A Payload-based Mutual Authentication Scheme for Internet of Things

Mian Ahmad Jan[a], Fazlullah Khan[a], Muhammad Alam[b,*], Muhammad Usman[c]

[a]*Department of Computer Science, Abdul Wali Khan University Mardan, Pakistan*
[b]*Instituto de Telecomunicações, Campus Universitário de Santiago, 3810-193 Aveiro, Portugal*
[c]*School of Computing and Communications, University of Technology Sydney, Australia*

Abstract

The Internet of Things (IoT) is a vision that broadens the scope of the Internet by incorporating physical objects to identify themselves to the participating entities. This innovative concept enables a physical object to represent itself in the digital world. There have been a lot of speculations and future forecasts about these physical objects connected with the Internet, however, most of them lack secure features and are vulnerable to a wide range of attacks. Miniature sensor nodes, embedded in these physical objects, limit the support for computationally complex and resource-consuming secured algorithms. In this paper, we propose a lightweight mutual authentication scheme for the real-world physical objects of an IoT environment. It is a payload-based encryption scheme which uses a simple four-way handshake mechanism to verify the identities of the participating objects. The real-world objects communicate with each other using the client-server interaction model. Our proposed scheme uses the lightweight features of Constrained Application Protocol (CoAP) to enable the clients to observe resources residing on the server, in an energy-efficient manner. We use Advanced Encryption Standard (AES), with a key length of 128 bits, to establish a secured session for resource observation. We evaluate our scheme for a real-world scenario using NetDuino Plus 2 boards. Our scheme is computationally efficient, incurs less connection overhead and at the same time, provides a robust defence against various attacks such as, resource exhaustion, Denial-of-Service, replay and physical tampering.

[*]Corresponding author
Email address: `alam@av.it.pt` (Muhammad Alam)

Keywords: Internet of Things, CoAP, Payload Encryption, Lightweight Authentication, Intrusion Detection, Resource Observation

1. Introduction

Technological advances in the field of wireless, cellular and sensor networks have laid a solid foundation for the IoT. It is a novel paradigm which encompasses the everyday physical world objects by enabling them to interact with each other using the unique addressing schemes [1]. It is estimated that around 50 billion such objects will be connected to the Internet by 2020[1]. These objects will be empowered to sense, process and control the physical world events and numerous phenomena of interest. This integration and interoperable communication will generate an enormous amount of data which needs to be stored, processed, analysed and transmitted in a very systematic manner [2]. Eventually, the IoT will lead us to the Internet of Everything (IoE), where the objects, data and processes will be integral parts of our daily lives. We are moving to an era where the Internet of embedded objects will become ubiquitous by integrating the virtual world of information with the physical world of objects.

The integration of physical objects with the Internet requires various communication models. This requirement will likely add some very ingenious and innovative malicious models to the future Internet [3]. Security provisioning in an IoT framework is a challenging task because each physical object has its own distinguishing features. The identity of each person, object and system connected with the Internet needs to be verified. In the absence of the identity verification, the intruders will gain access to the network and perform various malicious activities. The consequences of these activities are diverse in nature with applications ranging from disabling a home security system, conveying false health readings to practitioners to activating false fire alarms.

Despite all these threats, most of the currently available IoT products in the market lack secured features. Thus, we are about to use products which are vulnerable to a wide range of security breaches. Rather than improving our lives, these products will

[1]http://www.cisco.com/web/solutions/trends/iot/indepth.html

lead us to a new era of cybercrimes. As a result, the IoT will more likely become the Internet of Vulnerabilities (IoV). Recently, Proofpoint Inc.[2], a leading security firm uncovered a cyber-attack involving physical objects. This is considered as the first major security breach in the world of IoT. Over a period of less than two weeks, $750,000$ malicious emails were transmitted from more than $100,000$ devices. Interestingly, more than 25 percent of those devices were real-world objects including televisions, refrigerators and other household appliances. It was so far the simplest attack regarded as misconfiguration and using default passwords were sufficient to conduct such attack. Hence, the IoT is exposed to various security threats which include the botnets along with the thingnets.

In view of the above discussion, we propose a lightweight mutual authentication scheme through payload encryption in this paper. The Constrained Application Protocol (CoAP) [4] is used as the underlying protocol to meet the requirements of the resource-starving objects. The major contributions of our research are as follows.

