2018 IEEE 15th International Conference on Mobile Ad-hoc and Sensor Systems

SecT: A Lightweight **Sec**ure **T**hing-Centered IoT Communication System

Chao Gao*, Zhen Ling‡, Biao Chen§, Xinwen Fu†, Wei Zhao¶
*University of Massachusetts Lowell, MA, USA. Email: cgao@cs.uml.edu
‡Southeast University, Nanjing, China. Email: zhenling@seu.edu.cn
§University of Macau, China. Email: bchen@umac.mo
†University of Central Florida, USA. Email: xinwenfu@cs.ucf.edu
¶American University of Sharjah, UAE. Email: wzhao@aus.edu

***Abstract*—In this paper, we propose a secure lightweight and thing-centered IoT communication system based on MQTT, *SecT*, in which a device/thing authenticates users. Compared with a server-centered IoT system in which a cloud server authenticates users, a thing-centered system preserves user privacy since the cloud server is primarily a relay between things and users and does not store or see user data in plaintext. The contributions of this work are three-fold. First, we explicitly identify critical functionalities in bootstrapping a thing and design secure pairing and binding strategies. Second, we design a strategy of end-to-end encrypted communication between users and things for the sake of user privacy and even the server cannot see the communication content in plaintext. Third, we design a strong authentication system that can defeat known device scanning attack, brute force attack and device spoofing attack against IoT. We implemented a prototype of *SecT* on a $10 Raspberry Pi Zero W and performed extensive experiments to validate its performance. The experiment results show that *SecT* is both cost-effective and practical. Although we design *SecT* for the smart home application, it can be easily extended to other IoT application domains.**

I. Introduction

The popularity of Internet of Things (IoT) has attracted the attention of hackers. On Oct. 21, 2016, a huge DDoS attack from the Mirai botnet was deployed against Dyn DNS servers and shut down a number of web services including Twitter [1]. The IoT reaper botnet was discovered in 2017 [2] and exploited newly found vulnerable IoT devices. Various attacks have been discovered against IoT devices, including attacks against IoT device hardware [3], operating system/firmware [4], application software [5] and networking protocols [1], [6], [7]. A complicated attack may exploit multiple vulnerabilities to achieve its goal. For example, Stuxnet exploits various Microsoft Windows vulnerabilities to attack specific industrial control systems such as those in Iran [4]. It can be observed an IoT device is subject to cyber/network attacks because it is connected to the Internet/network and has vulnerabilities. An IoT system is as strong as the weakest link. This paper focuses on security issues at the network protocol level.

As an intuitive IoT application domain, smart home has attracted both manufacturers and attackers recently. We want to connect various household appliances to the Internet for easy access and automated control. We see a variety of smart home products on market, including smart plugs, smart bulbs, smart surveillance/security cameras, Amazon Echo and Google Home. A smart home IoT system is often composed of three major components: device/thing, cloud server and controller (e.g., app on a smartphone). Based on who authenticates a user/controller for the use of an IoT device, we can categorize a smart home system into server-centered and thing-centered systems. In a server centered smart home system, the cloud server authenticates the user. In a thing-centered smart home system, the device/thing itself authenticates the user.

In this paper, we focus on the thing-centered smart home system. Although the server-centered smart home system dominates the market, there are thing-centered smart home products such as those from DLink, Xiongmai and Edimax. The major advantage of a thing-centered smart home system is it preserves user privacy since the server is mainly a relay between a user and thing in a thing-centered system. Recall a smart home system is often behind network address translation (NAT) and requires a relay on the Internet so that the device builds a persistent connection to the relay server and a user can control the device from the Internet. It does not store much of user data. Another advantage is since a thing stores user credentials and the server is merely a relay, it avoids the single point of security failure in case that a server is compromised.

The major contributions of this paper are summarized as follows.

1) We explicitly identify critical functionalities in bootstrapping a thing based on our analysis of commercial products. Pairing refers to how a user claims the ownership of a device and establishes a communication venue with the thing. Binding is defined as how a thing is connected to the Internet and the user/controller is associated with the thing over the Internet. We design secure bootstrapping strategies and establish the root of trust for things.
2) We design an end-to-end encrypted communication strategy to protect user privacy. Our system uses the lightweight IoT communication protocol, MQTT, which does not provide end-to-end encryption between a thing and user. With end-to-end encryption, even the server cannot see the plaintext communication content between

2155-6814/18/$31.00 ©2018 IEEE
DOI 10.1109/MASS.2018.00018

the thing and user.

3) We design a strong authentication system that can defeat known device scanning attack, brute force attack and device spoofing attack. Secure bootstrapping serves as the root of trust in our thing-centered IoT system. It ensures that later operation is secure.
4) We implemented a prototype of the proposed system as a smart plug on Raspberry Pi and performed extensive experiments to evaluate the networking performance and cryptographic operation performance. The experiments on a $10 Raspberry Pi show that our deign is efficient and practical.

The rest of this paper is organized as follows. We introduce background and problem statement in Section II. In Section III, we elaborate key techniques employed by our thing centered secure IoT communication system. We present the detailed communication protocol of our system in Section IV. Security analysis is presented in Section V. We evaluate the thing centered secure IoT system *SecT* in Section VI. Related work is presented in Section VII. We conclude the paper in Section VIII.

II. Problem Statement

In this section, we first introduce our previous research [6], [7], which analyzes thing centered smart home systems. We then summarize security issues of those systems, which often lack necessary security measures to counter local and cyber attacks. In a local attack, the adversary will be in the proximity of a victim home. In a cyber attack, the adversary is on the Internet and attacks remotely. In this section, we briefly introduce such systems and identify the security issues.

