Secure Communication and Firewall Architecture for IoT Applications

Navinkumar Maheshwari
Department of Electronic Systems Engineering
Indian Institute of Science, Bangalore, India 560012
Email: navinkumar@iisc.ac.in

Haresh Dagale
Department of Electronic Systems Engineering
Indian Institute of Science, Bangalore, India 560012
Email: haresh@iisc.ac.in

***Abstract*—Internet of Things (IoT) enables global connectivity to remote smart devices. This technology involves sensing, communication, and processing of real time data received from billions of connected devices with minimal human intervention. The exposure to the Internet and constraints in IoT devices, typically limited memory, low processing ability, and mostly battery based operations make them vulnerable to various attacks. These attacks include but are not limited to Denial of Service (DOS), Man-in-Middle(MIM), Sybil and flooding attacks. Security becomes vital in IoT applications as they are expected to interact with the physical world, especially in safety critical applications like health, defense, automobiles etc. The traditional security model for Internet applications is not suitable for IoT, as it is mostly non-realtime and non-safety critical. Further, end-nodes are not considered to be energy-constrained devices in this model. Therefore, it is important to have alternative solutions that provide meaningful security to IoT devices/applications. In this paper we propose a novel secure communication and firewall architecture suitable for IoT applications. It is based on the idea of off-loading computational load from IoT devices by introducing a server entity in the network. Further, we also discuss design of protocol, entity states and other implementation details. In the end, we compare our solution with the state of the art DTLS protocol (RFC 6347) defined by the IETF.**

***Index Terms*—IoT, network security, communication architecture, DOS, firewall, DTLS**

I. Introduction

It is estimated that billions of devices will be connected around the world via IoT in the near future. The entire infrastructure may contain a combination of low-end and high-end compute systems that can get added or removed dynamically in the network. An IoT device consists of sensors, actuators, communication interfaces, operating system, system software, preloaded applications, and lightweight services [1]. It primarily collects data using sensors, processes it using system software and then sends to actuators for action. Gupta et al. show how continuous monitoring of ECG and other vital parameters can be implemented using Raspberry Pi, Linux and MYSQLdb [2]. IoT is applicable to a wide range of promising applications such as e-health, intelligent transportation systems, smart grid, smart cities and environmental monitoring [3]. The ever increasing role of technology and connectivity in human life has made IoT enabled systems a reality.

IoT by definition means close and timely interaction between remote devices, machines, humans, and environment. Therefore, risk involved is high compare to traditional network applications. It provides malicious users a new attack surface that consists of IoT devices, communication channels between the devices as well as between the devices and the back-end system, IoT-specific back-end applications and back-end data storage [4]. Physically damaging a remote device or sniffing and modification of sensitive data can result in a disaster. Researchers at the University of Michigan have shown how a traffic light controller can be easily hacked and timings manipulated, creating chaos and accidents on road [5]. Kim, Hyun-Jin et al. have put across major IoT exploits and real threat cases in energy service, smart home service and E-health domain [6]. Hacking into an automobile system or health monitoring device may lead to loss of human life. The number of IoT devices is predicted to reach 41 billion in 2020 with an $8.9 trillion market as stated in the 2013 report of the International Data Corporation (IDC) [7]. With this scale of proliferation, bringing privacy and security in challenging environment of IoT is a very critical issue. Security of IoT systems is of paramount importance. However, IoT has its own challenges due to limitations in maximum code complexity, the size of buffer (RAM), the amount of computation feasible in a given period of time, available power and periodic updating of the software [1]. Due to the reasons stated above, it becomes difficult to use traditional Internet security measures like TLS/SSL [8] in IoT systems. They are compute intensive and hence, power-hungry. Therefore, most of the IoT applications in the market today, lack basic security measures. poor physical security, insufficient authentication and authorization of entities, lack of transport encryption, insecure network services, insecure software/firmware etc. are some of the top vulnerabilities that define the attack surface of IoT applications [9].

In this paper, we show how the secure communication architecture proposed by [10] can be modified and integrated with the firewall architecture proposed by authors to improve security. We provide details of implemented prototype of proposed solution on an embedded device.In the end, we compare the proposed solution with the already reported solution in the literature and discuss performance measurements.

The paper proceeds as follows: Section II describes various security issues in the IoT domain. Section III explains the proposed architecture. Section IV describes the firewall

978-1-5386-1182-1/18/$31.00 ©2018 IEEE

architecture. Section V provides implementation details and Section VI gives experimental setup and results followed by conclusion in Section VII.