1. Our proposed scheme verifies the identities of clients and the server communicating with each other. It is a payload-based mutual authentication scheme which incurs a small connection overhead and is computationally simple and robust. A lightweight handshake mechanism consisting of only two round-trip message exchanges is used for the client and server authentication. Both the client and server challenge each other for authentication by generating encrypted payloads. The payload of each message is kept to a maximum of 256 bits.
2. Our proposed scheme uses the lightweight features of CoAP only for the exchange of resources among the clients and server. These lightweight features include asynchronous message exchange, small-sized CoAP header, simpler caching mechanism and parsing complexity. CoAP alone does not provide security features but relies on Datagram Transport Layer Security (DTLS) [5] at the transport layer for security provisioning. However, DTLS is computationally complex and requires ample of network resources which is not the case with most of the resource-starving physical objects of an IoT. Our proposed scheme replaces

[2]http://www.proofpoint.com/about-us/press-releases/01162014.php

DTLS and incorporates all the features necessary for securing the physical objects. Unlike DTLS, our scheme does not introduce a separate protocol layer which further reduces the computation and communication cost. Our scheme can be a lightweight yet robust and secure alternative to the DTLS for the IoT objects, a claim validated by our experimental results.

3. Malicious nodes are restricted from establishing multiple connections with the server at a given time which eliminates the possibility of resource exhaustion and DoS attacks. Each client is allowed to establish only a single connection with the server in order to fairly utilize its limited resources. Our scheme ensures that the resources are provided only to the legitimate clients by a legitimate server.
4. Our proposed scheme is resilient not only to resource exhaustion and DoS attacks but also to the replay attack. Freshness of the notification updates from the server is used as a measure to detect a replay attack.

The rest of this paper is organized as follows. In Section II, related work from the literature is provided. In Section III, we first present an overview of the problem statement followed by our proposed lightweight payload-based mutual authentication scheme. The design decisions and objectives of our proposed scheme are also explained in this section. In Section IV, the resilience of the proposed scheme is analyzed against various attacks during mutual authentication and data exchange. In Section V, we provide the experimental results for our proposed scheme. Finally, the paper is concluded and the future research directions are provided in Section VI.

2. Related Work

In this section, we provide related research securing IoT objects and various authentication schemes to better manage communications between these objects. First, an overview of the CoAP protocol is presented followed by a brief description of various authentication and cryptographic schemes to meet the ever-existing security and privacy challenges in the IoT.

The HTTP-based web technology uses the Representational State Transfer (REST) architecture [6]. This protocol requires extensive memory and computational capabilities for resource provisioning. The objects in an IoT environment are mostly resource-constrained so their capabilities are restricted to support the HTTP-based web services. Hence, the Internet Engineering Task Force (IETF)[3] has formed a special working group known as Constrained RESTful Environments (CoRE). This working group is responsible for developing lightweight protocols to provide web resources for IoT [7]. The CoAP protocol is one such product of this working group which inherits a subset of the HTTP features to meet the requirements of a resource-constrained IoT [4].

CoAP uses a simple request/response interaction model for exchanging the resources among clients and servers. Each client has the option to register itself with a particular server for the resource observation [8]. The registration request is a confirmable (CON) message having an *observe* option which is a 24-bit sequence number. A CON request needs to be acknowledged (ACK) by the server with a matching response. CoAP supports four types of messages: Confirmable (CON), Non-Confirmable (NON), Acknowledgement (ACK) and Reset (RST). The protocol supports various methods such as GET, POST, DELETE and PUT for resource manipulation. The CON and NON messages may carry a request or response with one or more methods applied to it. CoAP has a fixed length binary header of only 4 bytes, hence, fewer resources are consumed. The cost-effective provisioning of RESTful services in Low-power Lossy Networks (LLNs) coupled with low complexity in terms of protocol header, message parsing, asynchronous transaction model and build-in resource discovery makes it an ideal choice for the IoT deployment. As a result of these distinguishing features, CoAP is an ideal replacement for the existing IoT protocols such as MQTT [9] and XMPP [10]. Hence, CoAP is deployed in various applications such as transport logistics [11], home automation system [12], smart cities [13] and freight supervision [14].

Like any other communication network, the information in an IoT environment is susceptible to various types of attacks at different layers. In [15], the authors highlighted various security challenges faced by these networks. The error-prone communication links coupled with the resource-constrained nature of objects restrict the use of Transport Layer Security (TLS) [16]. In addition, the packets may arrive out of order,

[3]https://www.ietf.org/

and may be missing and/or corrupted. Hence, the DTLS is an obvious choice for securing the communication in a CoAP-based IoT. The handshake and the record layers of DTLS incur 25 bytes of overhead in each datagram header. IEEE.$802.15.4$ specifies a physical layer Maximum Transmission Unit (MTU) of only 127 bytes. Hence, only 60-75 bytes are left for the payload after the addition of DTLS, Medium Access Control (MAC) and upper layers headers [17].

