I J C T A, 9(20), 2016, pp. 43-49
© International Science Press

End-to-End Encryption Scheme for IoT Devices Using Two Cryptographic Symmetric Keys

Sheik Al Farhan* and Kavitha C. R.**

ABSTRACT

With growing interest and research towards building a connected world using IoT devices, security of these devices becomes a great concern. There are various methods available today to ensure secure exchange of information from and to these devices, most commonly used methods today ensure security between the device and the server it communicates with. The server usually can control these devices, but if the server itself is compromised, the safety of these devices is compromised as well. There can be instances where even the control of a server on certain devices can be of great threat. This is true for the devices which control opening and closing of doors, stoves or even electricity. Such devices must only be operated on the user's instructions and their security should not be governed by server security. In this work, we propose an encryption scheme to provide complete control to the owner of the device, which makes use of two cryptographic symmetric keys. These two keys are used to encrypt information exchange of device-server and user-device respectively.

Keywords: Cryptography, Cyber Security, Encryption, Internet of Things, Raspberry Pi, Symmetric Keys

1. INTRODUCTION

Internet of Things (IoT) refer broadly to the extension of network connectivity and computing capability to objects, devices, sensors, and items not ordinarily considered to be computers [3]. Most electrical or electronic devices like televisions, ceiling fans, lights, any kind of sensor etc. can be controlled remotely.

Type of communicated information is governed by the kind of electronic device and may include the current state of the device (eg: it's on or off, full or empty, charged) or data from sensors like a temperature sensor, location trackers, heart beat monitor, etc. The use cases for IoT are infinite in number, for example, a university can use IoT to track all its assets in the campus, a logistics vendor can use it to track shipments or a person can control electronic devices at her home.

IoT has multiple communication models [3], one such model is device-to-cloud communication. Such a communication can be implemented using standard protocols like Messaging Queue Telemetry Transport (MQTT) [1], Extensible Messaging and Presence Protocol (XMPP) [4], Data Distribution Service (DDS) [5] or Hypertext Transfer Protocol (HTTP).

Security concerns like confidentiality, integrity and authenticity of information exchanged needs to be addressed while developing an IoT ecosystem. Publish/subscribe model [6], which is commonly used for device-to-cloud communication addresses these by exchanging information over SSL (Secure Sockets Layer) and encrypting the information end-to-end using shared tokens. This ensures that only server and device can communicate using tokens, information is encrypted end-to-end, is transported confidentially, maintaining

* Department of Computer Science and Engineering Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Amrita University Bengaluru, Karnataka, India, *Email: mail@farhansheik.com*

** Department of Computer Science and Engineering Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Amrita University Bengaluru, Karnataka, India, *Email: cr_kavitha@blr.amrita.edu*

the integrity and validating the authenticity, an attack on this network is only possible when the tokens are somehow compromised which could be due to a flaw in the algorithm.

Consider a scenario in which device management server is compromised, the attacker can then simply control all the devices around the world and create chaos, the extent of damage is not much if the devices were simply sensors which relay data, but what if the devices respond to instructions like for instance opening of doors, changing temperature of heaters. In such a situation, we would only desire the user to have control on the devices or a high trust is required from the service provider offering the server. To avoid such an extent of damage, the server too must not be able to govern these devices without user's discretion.

For transferring complete control of device only to user, in our work, we propose to use two cryptographic symmetric keys. This would ensure device-user security, where in, one key is known to server and device and another key is known only to user and the device. There could be some devices which don't have to be controlled by the user, like for instance a temperature sensor, which simply relays temperature data to server, in such cases we wouldn't need to use the user-device key, only when a function is to be performed by a device upon user instruction the user-device key will be used.

No copy of user-device key is stored on server and hence, even if the server was compromised the extent of damage is low.

To demonstrate our work, we have used AES (Advanced Encryption Standard) [2] for symmetric key encryption on device and server. A web-based client is used to manage the device; a raspberry pi is used to control a power socket with the help of a relay switch.

The rest of the paper is organized as follows. Different components and their function involved in implementation are explained in section II. Implementation details are presented in section III. Concluding remarks are given in section IV.

2. DIVING DEEPER

In this section, we describe different components and their function involved in implementation of our proposed approach.

2.1. Service Provider

The service provider could be a company offering IoT devices, the device management service and the client software to the users. The service provider is responsible for deploying and managing the whole architecture. However, the devices could be manufactured by either the service provider or probably any other company. The service provider embeds a unique device identifier onto the device in order to recognize it while the communication happens. Server-device key is stored on to the device and as well as in the server's database along with the corresponding device identifier. This key is necessary for communication between the device and the server. The service provider also enables a provision for the user to set her user-device key, which the user will require when she wishes to instruct her devices.