A. Background

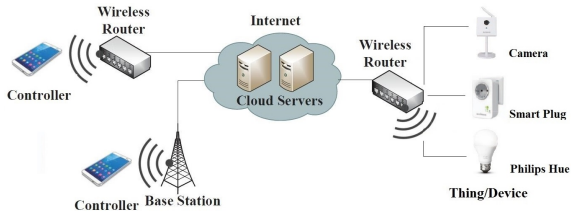


Fig. 1. A typical home automation loT system architecture

Figure 1 shows a typical home automation loT system architecture, which normally has three basic components: thing, controller, and server. 1. A thing can refer to a wide variety of devices connected to the Internet, such as IP cameras, smart plugs and smart lights. We will use the term *thing* and *device* interchangeably in this paper. 2. The controller is usually an application on a PC, smartphone or tablet of a user. We will use the term *controller* and *user* interchangeably in this paper. 3. A server is often hosted in a cloud and connects things and controllers together.

For home automation, a thing is often behind a wireless router, which adopts NAT and is the key component forming a local network of home systems. An IoT system often implements two suites of protocols: local communication protocol and remote communication protocol. With the local communication protocol, things and controllers communicate through the router and may not need the public Internet. With the remote communication protocol, controllers communicate with things using the server as a relay since things are behind routers with NAT and normally cannot communicate with things. In this case, things often build a persistent connection with the server.

We carefully studied Edimaxs smart plugs (over $45 on Amazon) and IP cameras (over $60 on Amazon) [6], [7]. Without loss of generality, this paper uses the Edimax smart plug system as an example in our discussion. The Edimax smart plug system is a thing centered IoT system where a thing authenticates a controller and its architecture is similar to the one in Figure 1. Now we introduce the *bootstrapping* process of the Edimax smart plug system. When the Edimax device is used for the first time, it works as an *open* AP, the controller connects to the open AP and configures the device so that the it can connect to the Internet. We denote the process that the controller establishes a communication venue with the device as *pairing*. When the Edimax smart plug controller pairs with the device, the communication venue is the open AP. After pairing, we have the *binding* process, i.e. how the controller will be associated with the device over the Internet. In the case of Edimax smart plugs, the controller obtains the MAC address of the device. The device then registers itself to a cloud server, denoted as the registration server. To control the device, the controller sends the authentication information with the device's MAC address to the registration server. The MAC address is used to associate/bind the controller with the device. The registration server then relays the authentication credential to the device for authentication. Once authentication is passed, a token is created and distributed to both the device and controller. Another cloud server, denoted as the command relay server, is used to relay the control command between the controller and device.

B. Issues

We reverse engineered the communication protocols of the Edimax smart plug and IP camera system and identified a few critical security vulnerabilities.

- There is no encryption of communication traffic. So it is easy to perform protocol analysis although some obfuscation strategies are used.
- In the pairing process, the open AP on the device may be abused. The risk of such practice may be small in a private setting like a home. However, for deployment in a public environment, anybody with access to the devices can reconfigure the system and may break into the system. The adversary could sniff the unencrypted traffic and get the home wireless router's passcode sent from the controller app to the device.
- In the binding process, the device's MAC address is used. However, the MAC address is predictable. This allows the

attacker to perform the device scanning attack to find the status of all smart plugs.

- In terms of authentication, the user name is fixed as "admin". The device does not limit the password attempts by a controller so that an attacker can infer the user's password through brute force attacks. Since only one way authentication of the controller by the device is performed, the attacker can launch the device spoofing attack, which registers a spoofed plug on the cloud server and waits for a user's authentication credential being sent from the controller. Using the credential, the attacker can completely control victim smart plugs.

The goal of this research is to address these issues for a thing-centered IoT system.

III. Secure Thing-centered IoT System Framework

In this section, we present our thing centered secure IoT system design that addresses security issues discussed in Section II. Our system uses the lightweight IoT communication protocol MQTT [8]. Therefore, we will first present the security features of MQTT and introduce end-to-end encryption for MQTT that does not have this feature. We then introduce secure pairing that allows only the owner of a thing to communicate with the thing. We present secure binding that connects the thing to the Internet and safely associate the user/controller with the thing over the Internet. At last, we discuss authentication between things (devices), controllers (users) and the server (broker) in an IoT environment in order to protect these entities.

A. MQTT and End-to-end Encryption

MQTT is a messaging broker system and uses a publish/subscribe protocol based on a "hub and spoke" model. The hub is the server/broker and clients are the spokes. Clients communicate with each other through the hub (broker/server) using messages. A topic is a namespace for messages on the broker. The message is relayed by the broker between clients. A client can subscribe to topics and publish messages to topics simultaneously. Clients do not need to initialize a topic before subscribing and publishing, and the broker will process the request automatically. MQTT supports mutual authentication through SSL/TLS and the pre-shared-key based encryption *between a client and the server*. It also supports the username/password authentication while the username/password authentication needs link encryption provided by SSL/TLS.

We adopt MQTT since it is lightweight and its communication overhead is low. An MQTT packet can be only 2 bytes. It is used by many IoT platforms including Amazon AWS IoT, Intel IoT, Microsoft Azure IoT and Google IoT platforms.