DTLS needs to be profiled to make it more friendly toward the resource-constrained networks [18]. Bhattacharyya et al. [19] proposed a lightweight authentication scheme to establish a unicast communication channel. Their scheme is based on symmetric encryption algorithm to reduce the energy and computation costs of sensor-embedded physical objects. The authors claimed that DTLS can be configured to develop an energy-efficient authentication scheme. However, they have not validated their claim. Granjal et al. [20] studied the use of DTLS as the underlying protocol for securing the communication in a CoAP-based IoT environment. The authors argued that the scarcity of payload space may not suit applications having larger payloads. They suggested to employ security at other layers such as compressed form of IPSec. Kothmayr et al. [21] proposed a robust security scheme using the RSA algorithm. They have presented their DTLS implementation in the context of system architecture to achieve high interoperability and low overhead. However, the computational overhead incurred by their handshake mechanism consumes higher energy, primarily due to the use of RSA-based complex cipher suites. In [22], the authors evaluated the performance of DTLS handshaking for the resource-starving objects using the Elliptic Curve Cryptography (ECC). The use of complex and resource-consuming cipher suites of an ECC algorithm result in higher energy consumption. In [23], the authors have proposed the DTLS implementation for smart phones (INDIGO) using CoAP. INDIGO uses extensive resource-consuming cryptographic cipher suites and its use is restricted only to smart phones. In [24], the authors have proposed a lightweight authentication scheme for resource observation in a CoAP-based IoT environment. The identities of the interacting clients and server are validated before establishing an authentication session. The proposed scheme lacks experimental results for determining the efficiency and accuracy of the authentication algorithm.

Although, DTLS-based schemes support a wide range of cipher suites for security provisioning, however, DTLS was originally designed for networks having abundant of resources. The resource-consuming complex cipher suites of DTLS do not consider the message length as a critical design criterion for securing a network. Therefore, using DTLS for an IoT implementation is an expensive choice and may not be an optimal solution for securing the network.

3. A Lightweight Payload-based Mutual Authentication Scheme

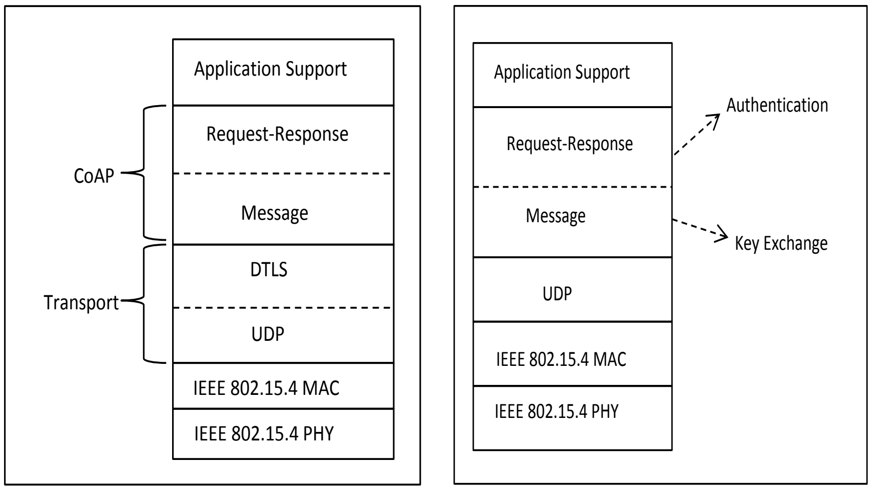
As discussed in Section 2, almost all of the CoAP-based implementations for IoT rely on DTLS for the secure exchange of resources among the physical objects. However, the DTLS-enabled CoAP stack introduces an extra protocol layer for security provisioning which increases the computational and communication cost. In our proposed scheme, no extra protocol layer is added. The absence of an additional protocol layer does not compromise the security of messages exchanged among the clients and server. Authentication is provided at the time of request-response interaction among the clients and server while the session key is exchanged within the payload of transmitted messages. In Figure 1, a comparison of our proposed scheme is provided with the existing DTLS-enabled CoAP stack.

In this section, we present our lightweight payload-based mutual authentication scheme for validating the identities of real-world physical objects. First, the problem at hand is identified which builds the foundation of our proposed authentication scheme.

3.1. Problem Statement

IoT incorporates physical objects which differ from each other in terms of various resources and operational behaviours. The sensor nodes embedded in them provide seamless and interoperable communication. These miniature sensors give a unique ID to each participating object in the IoT paradigm. Any physical object is smart as long as it can identify itself to the participating objects in the network. To make an object smart enough, it requires an IP address and an embedded sensor node [25].

A small scale IoT communication model consisting of various physical objects is shown in Figure 2. Here, the electric heater, the refrigerator and the rest of the objects



(a) DTLS-enabled CoAP Stack (b) Payload-based Mutual Authentication

Figure 1: DTLS-enabled CoAP Stack vs. Our Proposed Scheme

are the clients which communicate with a NetDuino[4] server for the observation of a temperature resource. The server monitors the temperature and provides it to the clients. Each client may specify certain conditions for the notification of temperature readings. The clients and server will not be able to communicate in the absence of an IP address. The presence of an IP address enables an object to perform various RESTful operations in a resource-constrained network.

