT security.

## References

1. Leshem G, David E, Domb M. Probability-Based Keys Sharing for IoT Security.

ICSEE International Conference on the Science of Electrical Engineering. 2018.

2. Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Gener. Comput. Syst. 2013; 29:1645–1660.

3. Sundmaeker H, Guillemin P, Friess P, Woelfflé S. Vision and challenges for realizing the Internet of Things. In Cluster of European Research Projects on the Internet of Things; European Commission: Brussels, Belgium. 2010; 3: 34–36.

4. Stellios I, Kotzanikolaou P, Psarakis M, Alcaraz C, Lopez J. A survey of IoT-enabled cyberattacks: Assessing attack paths to critical infrastructures and services. IEEE Commun. Surv. Tutor. 2018; 20:3453–3495.

5. Eschenauer L, Gligor VD. A key-management scheme for distributed sensor networks. Proceedings of the 9th ACM conference on Computer and communications security, Washington DC.11-2002; 341-47.

6. Alagheband MR, Aref MR. Dynamic and secure key management model for hierarchical heterogeneous sensor networks. Iet Information Security [IF: 1.04]. DOI — 10.1049/iet-ifs.2012.0144, 2012

7. Sciancalepore S, Piro G, Boggia G, Bianchi G. Key Management Protocol with Implicit Certificates for IoT systems. Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems. Florence, Italy. ACM, NY, USA. 2015; 37-42. ISBN: 978-1-4503-3502-7

8. Roman R, Alcaraz C, Lopez J. Key management systems for sensor networks in the context of the Internet of Things. Nicolas Sklavos, Computers & Electrical Engineering. 2011; 37:2; Pages 147-159.

9. Wazid M, Das AK, Odelu V. Design of Secure User Authenticated Key Management Protocol for Generic IoT Networks. IEEE Internet of Things Journal. 2018; 5:1; 269-282: ISSN: 2327-4662

10. Benslimane Y, BenAhmed K. Efficient End-to-End Secure Key Management Protocol for Internet of Things. International Journal of Electrical and Computer Engineering (IJECE). 2017; 7:6; 3622 3631 ISSN: 2088-8708.

11. Mahmood Z, Ning H, Ghafoor A. A Polynomial Subset-Based Efficient Multiparty Key Management System for Lightweight Device Networks. Sensors. 2017; 17(4): 670. doi:10.3390/s17040670

12. Mohammad M. Internet of Things: A Comprehensive Overview on Protocols, Architectures, Technologies, Simulation Tools, and Future Directions. Energies. 2023; 16.8:3465.

13. Gerodimos A, Maglaras L, Ferrag MA, Ayres N, Kantzavelou I. IoT: Communication protocols and security threats. Internet Things Cyber-Phys. Syst. 2023; 3: 1–13.

14. Domínguez-Bolaño T, Campos O, Barral V, Escudero CJ, García-Naya JA. An overview of IoT architectures, technologies, and existing open-source projects. Internet Things. 2022; 20:100626.

15. Python Own Certificate Authority (ownca). https://packagegalaxy.com/python/ownca

**How to cite this article:** Leshem G, Domb M. Strengthening IoT Network Protocols: A Model Resilient Against Cyber Attacks. IgMin Res. Feb 12, 2024; 2(2): 084-096. IgMin ID: igmin149; DOI: 10.61927/igmin149; Available at: www.igminresearch.com/articles/pdf/igmin149.pdf