II. IoT Security Issues

Vulnerabilities introduced in IoT systems because of large attack surface and limitations of IoT systems are discussed in [11]. Some of them are listed below

- Programming errors
- Web-based vulnerability
- Weak access control or authentication
- Improper use of cryptography

As a result of these vulnerabilities, IoT devices are prone to attacks. Table I shows frequently observed attacks in a network to which, IoT systems are susceptible [12]. Therefore, there is a need for lightweight, robust security mechanisms that provide authentication, authorization of the participants along with confidentiality and integrity of the data. To meet the need of a security system for constrained nodes, the IETF working group "DTLS In Constrained Environments (DICE)" is making great efforts to make a DTLS profile for the IoT environment [13]. DTLS is a datagram transport layer protocol that provides secure communication on UDP layer. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery [14]. DTLS/UDP is preferred for IoT applications as it has less overheads compare to TLS/TCP. However, DTLS also has a disadvantage. DTLS uses asymmetric and symmetric key cryptography, during handshake and communication phases respectively. In an IoT environment, an entity may have to make multiple connections with multiple entities simultaneously. Therefore, the use of asymmetric key cryptography (highly time and memory-consuming operation) for each and every connection may lead to rapid memory and power exhaustion.

III. Proposed Architecture

The proposed architecture uses the idea of decentralizing authentication and authorization by using local Certifying Authority(CA). It also off-loads the authentication and authorization processes from embedded devices to a local server. The proposed architecture is inspired by the work in [10]. Below, we define few terms that are used in this paper.

- Entity: Every device participating in the proposed architecture.
- Entity Name: Identifies the entity uniquely.
- Group Name: Depending on its properties, each entity belongs to exactly one group, identified by "Group Name".
- Server Entity: An entity that provides a service on request.
- Client Entity: An entity that initiates service requests.
- Local CA: A local certifying authority that signs every entity's public key. This allows every entity to verify credentials of any other entity in the network using CA's public key.
- Certificate: Certificate of an entity is it's public information (public key, name etc.) encrypted using CA's private key. The process of generating a certificate is called signing.
- Auth: Special type of server entity that takes care of authentication and authorization on behalf of embedded devices.
- Distribution key(DK): Symmetric key used for communication between an entity and Auth.
- Session Key(SK): Symmetric key used for communication between any two entities in the network.
- Message Header: Header used in all communications between entities in the network. It is always sender entity's name, terminated by symbol colon(:). This allows the receiver entity to identify the sender entity.
- Tag: Tag is a first byte of every message communicated, following message header. It is used to differentiate between various types of messages. Here the tag names will include an underscore(_) symbol.
- Nonce: Nonce is a random number generated to verify the identity of a certificate owner. The random number is encrypted using public key signed in certificate and sent to the owner. In return, the same random number is expected, since only the owner possesses private key, only it can decrypt and send it back.
- Symmetrically Encrypted Message: It is a concatenation of encrypted_message_tag, cipher text, and its hash value. (The hash value must also contain the authentication details, for which we use distribution key in generation of hash)

TABLE I
List of IoT attacks

Attack Name	Methods/Strategic Attacks
Denial of Service(DOS)	Flooding with large number of packets and exhausting memory and battery
Sybil	Single Node with multiple identities
Spoofing, Altering, replayed routing information	Impersonating another device or user,creating routing loops, extending or shortening sources' routes
Sinkhole	Attracting network traffic by advertising fake routing update
Hello flood	Using hello packets as a weapon to attack
Acknowledgment spoofing	Spoof the link layer acknowledgments for overhead packets
Flooding	Repeat the request of a new connection until the IoT system reaches a maximum level
De-synchronization	Disruption of an existing connection

A. Local CA

In a PKI architecture, CA is a third party that provides certificates to servers, in general. In the proposed architecture, we use PKIs initial trust-building process between two parties. So every entity needs to have a certificate signed by CA. But it is expensive to get a certificate from globally well-known CAs and it is limited only to global IP addresses. In addition, it works on *Trust Chains* [15] that are often not affordable for an embedded device. Therefore, in the proposed architecture,

we use a public-private asymmetric key pair to certify/verify public modules of all the entities. Henceforth, this public-private asymmetric key pair is referred to as local CA.

B. Auth

Auth is the central body that controls the communication amongst all entities in the network. Being an entity, Auth also needs to have a certificate signed by the local CA.

To facilitate communication between entities, Auth requires to maintain four tables as given below:

- **Group Table:** Contains information about the validity period for a given group name.
- **Entity Table:** Contains information of all the registered entities, which includes Entity Name, Group Name, Public Key, Distribution key, validity, IP address and Destination Port address.
- **Access Table:** Contains information regarding authorization of an entity, i.e. whether an entity is allowed to access a particular entity/group.
- **Session Table:** Contains information regarding active sessions between all the entities in network, such as session keys and their validities. Information is erased if validity of the session is over.

Group Table and *Access Table* are used as inputs by Auth. Therefore, the information required in these tables has to be entered initially by the network administrator.