The resource-constrained network of Figure 2 may be susceptible to a wide range of malicious activities. The intruder may intercept the refrigerator data, manipulate it and replay in other parts of the network. It is also possible that the attacker may inject fabricated data of its own. Therefore, a huge amount of data is at risk which may result in the malfunctioning of the whole network. Similar to a legitimate device, a malicious object also requires an ID to participate in network communication. Each participating object needs to be authenticated to establish its true identity in the network. In the absence of ID validation and authentication, an attacker will always be able to conduct a wide range of malicious activities. An intruder may establish multiple connections

[4]http://netduino.com/

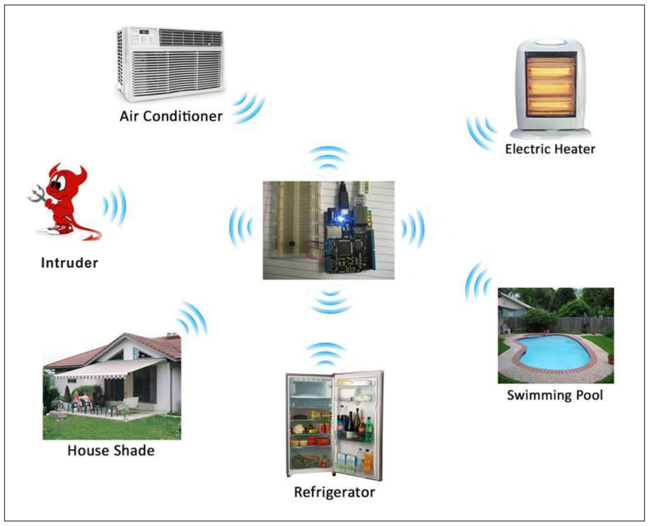


Figure 2: A Vulnerable Internet of Things Connected Environment

with a server at a given time so that multiple network resources will be seized by the malicious object. This results in the denial of services to the legitimate objects and ultimately causes scarcity of resources in the network [26]. As wireless medium is shared among various objects, the intruder may block access to the network resources by continuously emitting jamming signals to interfere with legitimate transmissions [27]. The presence of wireless medium can easily expose objects to eavesdropping as well.

In order to detect and mitigate various types of attacks in a network, extremely lightweight but secure protocols need to be designed in view of the limited resources of embedded sensor nodes. In today's Internet, considerable efforts have been spent on securing the existing standard authentication and authorization protocols such as TLS [16] and Kerberos [28] among others. It indeed saves a lot of efforts if these protocols are customized to be feasible for resource-constrained networks. However, these protocols were not designed with the constrained networks in mind. Therefore, profiling of an existing authentication protocol may not result in an optimal solution. Another major concern is that the existing authentication protocols may not necessarily comply with their original design objectives if they are used outside their intended domains of applications [29].

Almost all of the authentication and encryption schemes in the resource-constrained IoT are based on DTLS protocol. The presence of computationally complex and resource-consuming cipher suites in DTLS provides an expensive secured solution to the objects of an IoT. As discussed in Section 2, most of the DTLS-based encryption schemes emphasize on the use of DTLS without stateless cookies. With the absence of stateless cookies, however, a server will be exposed to replay attacks [30].

In view of the above discussion, our goal is to develop a lightweight CoAP-based authentication scheme for the objects in an IoT paradigm. An important question which arises is "How lightweight does a protocol need to be?"A protocol is sufficiently lightweight if there are ample resources available to other tasks after its deployment. Therefore, the aim of our proposed scheme is to develop an authentication scheme which meets the above requirements. Our proposed authentication scheme uses the basic principles of the CoAP protocol for RESTful interactions. It is a lightweight mutual authentication scheme for any client wishing to interact with a server to establish a secure communication channel. Upon mutual authentication, the clients communicate with the server for the resource observation. Only the authenticated clients are allowed to observe the resources.