2.2. Server

The server is a very critical component among all components involved in the proposed technique, there could be more than one server deployed for load balancing purposes. These servers can even be spread across different continents. It all depends on how a service provider would want to set up this architecture.

For simplicity, we would refer to a cluster of servers as a server. Primary function of a server includes device presence management, validating and storing information that is being relayed by the devices, route

encrypted information from a client to desired device, manage user information and handle device access through a client.

The server determines if a device is online by the data it receives periodically. Suppose, the server does not receive data for a threshold period of time, it would assume that the device is offline.

When the server receives data from a device, the data will typically comprise of the device identifier and the encrypted data. The server then records the current IP address associated with the device. Device identifier is used to obtain the shared key (server-device key) from its database, and data is decrypted. Data is dropped if the decryption fails, this happens when the identifier is possibly wrong, or there is an attempt to send dummy data.

When the decryption is a success, the data is then parsed and required data is stored in the database. The data can be of different formats. For instance, for a simple on-off device, operations are to switch on the power supply or to cut it off. Then only the current state (on or off) information is sent to the server.

The server also maintains user information. A user can register herself using a client, the server authenticates a user and the devices that belong to her. The client shows the devices that belong to that user and the appropriate options associated with it.

2.3. Client

A client could be a mobile application or web based application, which the user would use to manage her devices. Primary function of a client includes user registration and authentication, managing devices from the client dashboard and perform client side encryption of the instructions issued to a device by the user.

The user registers herself using a client and signs in, she then has to add devices that belong to her by supplying a device identifier and user-device key. A token is requested from the server which is then encrypted using user-device key by the client, this encrypted token is then sent to the server which forwards it to the appropriate device. It should be noted that the server is never aware of the user-device key. All encrypted messages that come from a client are encrypted and are simply routed to appropriate devices by the server.

The device is expected to decrypt the received encrypted token using user-device key and send it to the server after encrypting it with server-device key. The server will validate this token and update its database, which proves that the device belongs to a particular user.

Whenever a user desires a device to perform an action, she selects an appropriate option from the dashboard. The instruction is encrypted by the client using user-device key and is sent along with device identifier to the server, which is then forwarded to the corresponding device.

2.4. IoT Device

Primary functions of an IoT device includes controlling the circuit attached with it based on the instructions being received from server, relay data periodically to the server and recover to its last known state on reboot.

The device relays its current state information to the server by encrypting it using server-device key, and it stores its current state in its memory, so that if the device happens to accidentally reboot, its state can be restored.

When a user's encrypted instruction reaches the device, the device decrypts this instruction and performs an action accordingly upon successful decryption. There is also a provision for the user to change the user-device key stored in the device.

The IoT device has a relay switch connected to one of its input/output pins. Based on the instruction, the pin has logic high or logic low, which controls the opening and closing of relay switch which acts like a regular switch to a power socket.

3. IMPLEMENTATION

In this section, we will discuss about how we have implemented our technique, we have used a Raspberry Pi micro-controller, a relay switch, and a computer. We have hosted our device management server, and a web based application on the computer. We have implemented a simple on-off device, its basic task is to switch on or switch off an electrical power socket. Any appliance can be connected to this socket. The whole set-up looks like the following diagram.

Since raspberry pi runs a Debian based operating system called Raspbian, everything we have performed on this device is similar to how we would do on any Linux system. We start with writing a simple HTTP (Hypertext transfer protocol) server using python, we will refer to this server as device. To avoid confusions with the device management server, we will refer to device management server as server.

Every time the device boots up, it checks if device identifier and encryption keys exist, these are stored in separate files namely dev.id, server.key and secret.key. Now it checks if current.state file exists, since the only state information we need to store is 1 or 0, that is, 1 for On and 0 for Off. If this state file does not exist, that means, the device has booted up for the first time, the device then creates a new file with default value 0. If the current.state file exists, it will be read and used.

We choose a particular GPIO (General Purpose Input Output) pin on the micro-controller, the device puts this pin at logic high for current state 1 or logic low for current state 0. A wire is connected between this pin and a relay switch, Depending on current state, the relay switch turns on or off based on logic high or low in the pin, which further controls the flow of current in the wire.

The device runs a task in background to encrypt current state using AES algorithm with the server-device key and sends it periodically via a post request on /update URL end-point to our server (10.0.0.20:8080) along with its device identifier.