Auth is a master entity that allows or rejects communication between two entities. It is also responsible for authenticating and authorizing an entity, generating and distributing DKs and SKs. Being a central command center, Auth by default becomes a prime target for attackers, therefore an Auth has to be a non resource-constrained device, withstanding all kinds of attacks on it.

C. Stages for Secure Communication

For all the entities, secure communication takes place in three stages as shown in Figure 1.

Consider an example of a client entity E_1 wants to communicate with a server entity E_2. Auth server entity must be running before the communication process starts.

1) **Registration:** Auth verifies the identity and information of the entities and registers them. The entities become part of the network. Communication between each entity and Auth is secured using DKs.
2) **Session Key Distribution:** E_1 requests Auth to grant communication session with E_2. If allowed, Auth creates a secure link between E_1 and E_2 by distributing a SK.
3) **Communication:** Secure communication between E_1 and E_2 happens using SK.

1) Registration Phase: Every entity has to register itself with Auth to become part of the communication network. The registration process is a two-step request-response sequence as shown in Figure 2. To register itself with Auth, an entity sends a message with *registration_request* tag `REQREG`, its public information (i.e. entity name, group name, public key) and certificate (Figure 3(a)). Auth validates the public information of an entity using local CA's public key. Auth generates a nonce N1, encrypts it using the entity's public key and sends nonce_verification_request message along with its own public information and certificate (Figure 3(b)).

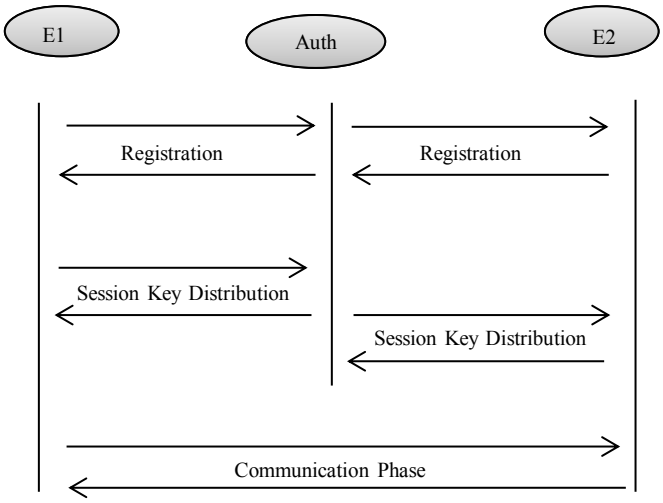


Fig. 1. All the phases of the communication

On arrival of this message, an entity validates Auth's public information in a similar manner. It decrypts N1 using its private key, generates another nonce N2, encrypts both N1 and N2 using Auth's public key and sends it back to Auth. This message is also tagged as nonce_verification_request message (Figure 3(c)).

After receiving this message, Auth decrypts it and verifies N1. Auth generates distribution key, gets its validity period from *Group Table* and registers the entity in *Entity Table*. The registration process is completed at this point. Auth then encrypts N2 along with distribution key and sends it to the entity with *registration_accepted* tag `ACPTREG` (Figure 3(d)).

The entity decrypts the message with *registration_accepted* tag and verifies N2. It acknowledges itself as registered and uses the given symmetric distribution key for all further communication with Auth.

a) Exceptions::

- If either the entity's public information or N1 is not verified by Auth, it sends back a message with *registration_reject* tag to the entity.
- If either Auth's public information or N2 is not verified by an entity, it drops the packet.

By this procedure, E_1 and E_2 get registered and obtain DK_1 and DK_2 as distribution key, respectively.

2) Session Key Distribution Phase: Every entity has to get a SK from Auth to communicate with any other entity in the network. The process of session key distribution is shown in Figure 4 and the formats of transmitted packets are shown in Figure 5. To get a session key, E_1 symmetrically encrypts *access_request* tag `REQACC` along with the name of E_2 and