3.2. *Payload-based Mutual Authentication*

CoAP and our authentication scheme are not two separate protocols. For the authentication, we have added security features into CoAP to make it more robust, efficient and secure against various malicious activities. Unlike the DTLS-enabled encryption techniques, our proposed scheme provides authentication using the payload of messages exchanged among the clients and the server. Both the client and the server challenges each other during the authentication process. The complete process is performed using four handshake messages where, the payload of each message is kept a maximum of 256 bits as shown in Figure 3. The Advanced Encryption Standard 128-bit (AES-128) in Cipher Block Chaining (CBC) mode [31] is used for the payload encryption. The key size of 128-bit is sufficient for most of the objects in the IoT paradigm due to the limited resources of the embedded sensor nodes. Our proposed scheme is completed using the following four steps:

1. Session/Connection initiation
2. Server challenge
3. Client response and challenge
4. Server response

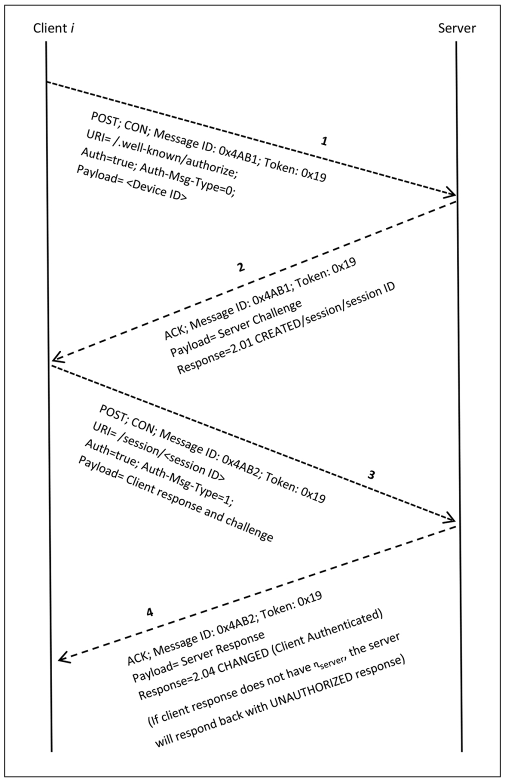

Client i
Server
1
POST; CON; Message ID: 0x4AB1; Token: 0x19
URI= /.well-known/authorize;
Auth=true; Auth-Msg-Type=0;
Payload= <Device ID>
2
ACK; Message ID: 0x4AB1; Token: 0x19
Payload= Server Challenge
Response=2.01 CREATED/session/session ID
3
POST; CON; Message ID: 0x4AB2; Token: 0x19
URI= /session/<session ID>
Auth=true; Auth-Msg-Type=1;
Payload= Client response and challenge
4
ACK; Message ID: 0x4AB2; Token: 0x19
Payload= Server Response
Response=2.04 CHANGED (Client Authenticated)
(If client response does not have η_{server}, the server
will respond back with UNAUTHORIZED response)

Figure 3: Four-way Authentication Handshake

The session initiation is preceded by the provisioning phase. It is a prerequisite offline phase during which the clients share a 128-bit AES pre-shared secret, λ_i, with the server. The pre-shared secret is known only to the server and the client to whom it belongs. The server maintains a table of such secrets, each one belonging to a particular client. Each object has a unique identifier (ID) associated with it which enables the server to perform a table look-up for the identity verification. Upon successful verification, both parties communicate with each other for exchanging the session key. It is assumed that the end-devices are temper-safe to avoid compromising the security primitives in accordance with the Internet Threat Model [32]. This model assumes that pre-shared secrets are hardcoded with the embedded sensor nodes in each physical object at the time of manufacturing and deployment. In case, if an attacker attempts to temper with the hardware of a physical object, an alarm is generated to notify about the security breach.

In the session initiation phase, each client sends a request message to the server similar to a Hello Client message. It is a confirmable request for the creation of a session as a resource at the server. The request is sent to the server URI, */.well-known/authorize*. Each message has a specific token for correlating the request with a matching response. The object ID is transmitted in the message payload. Two options, *Auth* and *Auth-Msg-Type*, are included in the request whose values indicate the type of operation performed on a resource at the server. Upon successful verification of the client ID, the resource *authorize* will be created at the server. The combination of *Auth*=true and *Auth-Msg-Type*=0 indicates a session initiation request to the server.

During the server challenge phase, the server retrieves the object ID from the message payload. Using this ID, the server performs a table look-up for a matching λ_i. If a match is found, the server responds back with an encrypted payload using the AES algorithm. As a new session is created for the client, a response code of 2.01 (*Created*) is included in the message. If the server is unable to find a matching λ_i, it will send a 4.01 (*Unauthorized*) response code. Moreover, the server needs to include the same message ID and the token which are present in the session initiation request. It can do so by simply copying them into the server challenge. This enables the client to correlate a confirmable request with the corresponding response message.