However, MQTT does not provide the feature of end-to-end encryption, which is necessary in case that clients may not want to disclose its communication content to the cloud server or we should provide such option to users. Recall that the MQTT server forwards messages between clients such as a controller/user and a device. We want the end-to-end encrypted communication between a user and a device so that the server does not know the content for the purpose of user privacy. The challenge is how the user and device perform key exchange. We can use the pre-shared-key (PSK) scheme. That is, at bootstrapping, the user and the device establish the pre-shared master key that will be used later to create session keys encrypting their communication. The pre-shared master key shall be updated after a period of time for freshness. If the user gets a certificate from the device, then the authenticated Diffie-Hellman key exchange can be used. That is, during the key exchange process, the user and device sign the messages they send out to the other party. Once the key exchange is completed, the device and the user will get the negotiated session key for encrypting MQTT messages. The cloud server forwards encrypted messages between users and devices. As a result, even if an attacker compromises the server, she cannot get the message content without the encryption key.

B. Secure Pairing

When the thing is used for the first time, a controller should be able to communicate with it at the bootstrapping time. It is a common practice in industry that an IoT device is open for any user to pair with the device and configure it. For example, an IoT device works as an open access point (AP) that a user can connect to and perform configurations on. However, anybody can sniff and connect to such device. We have to address how a user can claim the ownership of the device and securely connect to and configure the device. Our philosophy of the ownership is that the device will display a secret message on screen, and whoever sees the message is the owner. Instead of working as an open AP, the IoT device can use Wi-Fi Protected Access II (WPA2) and display a onetime passcode on an LCD. The user can use this passcode to connect to the IoT device.

Once a user is done with pairing and configuration, the IoT device cannot be reset or get into the AP mode displaying a onetime password unless the owner explicitly unpairs herself (i.e., deletes her account stored within the device) and the device. Once the user relinquishes her ownership, the device is reset to the factory setting and another user can take the ownership, pair with the IoT device, configure and use it. Only the owner can work as an administrator and authorize other users to operate the IoT device. In this case, we can guarantee that at the bootstrapping time, only the owner of the device can pair her controller with the device and exchange information during the next binding process. An attacker cannot connect to the device in this process and obtain the device information in the binding process by sniffing.

C. Secure Binding

Secure binding is another process of bootstrapping so that the server can associate a controller/user with a device and relay messages correctly between them on the Internet. Our secure binding process has two major functions: 1. The controller should bind the thing to the Internet, for example, home WiFi. The controller can require a user to input the WiFi SSID (Service Set Identifier) and passcode of a wireless router and send the information to the thing, which can then

connect to the Internet. Because of secure pairing, binding is secure. For example, the home WiFi passcode will not be subject to sniffing. 2. The thing generates a random MQTT topic and gives it to the controller. This random MQTT topic is unpredictable by a third party and will be used for later communication over the Internet between the controller and thing, which subscribe and publish to this random topic. Therefore, even if the server serves a large number of controllers and things, other controllers and things which may be malicious cannot subscribe and publish to random topics that are not assigned to them.

D. Authentication

Table I lists all types of authentication that we should address between things (devices), controllers (users) and the server (broker) in an IoT environment. We use a MQTT server in this paper. Table I also lists the potential authentication strategies. Recall we are developing a thing-centered authentication system while considering security of all components in the IoT system.

- A thing authenticates a controller for the purpose of device operation through password or certificate authentication. 1. The thing can act as a CA (certificate authority). The controller generates a public key pair (public key, private key) and the thing signs the certificate for the controller. The controller stores its certificate and the thing's certificate for the purpose of authentication. We can see that in this case, each thing/device performs as a CA and generates a certificate for itself and its users. 2. The thing can also generate a username/password pair and give it to the controller for later authentication.
- A thing authenticates the server for its genuineness. The thing can perform a SSL/TLS certificate based authentication of the server.
- The controller authenticates the server for its genuineness. A controller can perform a SSL/TLS certificate based authentication of the server.
- The controller authenticates the thing for the use of resources on the thing (if there is such need) through password or certificate authentication. This case is rare in the current IoT application domains although it is possible that a device may want to access the resources at the controller.
- The server authenticates a thing for the use of the server through certificate authentication. The purpose is to prevent the abuse of the server by people other than those who own products from the specific manufacturers and its partners. It will also defeat the device spoofing attack discussed in Section II. The manufacturers have to perform as a CA and generate a certificate for each of its devices. The server stores the manufacturer's certificate and certificates of all IoT devices. To authenticate a thing, the server obtains the thing's certificate, checks if it is in its database and then perform challenge and response protocol in order to check if the thing has the correct private key.
- The server authenticates the controller for the use of the server through security tokens. Recall that in the binding process, the thing creates a random topic for the thing and controller communicating with each other. We treat this random topic as a security token. Users who know the security token can use the server, which implements topic restriction through an access control list (ACL).

TABLE I
AUTHENTICATION BETWEEN THINGS, CONTROLLERS AND SERVER. SECURITY TOKENS HAVE TO BE UPDATED REGULARLY.

$\mapsto$	***Thing***	***Controller***	***Server***
Thing	ø	Password/Certificate	Certificate
Controller	Password/Certificate	ø	Certificate
Server	Certificate	Token	ø

IV. COMMUNICATION PROTOCOL

In this section, we first introduce the architecture of SecT, a lightweight **sec**ure **t**hing-centered IoT communication system and then present the communication protocol.

A. Architecture of the System

Figure 2 shows the architecture of SecT, which has four components, including controller, cloud server, thing/device (a smart plug as an example), and local server which is hosted on the thing. The thing connects to the Internet through WiFi (while Ethernet is an option too). The controller is an app on a PC, smartphone or tablet. Both cloud server and local server are MQTT servers. The core functionality of the cloud server is to relay messages between controllers and things. It also authenticates controllers and things to prevent abuse of the server. If the device and controller are in the same local network, the controller can communicate with the device locally and control it through the local server on the thing. Otherwise, they can communication over the Internet through the cloud server. Recall the cloud server is needed since a thing is often behind NAT and a thing cannot communicate with a controller directly over the Internet.