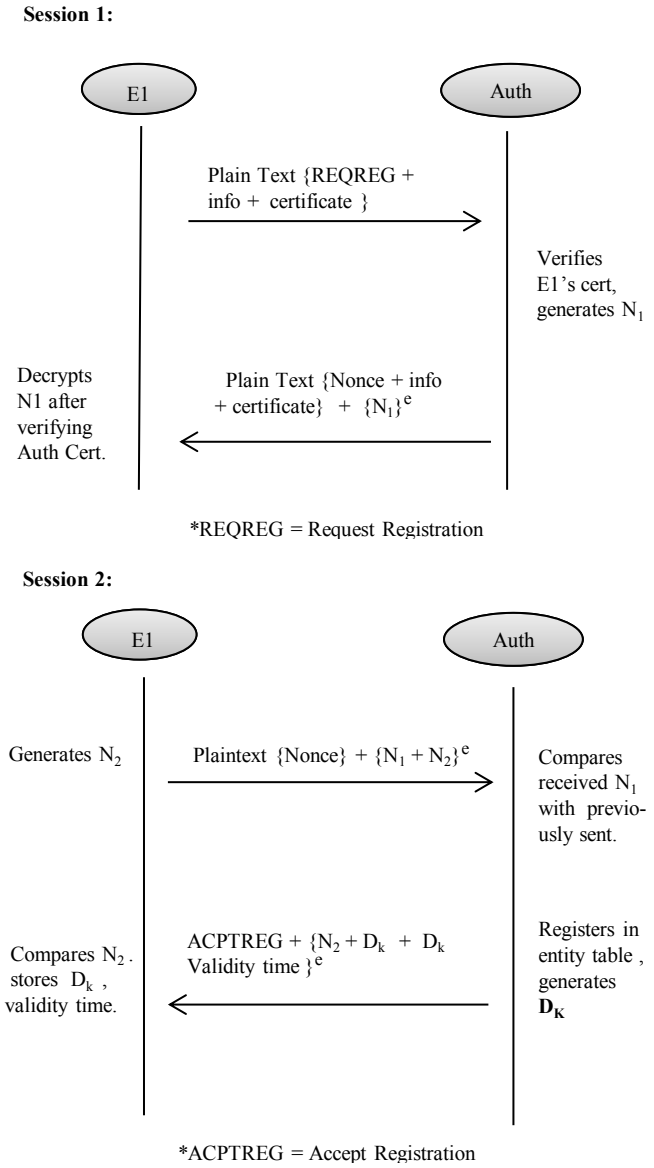


Fig. 2. Registration phase

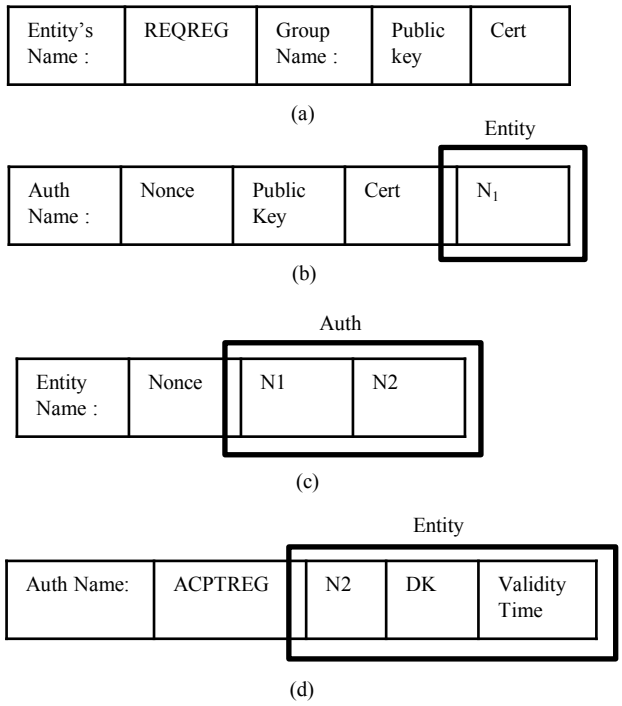


Fig. 3. Packet formats for registration phase; (a) Registration request message (b) Nonce request message (c) Nonce response message (d) Registration accepted message

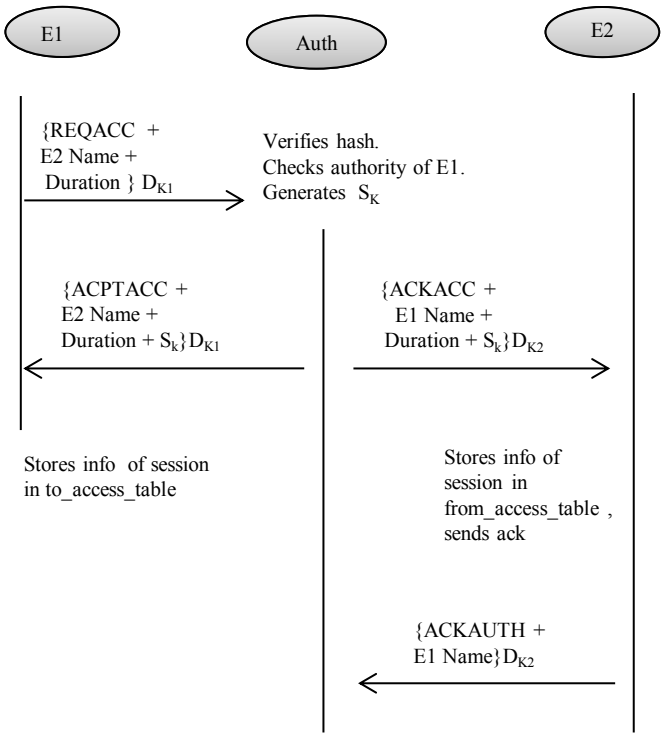
the duration for which it intends to communicate, using DK_1. The duration is specified in minutes. The message header is attached with the symmetrically encrypted message and sent to Auth (Figure 5(a)).