To create the challenge, the server generates a pseudo-random nonce, η_{server}, and a potential session key, μ_{key}, of 128 bits each. The nonce is a temporary number which is used only once by an object in the entire cryptographic communication. At this stage, an encrypted payload is generated by the server. First, an Exclusive OR (XOR) operation is performed on λ_i and μ_{key} using Equation 1. XOR operation is computationally inexpensive and is extremely common as a component in complex ciphers. Moreover, this operation does not leak information about the original plaintext and applying it twice enables the retrieval of original plaintext. In our case, plaintext is the session key.

$$\psi_{\textit{resultant}} = \lambda_i \oplus \mu_{key}. \tag{1}$$

The 128-bit resultant, $\psi_{resultant}$, is appended to η_{server} and encrypted with λ_i to generate a payload of 256-bit using Equation 2. This payload, $\gamma_{\textit{server-payload}}$, is transmitted to the client as a challenge.

$$\gamma_{\textit{server-payload}} = \textit{AES}\{\lambda_i, (\psi_{resultant} | \eta_{server})\}. \tag{2}$$

In the client response and challenge phase, the client needs to decipher the encrypted payload, $\gamma_{\textit{server-payload}}$, to retrieve the potential session key, μ_{key}. If the client is successful to do so, it will have the correct η_{server} and μ_{key}. The η_{server} and the μ_{key} are known only to the server and the λ_i belongs to a specific client. Only a legitimate client can decipher this challenge. An intruder can only eavesdrop on the η_{server} and the μ_{key}, but not the λ_i in accordance with the Internet Threat model. The client uses its λ_i to decipher the payload. Upon successful deciphering, the client has authenticated itself. As mutual authentication requires both parties to be verified, the server also needs to authenticate itself. The client generates a new encrypted payload similar to the server. First, an XOR is performed on η_{server} and λ_i using Equation 3.

$$\psi_{\textit{resultant}} = \eta_{server} \oplus \lambda_i. \tag{3}$$

The 128-bit resultant, $\psi_{resultant}$, is appended with η_{client} and encrypted with μ_{key} to generate an encrypted payload as shown in Equation 4. The 256-bit encrypted payload, $\gamma_{\textit{client-payload}}$, is transmitted to the server as a challenge.

$$\gamma_{\textit{client-payload}} = \textit{AES}\{\mu_{key}, (\psi_{resultant} | \eta_{client})\}. \tag{4}$$

Finally, in the server response phase, the server deciphers the encrypted payload, $\gamma_{\textit{client-payload}}$, of the client challenge to observe the η_{server} in it. If present, the server realizes that the client has successfully authenticated itself. The server retrieves the

η_{client} and creates an encrypted payload of its own by appending the η_{client} to μ_{key} and encrypting with λ_i as shown in Equation 5. Next, the 256-bit encrypted payload, $\gamma_{\textit{server-payload}}$, is transmitted in response to the client challenge.

$$\gamma_{\textit{server-payload}} = \textit{AES}\{\lambda_i, (\eta_{client}|\mu_{key})\}. \tag{5}$$

At this point of time, the status of the resource changes to $Authenticated$ at the server because the client is successfully authenticated and is eligible to observe the resources. However, the client has yet to verify the authenticity of the server, so it decrypts the payload, $\gamma_{\textit{server-payload}}$, and observe η_{client} in it. As the client is the one which generated the η_{client}, the client comprehends that the response is from a legitimate server. At this stage, both the client and the server are mutually authenticated and they have agreed upon a common session key, μ_{key}, for exchanging the data packets.

In our authentication scheme, the *Auth* and the *Auth-Msg-Type* options play an important role. Both of these options are critical and unsafe-to-forward. A critical or elective option is related to the endpoint while the safe or unsafe-to-forward is used in the proxy context. If an endpoint is unable to understand a request message having a critical option, it must return a 4.02 *Bad Option* response to the sender. These options do not have any default values unlike the core options such as *Max-Age*, *Block*, *Uri-Path* and *Uri-Authority* [4]. Each of these options is to be assigned a number by the Internet Assigned Numbers Authority (IANA)[5]. The *Auth* has a length of 0 byte because if the message header indicates that this option is present, it is *true* and thus there is no need to occupy a byte. For the lightweight authentication schemes, it is important to have a simple message format with extremely lightweight options and related fields. The formats of these two options are shown in Table 1.

The presence of the *Auth* option indicates that the POST method carries an authentication payload in the request message. The *Auth-Msg-Type* is always used in combination with the *Auth* and its value indicates the type of authentication request by the client. If its value is 0, it indicates a session initiation request and if its value is 1, it

[5]https://www.iana.org/

No.	C	U	N	Name	Format	Length	Default
TBD	Yes	Yes	No	Auth	empty	0	(none)
TBD	Yes	Yes	No	Auth-Msg-Type	uint	1	(none)

*C=Critical, *U= Unsafe-to-Forward, *N=NoCacheKey, *Length in Bytes

Table 1: Format of the Authentication Options

indicates a client response and challenge respectively. The four steps of our payload-based mutual authentication along with the data exchange is shown in Algorithm 1.