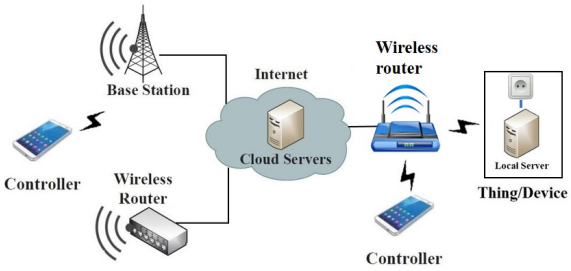


Fig. 2. System Architecture

B. Communication Protocol

1) Bootstrapping phase: When a thing is used for the first time, a controller pairs with the thing through WiFi on the thing, communicates with the thing through the local MQTT

server, connects the thing to the Internet and binds with an authenticated user. After the paring and binding phases, the controller and the thing negotiate three authentication credentials: 1. a secret MQTT topic binding the thing and controller. 2. a username/password pair for a thing authenticating a controller, 3. a master encryption key for the end-to-end encryption between the thing and controller. These three authentication credentials should be saved in a configuration file locally in the device.

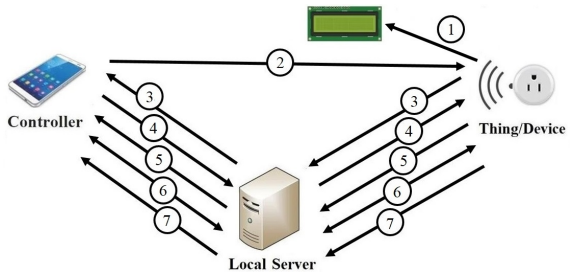


Fig. 3. Paring and Binding Phase

The detailed procedure shown in Figure 3 is introduced as follows. **STEP 1:** When the device powers up for the first time, it works as a WPA2 access point (AP). We call it the *Thing-AP* to differentiate it from a home wireless router. *Thing-AP* displays a onetime WPA2 passcode on an LCD screen. Meanwhile, the device starts a local MQTT server ready to communicate with the controller.

STEP 2: The controller connects to *Thing-AP* using the displayed onetime passcode. Recall our philosophy is who sees this passcode is the owner of the thing. At this time, secure pairing is completed. The secure pairing process consists of STEP 1 and STEP 2.

STEP 3: The thing generates a random MQTT topic denoted as T_{token}, which is used to bind the thing and controller at the cloud server, and sends it to the controller using a known topic $T_{local,default}$ through the local MQTT server over the secured WiFi. T_{token} is a randomly generated 25-character string from 10 digits, 26 upper-case and 26 lower-case alphabetic letters. It has $25 * log_2(62) = 148.85$ bits of entropy and is strong enough to defeat brute force attacks [9].

STEP 4: The controller provides the home wireless router's SSID and passcode to the device through *Thing-AP*. By far secure binding is completed. The secure binding includes STEP 3 and STEP 4.

STEP 5: The thing also generates and provides a pair of username/password for later authentication of the controller by the IoT device. We call it the *Ctlr-U/P*. Recall that the thing/device and controller can also use the certificate based authentication in which the device signs a certificate for the controller. STEP 5 sets up the authentication between the controller and thing.

STEP 6: The controller and the device negotiate a master encryption key that is used later to encrypt later communication. If certificate based authentication is used for the thing authenticating the controller, the thing and controller can also perform key exchange later. STEP 6 sets up the end-to-end encryption.

STEP 7: We also want to encrypt the communication link between the MQTT server and controller/thing. For remote communication over the Internet, we use SSL/TLS. For communication in a local network, we use the pre-shared-key (PSK) supported by MQTT. Therefore, the last step of bootstrapping phase is the thing creates a PSK and gives it to the local MQTT server and controller. The pre-shared key scheme is used between the local MQTT server and controller/thing. Now for local use, even the local (home) WiFi is open, communication in our IoT system is encrypted.

2) Registration phase: After bootstrapping, the thing will reboot. It connects to the home WiFi router using the WiFi credentials obtained in STEP 4. It will register the random topic T_{token} to the cloud server, which puts T_{token} into its topic access control list (ACL). The detailed procedure shown in Figure 4 is introduced as follows.

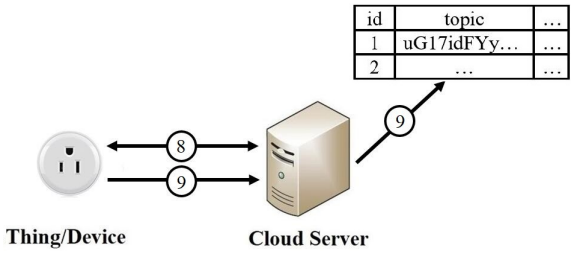


Fig. 4. Registration Phase

STEP 8: Once the thing has access to the Internet, it builds a SSL/TLS connection with the cloud MQTT server. Here the server and thing will perform mutual authentication. The thing authenticates the server through the server's certificate. The server can authenticate the thing through the thing's certificate or pre-defined username/password. Here we assume that the cloud server knows the thing's certificate or pre-defined username/password, which should be configured by the manufacturer.