On getting an encrypted message from E_1, Auth generates hash value from cipher text and compares it with the received hash value to verify that integrity and authenticity of the cipher text is not violated. Auth then decrypts cipher text using DK_1. It checks in *Access Table* whether E_1 is authorized by the network administrator to communicate with E_2. It grants E_1 a session with E_2 for up to the maximum duration specified in the *Access Table*. Auth then generates a session key (SK_1) and updates the *Session Table* with related information. Auth encrypts *access_granted* tag `ACPTACC`, E_2's information (name, IP and Port), SK_1 and allowed duration using DK_1, and sends a symmetrically encrypted message along with message header to E_1(Figure 5(b)). After that, it encrypts *access_acknowledgment* tag `ACKACC`, E_1's information, SK_1 and allowed duration using DK_2, and sends a symmetrically encrypted message along with message header to E_2 (Figure 5(c)). On receiving an encrypted message from Auth, both E_1 and E_2 verify the integrity and authenticity of the received message by generating the hash value from cipher text and comparing it with the received hash value. Both entities decrypt message using their respective distribution keys. E_1 updates its *To Access Table* with E_2's information, SK_1 and granted duration. E_2 updates its *From Access Table* with E_1's information, SK_1 and granted duration. E_2 then sends a symmetrically encrypted message with message header *auth_acknowledgment* tag(`ACKAUTH`) to Auth.

To Access Table and *From Access Table* are tables managed by the client and the server entities, respectively, for session related information.

a) Exceptions:

- If the hash value generated from cipher text is not equal to received hash value, the packet is dropped.
- If E_1 is not authorized to communicate with E_2 in *Access Table*, Auth sends a symmetrically encrypted message with message header and *access_reject* tag`RJCTACC` to E_1.



* REQACC= Request Access , ACKACC = Acknowledge Access, ACKAUTH = Acknowledge Auth, ACPTACC = Accept Access

Fig. 4. Session Key Distribution Phase

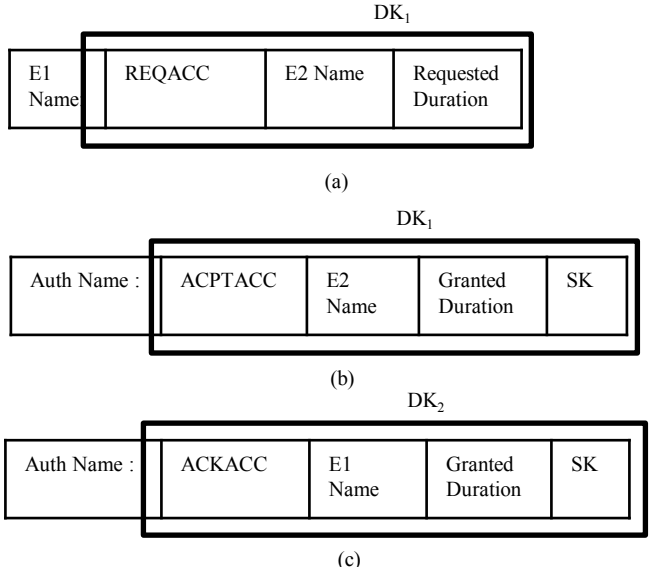
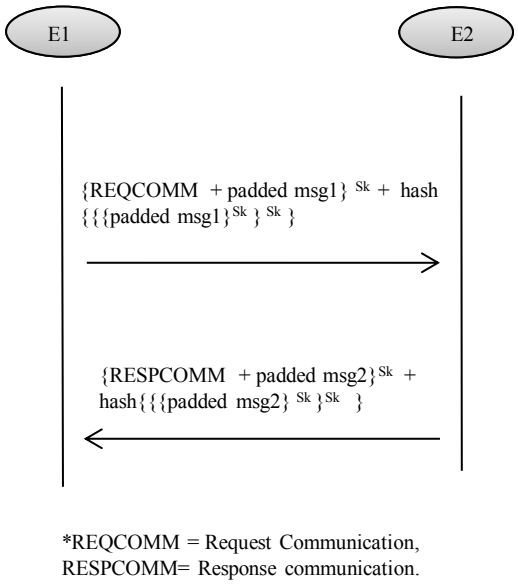


Fig. 5. Packet formats for Session Key Distribution Phase; (a) Access request message (b) Access request accept message (c) Access request acknowledge message



*REQCOMM = Request Communication, RESPCOMM= Response communication.

Fig. 6. Communication Phase

3) Communication Phase: The process during communication phase is shown in Figure 6 and the message formats are shown in Figure 7.