4. Resilience of the Proposed Scheme Against Various Attacks

Our payload-based mutual authentication scheme is resilient to resource exhaustion, DoS and replay attacks. Each client is restricted to establish a single connection with a given server at a particular time which eliminates the possibility of resource exhaustion. Resource exhaustion occurs when the resources necessary to perform an action are entirely consumed, therefore preventing that action from taking place. A client establishes more than one connection with a server to acquire multiple resources which causes scarcity of such resources to other clients. Resource exhaustion results in an unfair distribution of resources, ultimately leading to a DoS attack.

The client of a resource may be an intruder or a legitimate object. Each client sends a session initiation request to the server in order to authenticate itself. The server checks its table for a corresponding object ID to verify if the client has an ongoing established session for data exchange after a successful authentication. If the given client has an ongoing session with the server, the request for a new session is discarded. Recall that the session initiation request is transmitted in a CON message for which an ACK or an RST response is mandatory. After the transmission of a CON message, the client triggers its timer and waits for a response within $a \times r$ seconds, where a is the acknowledgement timeout and r is the acknowledgement random factor. The acknowledgement timeout is the amount of time a client must wait for an ACK or an RST response before a re-transmission attempt. Its default value is 2 seconds. The acknowledgement random factor ensures that there is no clashing in the timeout values used for subsequent transmissions. Its default value is 1.5 and it must always be greater

Algorithm 1 Lightweight Payload-based Mutual Authentication

```
1: Initialization:
      (a) Each Client C$_i$ is provided with a unique Object_ID$_i$ and $\lambda_i$
      (b) A server $S$ is provided with all Object_ID$_i$ and $\lambda_i$ stored in an array $A[\,][\,]$
2: Step 1: [Input: (Object_ID$_i$, $\lambda_i$)]        ▷ Object_ID$_i$, $\lambda_i$=$2^{128}$, where $i$=1,2,3,...,$N$
3: for $i = 1 : N$ do        ▷ Nested For loop generates a two-column server table
4:     for $j = 1 : 2$ do
5:         input (A[i][j])        ▷ Object ID and $\lambda$ of C$_i$ are stored in the array
6:     end for
7: end for
8: Step 2 [Session Initiation]: C$_i$ sends a CON message containing Object_ID$_i$ in the payload to $S$
9: Step 3 [Server Challenge]: $S$ retrieves Object_ID$_i$ to find a matching $\lambda_i$
10: if Object_ID$_i$ == A[i][j] then        ▷ Object ID of the client matches the Object ID in the server table
11:     Session Created $2.01$
12:     Step 4: $S$ responds with an encrypted payload, AES $\{\lambda_i, (\lambda_i \oplus \mu_{key}|\eta_{server})\}$
13:                                  ▷ where $\mu_{key}, \eta_{server} = 2^{128}$ and message size=$2^{256}$
14: else
15:     C$_i$ Unauthorized $4.01$
16: end if
17: Step 5 [Client Response & Challenge]: C$i$ deciphers challenge and responds with an encrypted payload, AES
    $\{\mu_{key}, (\eta_{server} \oplus \lambda_i|\eta_{client})\}$,
18: Step 6 [Server Response]: $S$ checks $\eta_{server}$ in the client challenge by comparing against the $\eta_{server}$ generated
    in the server challenge
19: if Both matches then
20:     [Access Granted]- C$_i$ Authenticated
21:     Step 7: $S$ responds as AES $\{\lambda_i, (\eta_{client}|\mu_{key})\}$        ▷ message size=$2^{256}$
22: else
23:     [Access Denied]-C$_i$ Unauthenticated
24: end if
25: Step 8: C$_i$ compares $\eta_{client}$ of Step 5 and Step 6.
26: if Both matches then
27:     $S$ is authenticated
28:     Step 9 [Data Exchange]: Mutual data exchange between $S$ and $C_i$ take place.
29: else
30:     $S$ is unauthenticated
31: end if
```

than 1, i.e., $1 < r \leq 1.5$. A client can re-transmit a CON request up to 4 times after the initial failed attempt provided it uses the same token and message ID.