STEP 9: After authentication, the thing communicates with the server through a known MQTT topic $T_{cloud,default}$ and registers the random topic T_{token} into the cloud server's topic access control list. From now on, a specific thing and its controllers/users can use the negotiated random topic T_{token} to communicate with each other.

3) Thing Discovery Phase: We now introduce how the controller finds and communicates with the thing. After bootstrapping, the controller app gets into the thing discovery phase and tries to find the device. Recall the thing runs the local MQTT server and listens for any connection request. The detailed procedure is introduced as follows.

STEP 10: The controller broadcasts a local server host IP request packet in the local network through the simple service discovery protocol (SSDP) [10]. Every time the controller tries

to control the device, it repeats STEP 10 to determine whether the controller and the device are in same local network.

STEP 11: The device responds with its IP address to the controller's SSDP request.

4) Data Communication Phase: Once the controller gets the local server IP through thing discovery, it will send two request packets to the thing for connection checking. One packet goes to the local MQTT server and another one goes to the cloud MQTT server. If authentication is successful, both local and remote MQTT servers will respond. If the device receives two identical response messages, both local and remote connections are successful. The detailed procedure is shown in Figure 5 and introduced as follows.

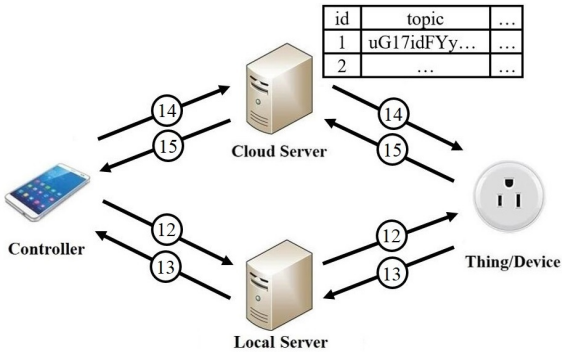


Fig. 5. Data Communication Phase

STEP 12: The controller sends a *request* message to the local MQTT server. For brevity, assume we use password authentication between the controller and thing. If authentication is successful, the local MQTT server forwards the valid request message to the thing using the negotiated random topic T_{token}.

STEP 13: The thing gets the requested message and responds back to the controller through the local MQTT server using T_{token}.

STEP 14: The controller sends a *request* message to the cloud server. After authentication, the cloud server forwards the valid request message to the thing using the negotiated random topic T_{token}.

STEP 15: The thing responds back to the controller through the cloud server using T_{token}.

Every time the controller tries to control the device, it repeats STEP 12 and STEP 13 locally or STEP 14 and STEP 15 remotely for requesting the current device status. All messages are in JSON [11] format and are encrypted with the AES-256-CBC mode [12]. There are usually three kinds of messages transmitted between the controller and thing: 1. device status request, 2. device response message, 3. control command. Note that after a period of time, the thing and controller will run STEP 6 to update the master encryption key.

5) Add/Delete User Phase: Only the owner of the device can authorize other users to operate the IoT device. The owner can send a request message to add or remove a user to the device. For adding a new user, the device will generate and provide the new user's authentication credentials to owner. After the new user connects to local MQTT server with credentials provided by owner, it initiates STEP 6 to negotiate a master encryption key. Then the device registers all new user's information to the cloud server. The detailed procedure shown in Figure 6 is introduced as follows.

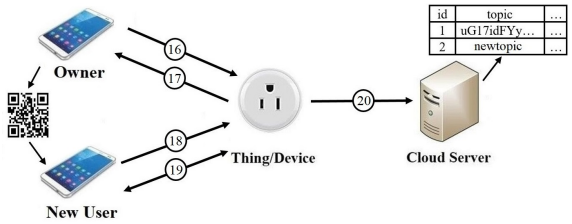


Fig. 6. Add New User

STEP 16: The owner sends a request message to the device for adding a new user.

STEP 17: The device generates a username/password pair *Ctlr-U/P* and a random topic T_{token} for the new user. Then the device generates a QR code containing these credentials and sends back to the owner.

STEP 18: The new user obtains the device information by scanning the QR code provided by the owner and connects to the device.

STEP 19: The new user repeats STEP 6 to negotiate the master encryption key with the device.

STEP 20: The device repeats STEP 8 and STEP 9 to register the new user's random topic T_{token} to the cloud server's topic ACL.

In order to remove the existing user, the owner sends a *delete* request to the device. The device deletes the user's authentication credentials and sends a *delete* request to the cloud server. The cloud server will remove the user's random topic T_{token} from its topic ACL.

V. Security Analysis

In this section, we perform the security analysis of our thing-centered IoT system.

A. Sniffing Attack

In the bootstrapping phase, the thing acts as a WPA2 protected AP. Only the owner can see the passcode displayed on the LCD of the thing. Therefore, all the bootstrapping messages are encrypted, including the negotiated username/password pair, encryption key, random MQTT topic and the home wireless router passcode sent from the controller to the thing.

In later phases including the registration phase and data communication phase, communication with the cloud server is SSL/TLS encrypted. Communication between the controller and thing is encrypted and even the cloud server cannot see the content. In the case of local communication in a local network, even if the home wireless router is not encrypted, the

communication between the controller and thing is encrypted through the local MQTT server's PSK encryption. All the negotiated keys will be updated after specific period to ensure the freshness of those keys.

In the thing discovery phase, a local attacker with access to the home WiFi router may also perform the thing discovery and obtain the thing's local IP address. However, since a local attacker does not know the pre-shared key between the controller and local MQTT server, the local attacker cannot even connect to the local MQTT server hosted on the thing.