Being a client entity, E_1 has to initiate the communication. To communicate to E_2, E_1 checks access session availability in its *To Access Table* and gets E_2's information. E_1 encrypts its data to be sent to E_2 with *request_comm* tag `REQCOMM` using SK_1. It then sends a symmetrically encrypted message along with message header to E_2 (Figure 7(a)).

On receiving an encrypted message from E_1, E_2 first checks for the availability of access session in its *From Access Table*. It verifies the integrity and authenticity of the received message as explained above. It decrypts cipher text using SK_1 and passes the data to the application layer. If some data is to be sent as a response, the same process is followed, but from E_2 to E_1 and with *response_comm* tag `RESPCOMM` (Figure 7(b)).

a) Exceptions:

- If an access session is not available in *To Access Table* of E_1, it requests Auth to grant access session to communicate with E_2 as explained in the previous subsection.
- If a session is not available in *From Access Table* of E_2, then it ignores the message.
- If the hash value generated from cipher text is not equal to received hash value, the packet is dropped.

IV. Dynamic Firewall

A firewall protects system from unauthorized access. It is an essential requirement in cyber security. But firewalls are generally not implemented in embedded systems, because of their complexity and limited resources. IoT systems, most of the time, may not require a full-fledged firewall. However, primitive conditional inspections are essential. For example, a

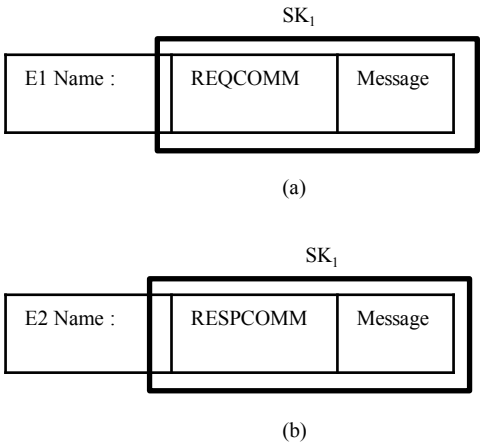


Fig. 7. Packet Formats for Communication Phase; (a) Communication request message (b) Communication response message

resource-constrained device talking only to its gateway, needs only IP address checking to protect it from unnecessary packet processing and DOS attacks. DOS attacks are commonly used against resource-constrained IoT devices as they are very effective. This is mainly because these devices do not have enough resources to implement traditional firewall solutions. For example, if a packet filtering firewall is implemented, a large number of packets can run the device out of memory or drain its entire battery power. This can be addressed if the packet filtering happens at the earliest, to avoid any time consuming memory copy operation. However, this needs in-depth knowledge of network stack implementation. Only OS developers have this knowledge, but the firewall rules are application specific. This requirement is difficult to handle a priori without considering large number of possibilities which affect the code length. On the other hand, application developers know specific requirements for a particular application. But they do not have required knowledge or access to network stack implementation. Therefore, they cannot enforce those conditions for required functionality. The above problems make adaptation of firewall in IoT systems difficult.

In addition, not all IoT systems have Ethernet connection. Many of them rely on other link layer protocols like UART and Bluetooth. In that case, we need to implement firewall rules and packet filtering in the link layer modules that are actually used.

To solve this problem, we propose a novel firewall architecture. We call it dynamic firewall because it is instantiated dynamically at runtime, and it also can be changed dynamically. The check rules that need to be applied are specified in callback functions instead of `if... else...` statements. These functions get a pointer to the data that needs to be checked and are supposed to return Boolean `true` or `false`. The proposed firewall manages the list of such functions and provides APIs to the OS and application developer to make use of it. As the application developer knows which conditions need to be checked, it can write it in the user-defined callback functions. These functions shall return `true` if the conditions are met. Later at runtime, he needs to add these functions in firewall check list using APIs provided by proposed firewall. On the other hand, an OS developer knows the implementation and also where to filter the data for best performance. He needs to place the proposed firewall API at an appropriate place, which passes data through all the listed functions.

A. Firewall Architecture

For generalization of the idea of packet filtering using user-defined functions, we propose two-dimensional linked-list architecture for the firewall. `Group` and `Rule` are two dimensions of the architecture. Here `Group` specifies the type of data and `Rule` specifies the user-defined callback function, i.e., condition. Packet data of the same type are considered to belong to a single group. For example, all packet data coming from the Ethernet can be clubbed under the group `Ethernet`. The groups are managed as a linked-list of data structures. These data structures have properties such as name of the group and first rule pointer. All the rules that need to be applied on group `Ethernet` are chained together under data structure having `Ethernet` as the group name. An OS developer can facilitate packet filtering of various data packet types using the firewall APIs. An application developer has to check for the available data groups for packet filtering provided by OS developer. After that it can put rules to filter data from available groups using firewall APIs.