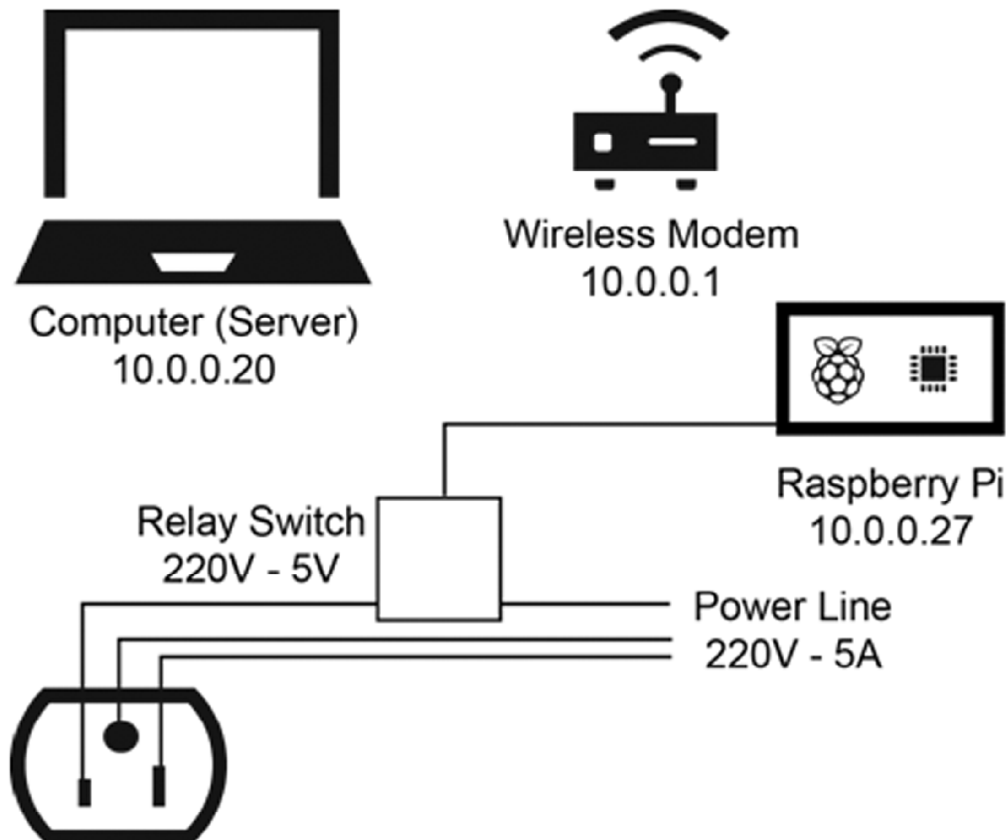

Figure 1: Architecture of proposed algorithm

Since the device has a HTTP server too, it can be accessed through its IP address and a port number, we have used port 1337 for our demonstration, we have connected a wireless adapter to the controller, and the device obtains its IP address dynamically. The device also has an authentication mechanism which we shall discuss shortly.

The device management server is running on the computer with static IP address 10.0.0.20:8080. The server has a database with tables like devices, users, user device access and authorization tokens. These tables and the usage in various scenarios are illustrated below.

Table 1
Illustrating devices schema

device_id	*ip_address*	*shared_key*	*state*	*last_seen*
cd95d	10.0.0.27	r3un1teE88	1	1461772264
7b9f6	10.0.0.25	er@winA132	0	1461771213
8f0a0	10.0.0.23	Q12onyx12Q	1	1461772269

As seen in Table -1, The devices schema contains various fields, the device_id field stores the identifier of the device, ip_address field stores the current IP address of the device, and is null if the device have never communicated with the server, shared_key stores the shared_key which is used by the device to encrypt all information which it sends it to the server, state field stores the current state of the device, 0 for Off and 1 for On, last_seen field stores the latest UNIX timestamp at which the device has sent its state information.

Devices table is pre-populated with the device identifier and shared key, and the same is set in the device before it is run. For every post request that comes on http://10.0.0.20:8080/update end-point, the server reads the device identifier in the request parameters and queries the database to check if the device identifier exists.

If device identifier exists, the shared key is retrieved and the data in the request parameters is decrypted using AES algorithm with the server-device key.

On success, the new state is set. Also, the IP address from which the request was received is updated. The last seen field is updated to the current timestamp. On failure, the request is ignored.

Table 2
Illustrating users schema

user_id	*username*	*pass*
1	john@xyz.com	p455w0rd
2	rony@abc.com	s3cr3t11
3	test@esl.com	1337p455

As seen in Table -2, The users schema contains the user_id field which is the unique identifier for a user, username field which stores the username of the user and pass field which stores the password of a particular user.

The user can access the web application by typing http://10.0.0.20 in any web browser and is prompted to enter her username and password, or the user can register herself. We have tried to keep the user information minimal for demonstration purposes.

Her stored credentials are then used to validate the user when she attempts to login. Upon successful authentication the user is redirected to a dashboard page, where the user can see the devices she added earlier and their current state (on or off) with a toggle switch to change the state if she desires.