B. Replay Attack

Here we consider an extreme case: the cloud server is compromised and may replay a captured command from the controller to the thing. To defeat such replay attack, each command message contains a timestamp, and the device will record the timestamp of the latest received valid message. When a new message arrives, the thing decrypts and compares the new timestamp with the recorded timestamp. If the new timestamp is earlier than or equal to the recorded latest timestamp, the message is invalidated and the device will ignore this command. From this discussion, we can also see the benefit of the thing-centered IoT system. Even a compromised cloud server cannot compromise the security of IoT devices.

C. Device Scanning Attack, Brute Force Attack and Device Spoofing Attack

Without the correct credentials, an attacker cannot even connect to the cloud server or local server and abuse it. The device scanning attack against Edimax smart plugs cannot be deployed against our system. The random MQTT topic T_{token} generated in the pairing and binding phase has 148.85 bits of entropy and cannot be practically predicted. The random topic will be updated after specific period. All such topics are registered in the cloud server authentication database along with the corresponding device and user information. The device and thing can only subscribe and publish to their own topics. The cloud server only accepts and forwards messages matching the topic and user information in its authentication database. It is difficult for an attacker to obtain response messages from other devices than her own, or receive control command messages from other users. Therefore, the attacker cannot predict the random topic and practically perform the device scanning attack or brute force attack since the topic is too long and random.

The device spoofing attack is a great challenge for many state-of-the-art IoT products including the Edimax smart plugs and IP cameras. This attack is infeasible against our IoT system since the attacker cannot predict the random MQTT topics of other IoT devices in our system. Even if a compromised cloud server leaks such random topic to the attacker, the attacker does not know the credentials negotiated between the controller and thing. Those credentials enable mutual authentication between controllers and things.

VI. Evaluation

In this section, we present our experiment setup and results.

A. Experiment Setup

The experiment setup is shown in Figure 7. The example system emulates a smart plug system and consists of Raspberry Pi Zero W, a 2×16 LCD screen and a power relay enabled strip, denoted as *plug*.

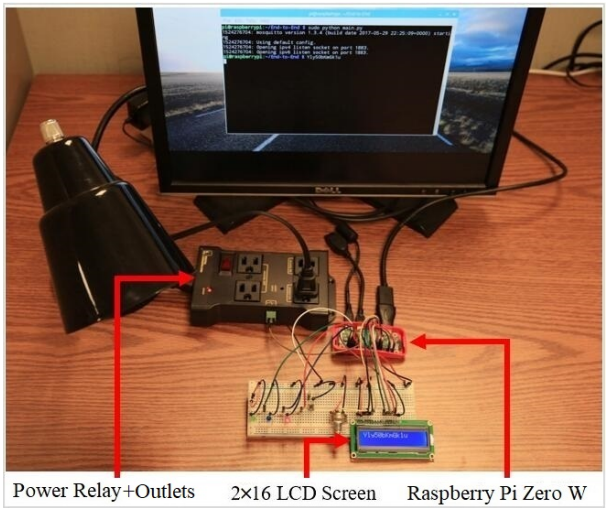


Fig. 7. Device Setup

The controller is a python program on a PC while we will develop an app for smartphones and tablets. The Raspberry Pi Zero W is installed with a full fledged Linux system and shipped with General-purpose input/output (GPIO) pins, HDMI display port, USB ports, Ethernet port and onboard WiFi (in Version 3 and Version Zero W). It allows the creation of various IoT devices such as a smart plug. We connect a power relay enabled plug to Raspberry Pi through GPIO pins. The Pi controls the plug through GPIO and turns on/off the connected plug. An LCD screen is connected to Raspberry Pi through GPIO and used for showing the onetime passcode of the WPA2 AP WiFi during the pairing process. The MQTT server is installed on the device for local communication.

Both the cloud server and local server use Mosquitto [13], which is a popular open source implementation of the popular IoT communication protocol MQTT. Mosquitto itself provides authentication strategies including password, SSL/TLS mutual authentication, pre-shared key (PSK) encryption and authentication. Mosquitto also supports authentication plugins [14]. We use MySQL as the authentication back-end on Mosquitto. The database has a user table and an ACL table. This plugin can perform authentication (user table for checking username/password) and authorization (ACL table for authorizing connection).

B. Networking Performance

Figure 8 shows the time required for sending a command controlling the device through the local server and cloud server. We run each performance test 30 times and show the average time. All communication links are SSL/TLS protected. End-to-end encryption is implemented. It can be observed that

the average runtime is tens of milliseconds for both remote and local control with Raspberry Pi 3 and Raspberry Pi Zero W. The average runtime for local and remote control with the Edimax smart plug is 40.204ms and 335.260ms respectively. Compared with the Edimax smart plug system, the runtime of a thing centered secure IoT system is feasible and acceptable.