The firewall provides at least five APIs given below

1) addGroup
2) removeGroup
3) addRule
4) removeRule
5) checkRule

The first four APIs are for application developer to manage groups and rules at runtime. `checkRule` API is for the application developer to use at appropriate places. It takes the data pointer and group name as inputs and returns `true` if the data is to be passed by the firewall. `checkRule` starts from head entry and searches in the group list until the group name matches. Head entry is empty if no entry for any group is present. It starts from the first rule and passes the data through all the rule functions until it receives `true`. By default the data is blocked by firewall if it do not receives `true` from any function.

B. Features

- **Adaptable:** As firewall APIs are easy to place by OS developers and easy to use for application developers, the proposed firewall is easily adaptable to various OSes and devices.
- **Simple:** The proposed firewall uses linked lists to manage rules, which are not very complex to process. It makes an already existing OS more secure with less complexity.
- **Dynamic:** Groups and rules can be added and removed dynamically at runtime by an application developer.

- **Scalable:** Firewall does not restrict on the type of data that can be checked. So it can be used to filter any type of data required for application development.

V. Implementation

The proposed communication architecture is implemented using Python and C languages. The python implementation targets non resource-constrained entities that have availability of `Python 3.5` interpreter. The C implementation focuses on resource constraints and portability. The implementations use AES-CBC-128 and RSA-1024 as symmetric and asymmetric encryption algorithms, respectively. They use XORed double encryption of data for hash generation.

The python code implementation provides an entire secure communication architecture that includes Auth, server and client entities. Further, a software stack that provides solutions for local CA is also implemented. Using this software stack, files required by all the entities for registration are generated. It uses `OpenSSL`[1] for asymmetric key generation. Auth implementation uses `SQLite`[2] to efficiently manage tables as databases. The server/client entity provides APIs to send/receive messages through proposed secure communication architecture.

The C implementation comprises of server and client entities. It provides portability files, that can be used to port the client/sever entity to any device and/or OS. Additionally, it is highly configurable and so provides freedom to an application developer to adapt the implementation and make it suitable for the corresponding platform. These properties are essential as IoT devices and their OSes are diverse. Because of these properties the foot-print of C implementation depends on its configuration. The minimum requirements are 2.5kB of ROM, 76B of DATA RAM and packet buffer size plus 20B of Stack RAM.

Dynamic firewall is implemented using C. As its formation happens at runtime, its implementation requires only 12B of DATA RAM to store head entry data structure. However, additional 12B per new group and new rule are required. It uses 12 bytes of Stack RAM and 316 bytes of ROM. These low memory foot-prints make it a perfect candidate for IoT devices.

VI. Experimental Setup and Results

An experimental setup showcasing "Secure Home Automation" was implemented. The application was implemented using *FreeRTOS* running on *EK-TM4C129EXL* IoT board. Secure communication architecture was ported for FreeRTOS and dynamic firewall was used to drop unwanted packets. Firewall rules were setup in such a way that packets only from Auth and entities allowed by Auth to communicate with the device were passed. The rules checked only IP address and UDP Source Port. The firewall was proven to be effective against ping flood DOS attack from Ubuntu 16.01 with $165\mu s$ interval. At the same time, Auth and other authorized entities were shown to be able to communicate with the device.

Prevention of all attacks related to MIM is theoretically proven as it uses AES and RSA, which are standard encryption algorithms.

For power measurements, we observed the current through controller chip of *EK-TM4C129EXL* and the voltage across it. For initial registration with the local Auth server it consumes $1900mJ$ and later to communicate a single session message with a client it consumes around $0.14mJ$ every time. A DTLS implementation, on the other hand, consumes $1900mJ$ of energy for every single client for every single session message. This is because, in DTLS a server repeats the handshaking process with each client every new session. It is to be noted that in our proposed solution handshaking needs to be done only once during the life time of DK. This explains the large gap between the power consumption during session message exchange. Further, the higher power consumption $1900mJ$ that we observed during handshaking in our implementation can be attributed to sub-optimal implementation of asymmetric encryption algorithm in software. We expect our implementation to perform better if a hardware accelerator support is available for asymmetric encryption. To support our claim we refer to work reported by Rifa-Pous et al [16] for energy and time consumption of various cryptographic operations. Devices (*iPAQ3970*) used by them for testing does not fit into the definition of resource-constrained devices. It exceeds desired specifications of any embedded system. Therefore, this we consider as a benchmark to evaluate any solution. We expect the actual power consumption during the handshake will reduce if we optimize our algorithm or use hardware accelerator to perform asymmetric encryption. We expect power consumption to fall between measurements reported by [16] and our implementation.