The time stamp stored in database is used to determine if the device is online or offline, if device hasn't sent its data to the server for a set threshold amount of time (say 5 seconds), then device is assumed to be offline and toggle switch is disabled for that device.

Suppose the user is new, she would see an empty dashboard, with an option to add a device. When the user clicks on the add device button, she is prompted to enter a device identifier.

The device identifier is then sent to server as a post request parameter to http://10.0.0.20:8080/add_device end-point from the website, the server then generates a temporary token, stores it in authentication tokens table along with the user_id and the device identifier, and responds with the token in the response. The following Table -3 below depicts the authentication tokens schema.

Table -3
Illustrating authentication tokens schema

token	*user_id*	*device id*
3b158b17-02c4-49e3-95c8	1	cd95d
e4e6817b-e5bc-493b-bdc2	1	7b9f6
b7a97054-0036-4ae0-adaf	2	8f0a0
27d8c4fc-dd2c-4d5b-a3e3	3	8f0a0

The user is now prompted to enter her user-device key. This key is then used to encrypt the obtained token in the browser. This encrypted token along with the device identifier is sent to the server on /do_auth URL end-point. Now, the server looks up the IP address of the device using the device identifier and sends the encrypted token to the device.

If the destination device's IP is 10.0.0.27, then the encrypted token is sent to http://10.0.0.27:1337/auth. When the device receives this request, it attempts to decrypt the token using the user-device key stored in the device. On success, it encrypts this token again, but with the server-device key, and sends it to the server along with its device identifier on http://10.0.0.20:8080/auth_verify end-point.

The server on receiving this request, decrypts the token using server-device key of the corresponding device. On success, it looks up the authentication tokens table using this token. It verifies if the token and the corresponding device identifier matches, and then user device access table is updated, with the device identifier and user identifier. The user device schema is depicted in Table -4, user_id field stores the identifier of the user as mentioned in Table -2 and device_id field stores the device identifier of the device which that particular user has access to.

Table 4
Illustrating user device access schema

user_id	*device id*
1	cd95d
1	7b9f6
2	8f0a0
3	8f0a0

It must also be noted that this enables multiple users to manage the same device. And, multiple devices can be managed by the same user, which can be observed in the user device access table. A user's access to a particular device can be revoked by simply destroying an entry from the same table. This could be done when a device is probably offline for a very long time.

When a user wants to change the state of device i.e. to control the power socket. She uses the toggle switch in the dashboard for that particular device of which she wishes to change the state. Turning the toggle switch off, will cut off the power to the socket and turning it on will allow the current to flow through the socket.

Whenever the user wishes to change the state, after toggling the switch in the dashboard, the user is prompted to enter her user-device key. This key is used to encrypt the new state (1 or 0 in our case) using AES algorithm in the browser.

The encrypted command/state, along with that particular device's identifier is sent to the server on http://10.0.0.20:8080/send_command end-point.

When the server receives a request on this end-point, it forwards the encrypted command to that device's IP address on /command end-point, example, http://10.0.0.27:1337/command.

The device then decrypts this command, and updates its state, and changes its signal on the pin to high or low, depending on the new state, which controls the relay switch and hence the flow of current.

4. CONCLUSION

In this paper, we have used two symmetric keys to provide end-to-end encryption between a user, server and device, which ensures complete control of an IoT device only by the user. This type of encryption scheme will be useful in devices such as those which can control doors of a building, electric stoves, or even air-conditioning systems, where control of these devices by a third party can be hazardous. The Raspberry Pi micro-controller has been used to implement a simple on-off IoT device which controls a power socket via a relay switch. Results demonstrate that the proposed encryption scheme does not allow a server or any other person to control these devices except for the owner of the device.

REFERENCES

[1] Collina, M., Corazza, G.E., Vanelli-Coralli, A.: Introducing the qest broker: Scaling the iot by bridging mqtt and rest. In: Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on. pp. 36–41. IEEE (2012)

[2] Daemen, J., Rijmen, V.: The design of Rijndael: AES-the advanced encryption standard. Springer Science & Business Media (2013)

[3] Karen Rose, Scott Eldridge, L.C.: The internet of things: An overview (2015), http://www.internetsociety.org/sites/default/files/ISOC-IoT-Overview-20151221en.pdf

[4] Kirsche, M., Klauck, R.: Unify to bridge gaps: Bringing xmpp into the internet of things. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on. pp. 455–458. IEEE (2012)

[5] Pardo-Castellote, G.: Omg data-distribution service: Architectural overview. In: Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on. pp. 200–206. IEEE (2003)

[6] PubNub: A new approach to iot security (2015), https://www.pubnub.com/static/papers/IoT Security Whitepaper Final.pdf