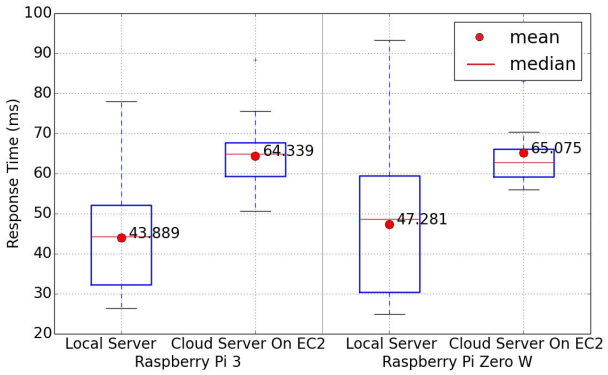
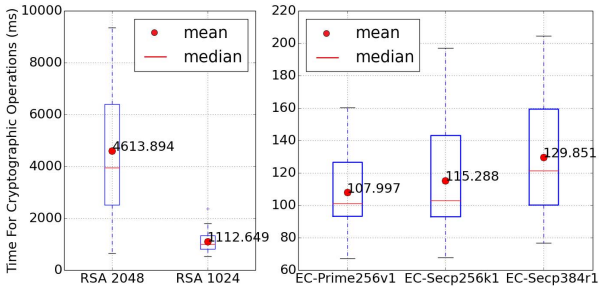


Fig. 8. Control Command Response Time

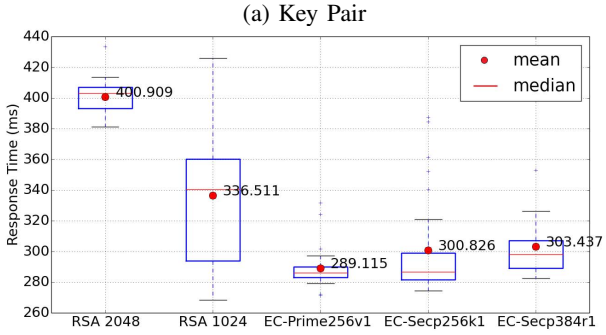
C. Cryptographic Operation Performance

Figure 9 shows the cryptographic operation performance on Raspberry Pi Zero W, which uses a system on chip (SoC) Broadcom BCM2835 and ARM11 CPU running at 1GHz [15]. We use the openssl library. *prime256v1* and *secp384r1* refer to NIST curves P-256 and P-384. *secp256k1* is used in Bitcoin and uses the ECDSA curve. Figure 9 (a) gives the time for generating public/private key pairs, Figure 9 (b) gives the time for generating certificates and Figure 9 (c) gives the time for certificate verification. We run each cryptographic operation 30 times and show the average time. It can be observed that on a $10 Raspberry Pi Zero W, Elliptic Curve performs significantly better than RSA and the corresponding cryptographic operation has acceptable performance. For example, the average time of creating a RSA 1024-bit key pair is around 1112ms while the average time of creating a prime256v1 key pair is around 108ms. The average time for RSA 1024 certificate verification is around 337ms while the average time for prime256v certificate verification is around 289ms. The observation is consistent with observations made in the bibliography in other contexts. The time taken by these Elliptic Curve operations are acceptable while RSA 2048 takes too much time and is not acceptable.

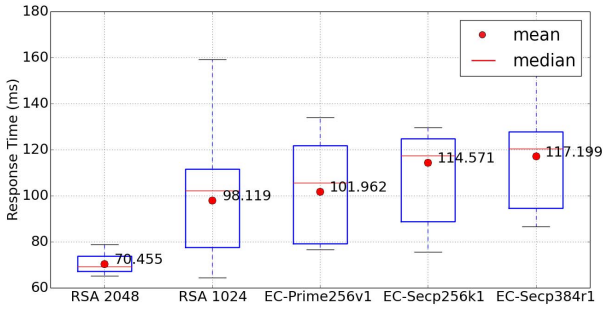
We choose AES-256-CBC to encrypt and decrypt the MQTT transmitted messages between a thing and a controller. Each transmitted message is around 180 bytes in the JSON format.



(a) Key Pair



(b) Certificate



(c) Certificate Verification

Fig. 9. Time for Cryptographic Operations

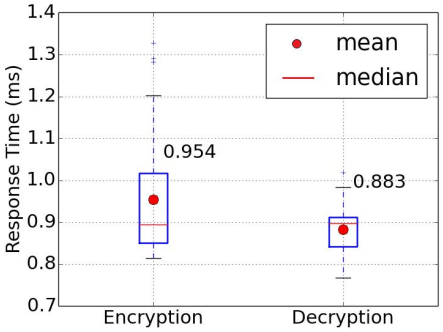


Fig. 10. Time for Encrypt/Decrypt

Figure 10 shows the time required for encrypting and decrypting those messages on Raspberry Pi Zero W. We run each operation 30 times and show the average time. It can be observed that the mean of encryption and decryption time is less than 1ms and very reasonable.

VII. Related Work

In this section, we review the most related work. It can be observed that our work is different from related work. We explore and extend the popular MQTT protocol and build our system on MQTT. We explicitly identify different phases such as pairing, binding and authentication setup and address the security issues in these phases. Our goal is to implement a practical smart home IoT system prototype.

There are efforts toward the IoT system architecture. In [16], a secure VIRTUS middleware was developed. Based on the standard security features of the open XMPP protocol, VIRTUS middleware provides IoT with a secure communication channel that is protected by the authentication (via TLS protocol) and encryption (SASL protocol) mechanisms. An end-to-end security solution for mobility healthcare IoT is proposed in [17]. The scenario includes a secure end-user authentication and authorization based on the certificate-based Datagram Transport Layer Security (DTLS) protocol, and secure end-to-end communication based on session recovery. A lightweight security protocol for the IoT that includes lightweight encryption, authentication, and key management is proposed in [18]. This protocol can achieve security and low resource consumption, and help maintain the sustainability of the system. According to the investigation and analysis of the embedded security in the Internet of Things, an embedded security framework is proposed for IoT as a feature of the software and hardware co-design method [19].