Table II shows energy and time required for various encryption unit processes for *iPAQ3970* given in [16]. For handshaking, DTLS requires 2 asymmetric encryption and 1 asymmetric decryption for communication between any two devices. Whereas, in our implementation we require 2 symmetric encryption and 2 symmetric decryption. Therefore, estimated power consumption for a single session message with a client is $1.06mJ$ for proposed solution and $41.56mJ$ for DTLS architecture. It is evident that RSA units require order of magnitude more energy and time than AES. This is the reason for the large gap in power consumption between proposed solution and DTLS implementation.

Consider a scenario of one server and five clients with 2 messages per session. Power consumption for the server is estimated to be $6.36mJ$ and $176.4mJs$, for proposed architecture and DTLS respectively. Here registration process for the server entity is not taken into account. For actual communication, DTLS and the proposed solution uses same standards and hence, same energy is required by both.

[1] *https://www.openssl.org/*
[2] *https://www.sqlite.org/*

TABLE II
ENERGY AND TIME CONSUMPTION PER UNIT ENCRYPTION OPERATION

Operation	Time(ms)	Energy(mJ)
AES-E	0.02	0.07
AES-D	0.71	0.46
RSA-E	1.35	0.82
RSA-D	61.39	40.87

VII. CONCLUSION

In this paper, we have proposed a secure communication and dynamic firewall architecture that out-performs conventional (DTLS) architecture for IoT applications. It off-loads computational load of authentication and authorization from the IoT devices by introducing a local server as Auth. It reduces the power consumed by an IoT device by an order of magnitude. This is very important as IoT devices are energy constrained, whereas, Auth is non resource-constrained. The proposed firewall is proven to be effective against DOS flood attacks. Making it suitable for IoT devices.

Even though security provided by DTLS and proposed architecture are comparable, the resource requirement of DTLS is significantly higher than that in the proposed solution.

In the proposed solution, we assume that the entities are already aware of Auth's details like IP address, Name etc. The future versions may consist of an *Auth Discovery Protocol* that can identify available Auth servers in an existing network infrastructure. It may introduce timing synchronization and also extend support for publisher-subscriber protocol such as MQTT.

REFERENCES

[1] Hossain, Md Mahmud, Maziar Fotouhi, and Ragib Hasan "Towards an analysis of security issues, challenges, and open problems in the internet of things" *Services (SERVICES), 2015 IEEE World Congress on*. IEEE, 2015.

[2] Gupta, M. Surya Deekshith et al. "Healthcare based on IoT using Raspberry Pi." *Green Computing and Internet of Things (ICGCIoT), 2015 International Conference on*. IEEE, 2015.

[3] Zhang, Yuanyu, et al. "On secure wireless communications for IoT under eavesdropper collusion." *IEEE Transactions on Automation Science and Engineering 13.3* (2016): 1281-1293.

[4] Wrona, Konrad. "Securing the Internet of Things a military perspective." *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015.

[5] Jacobs, Suzanne. Researchers Hack Into Michigan's Traffic Lights. MIT Technology Review. August 19, 2014. *https://www.technologyreview.com/s/530216/researchers-hack-into-michigans-traffic-lights/* .June 19,2017.

[6] Kim, Hyun-Jin, et al. "A Study on Device Security in IoT Convergence." *Industrial Engineering, Management Science and Application (ICIMSA), 2016 International Conference on*. IEEE, 2016.

[7] Yang, Yuchen, et al. "A Survey on Security and Privacy Issues in Internet-of-Things." *IEEE Internet of Things Journal (2017)*.

[8] Dierks, Tim. "The transport layer security (TLS) protocol version 1.2." (2008).

[9] Wrona, Konrad. "Securing the Internet of Things a military perspective." *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015.

[10] Kim, Hokeun, et al. "A Secure Network Architecture for the Internet of Things Based on Local Authorization Entities." *Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference on*. IEEE, 2016.

[11] Papp, Dorottya, Zhendong Ma, and Levente Buttyan. "Embedded systems security: Threats, vulnerabilities, and attack taxonomy." *Privacy, Security and Trust (PST), 2015 13th Annual Conference on.* IEEE, 2015.

[12] Nawir, Mukrimah, et al. Internet of Things (IoT): Taxonomy of security attacks." *Electronic Design (ICED), 2016 3rd International Conference on*. IEEE, 2016.

[13] Han, Jiyong, Minkeun Ha, and Daeyoung Kim. "Practical security analysis for the constrained node networks: Focusing on the dtls protocol." *Internet of Things (IOT), 2015 5th International Conference on the*. IEEE, 2015.

[14] Rescorla, Eric, and Nagendra Modadugu. "Datagram transport layer security version 1.2." (2012).

[15] Housley, Russell, et al. *Internet X. 509 public key infrastructure certificate and CRL profile.* No. RFC 2459. 1998.

[16] Rifa-Pous, Helena, and Jordi Herrera-Joancomart. "Computational and energy costs of cryptographic algorithms on handheld devices." *Future internet 3.1 (2011)* pp: 31-48.