Authentication and key management are active topics in the IoT research. A two-way IoT authentication security system based on the DTLS protocol was proposed and the feasibility (low overhead and high interoperability) of the system is proved in [20]. Sima [21] proposes a lightweight security authenticated and key exchange protocol. However, [22] pointed out the vulnerability of Sima's protocol and modified Sima's protocol to defeat these attacks. For energy-saving in data encryption, an efficient key generation mechanism is proposed based on the triangle security algorithm (TBSA) [23]. This algorithm provides secure data transmission between sensors for a IoT-based smart home system.

VIII. Conclusion

In this paper, we propose *SecT*, a lightweight secure thing-centered IoT communication system. We introduce the key techniques including secure bootstrapping, end-to-end encrypted communication and device mutual authentication mechanism. *SecT* protects user privacy since the cloud server does not store or see user data in plaintext. We introduce the communication protocol in detail and have implemented a real world prototype. We performed extensive experiments and evaluate both networking performance and cryptographic operation performance on a $10 Raspberry Pi Zero W. Our experiments results show that *SecT* is both cost-effective and practical. Although we design *SecT* for the smart home application, it may be extended to other IoT application domains. As future work, we plan to test our design on ra5350f SoC (symtem on a chip), which is used by Edimax plugs and cameras, and other popular hardware platform used by IoT manufacturers.

References

[1] S. Hilton, "Dyn analysis summary of friday october 21 attack." http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/, Oct.2016.

[2] A. Greenberg, "The reaper iot botnet has already infected a million networks." https://www.wired.com/story/reaper-iot-botnet-infected-million-networks/, Oct. 2017.

[3] G. Hernandez, O. Arias, D. Buentello, and Y. Jin, "Smart nest thermostat: A smart spy in your home," in *Proceedings of the Black Hat USA*, 2014.

[4] N. Falliere, L. O. Murchu, and E. Chien, "W32.stuxnet dossier." https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf, February 2011.

[5] R. Chirgwin, "Get pwned: Web cctv cams can be hijacked by single http request - server buffer overflow equals remote control." http://www.theregister.co.uk/2016/11/30/iotcamerascompromisedbylongurl, Nov.2016.

[6] Z. Ling, J. Luo, Y. Xu, C. Gao, K. Wu, and X. Fu, "Security vulnerabilities of internet of things: A case study of the smart plug system," *IEEE Internet of Things Journal(IoT-J)*, vol. 4, pp. 1899–1909, Dec.2017.

[7] Z. Ling, K. Liu, Y. Xu, Y. Jin, and X. Fu, "An end-to-end view of iot security and privacy," in *GlobeCom*, 2017.

[8] mqtt.org, "MQTT," April 2018.

[9] W. McLaughlin, "Understanding password complexity." https://blog.securityevaluators.com/understanding-password-complexity-5e0d23643a2a, November 2016.

[10] Wikipedia, "Simple service discovery protocol." https://en.wikipedia.org/wiki/Simple_Service_Discovery_Protocol, February 2018.

[11] "Sha-1." https://en.wikipedia.org/wiki/SHA-1, 2018.

[12] N. W. Group, "The aes-cbc cipher algorithm and its use with ipsec." https://tools.ietf.org/html/rfc3602, 2003.

[13] Mosquitto, "Mosquitto concepts." https://mosquitto.org/, 2018.

[14] J. Mens, "Authentication plugin for mosquitto with multiple back-ends (mysql, redis, cdb, sqlite3)." https://github.com/jpmens/mosquitto-auth-plug.

[15] "Introducing raspberry pi zero w." https://www.raspberrypi.org/magpi/pi-zero-w/, 2017.

[16] D. Conzon, T. Bolognesi, P. Brizzi, A. Lotito, R. Tomasi, and M. A. Spirito, "The virtus middleware: An xmpp based architecture for secure iot communications." In Computer Communications and Networks (ICCCN), 2012 21st International Conference on (2012), 2012.

[17] S. R. Moosavi, T. N. Gia, E. Nigussie, A. M.Rahmani, S. Virtanen, H. Tenhunen, and J. Isoaho, "End-to-end security scheme for mobility enabled healthcare internet of things," *Future Generation Computer Systems*, vol. 64, pp. 108–124, 2016.

[18] X.-W. Wu, E.-H. Yang, and J. Wang, "Lightweight security protocols for the internet of things." Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017 IEEE 28th Annual International Symposium on (2017), 2017.

[19] S. Babar, A. Stango, N. Prasad, J. Sen, and R. Prasad, "Proposed embedded security framework for internet of things (iot)." Wireless Communication, Vehicular Technology, Information Theory and Aerospace and Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on (2011), 2011.

[20] T. Kothmayr, C. Schmitt, W. Hu, M. Brnig, and G. Carle, "A dtls based end-to-end security architecture for the internet of things with two-way authentication." Local Computer Networks Workshops (LCN Workshops), 2012 IEEE 37th Conference on (2012), 2012.

[21] S. Arasteh, S. F. A. Yang, and H. Mala, "A new lightweight authentication and key agreement protocol for internet of things." Information Security and Cryptology (ISCISC), 2016 13th International Iranian Society of Cryptology Conference on (2016), 2016.

[22] X. Fan and B. Niu, "Security of a new lightweight authentication and key agreement protocol for internet of things." Communication Software and Networks (ICCSN), 2017 IEEE 9th International Conference on (2017), 2017.

[23] S. Pirbhulal, H. Zhang, M. E. E. Alahi, H. Ghayvat, S. C. Mukhopadhyay, Y.-T. Zhang, and W. Wu, "A novel secure iot-based smart home automation system using a wireless sensor network," *Sensor(Switzerland)*, vol. 17, no. 1, pp. 1–19, 2017.