If the client is an intruder, there will be a mismatch of IDs with the server, i.e., the object ID does not match with any of the object ID stored within the server table. The server responds back with an RST message indicating an invalid object ID. At this point, the server does not know if a client is an intruder or a legitimate one. An intruder may re-transmit the same CON message up to 4 times provided it uses the same object ID, message ID and token for each re-transmission attempt. After exhausting all attempts, the client is declared as an intruder and any further attempts are considered as attacks. The server refrains from any further communication with the given client, i.e., the intruder. On the other hand, re-transmission with a different ID will generate an RST response which is an indication that no further communication will take place for establishing a session. An intruder may eavesdrop on the wireless link between a legitimate client and the server, stole the object ID and use it for communication with the server. There will be a match of IDs and the server will generate a challenge. As the intruder does not have a valid λ_i, so it will not be able to decipher the challenge. In case of a legitimate client, a match will always be found in the look-up table unless its object ID is tempered. Even if the object ID is tempered by an error-prone communication link or an intruder, the client has the option to re-transmit the CON message after waiting for $a \times r$ seconds. A match with the server table is not a guarantee that a session will be negotiated with a given client. If the IDs match, the server further evaluates the request to verify if an ongoing established session exists with the given client. If a session is found, the server responds back with an ACK message indicating that the client needs to refrain from further re-transmissions.

The object ID of a client plays a crucial role in resource allocation and utilization. In the server table, each ID is linked with a matching λ_i, the key required for deciphering a challenge. The use of an invalid object ID generates an RST response while eavesdropping results in an unsuccessful attempt of deciphering the server challenge. In both cases, the intruder fails to authenticate itself and is unable to observe the resource, i.e., temperature readings, at the server. The procedure for detecting the resource exhaustion attack is shown in Algorithm 2.

Algorithm 2 : Detection of Resource Exhaustion Algorithm

```
 1: Initialization: $A[\,][\,]$, Object_ID$_i$, Re_Try
 2: Step 1: [Input: (Object_ID$_i$, $\lambda_i$)]
 3: for $i = 1 : N$ do
 4:     for $j = 1 : 2$ do
 5:         input (A[i][j])
 6:     end for
 7: end for
 8: Step 2: Client C$_i$ sends a session initiation request
 9: Step 3: Server $S$ retrieves Object ID from the session initiation request
10: for $i : 1 : N$ do                                      ▷ Search through the server table
11:     if Object_ID$_i$ == A[i][j] then      ▷ Object ID of the client matches the Object ID in the server table
12:         Step 4: Check for an ongoing established session
13:         if OnGoingSession == True then
14:             Step 5: Discard Session Initiation Request
15:             Step 6: Send ACK response to C$_i$      ▷ ACK informs C$_i$ that this client is already observing resources
16:         else
17:             Generate a Server Challenge and send to C$_i$
18:             if C$_i$ decipher the challenge then
19:                 C$_i$ is a legitimate client      ▷ S proceeds with establishing a session after successful authentication
20:             else
21:                 C$_i$ is an Intruder                   ▷ S refrain from further communication
22:             end if
23:         end if
24:     end if
25:     if Object_ID$_i$ does not match A[i][j] then
26:         Step 7: $S$ responds back with an RST
27:         Step 8: C$_i$ does not proceed with any re-transmission attempts
28:     else
29:         Step 9: C$_i$ re-transmits after waiting for $a \times r$ seconds
30:         Re_Try++
31:     end if
32:     if Re_Try > 4 then
33:         C$_i$ is an intruder and is blacklisted
34:     else
35:         Step 10: C$_i$ re-transmits again
36:     end if
37: end for
```

Apart from resource exhaustion and DoS attacks, our scheme is resilient to the replay attack as well. In our authentication scheme, η_{server} and η_{client} are generated by a pseudo-random number, R_i, which is appended to a timer, T_i. This combination of R_i and T_i assures that an intruder will find it even more difficult to replay messages. Here, T_i is used to guarantee that η_{server} and η_{client} are non-reproducible while R_i is used to ensure that η_{server} and η_{client} are non-predictable. The non-reproducible nature of T_i and the non-predictable features of R_i make it quite difficult for an intruder to launch a replay attack. If in case, an intruder manages to capture μ_{key}, it may communicate with a given client by posing itself as the server of the network. The intruder may intercept the data in-transit between the client and a server, uses the captured μ_{key} and replay the incoming data from the server to the client. Moreover, it may replay the captured data to other clients in the network as well. This type of replay attack is short-lived and last only for the duration of a particular session.

Upon successful mutual authentication, both the client and the server are authorized to use μ_{key} for exchanging the data packets. Each client sends a registration request to the server for resource observation. The Registration Request Message (RRM) has an *observe* option, which has a 24-bit sequentially incremental sequence number. When the server realizes the presence of an *observe* option in an RRM, it registers the client and notifies it upon each state change of a resource. The client may specify certain conditions for the transmission of data from a server, also known as notification updates. Upon transmission to a server, each client stores the transmitted RRM in its queue, which has a unique token and a message ID for correlating the notification updates from the server. The incoming notifications from the server will have an incremental sequence number, a token and a message ID similar to an RRM. Potentially, a single RRM can generate an enormous amount of notification updates. Once the conditions specified by a client for notification updates are fulfilled at the server, data packets, β_{data}, are transmitted to the given client as shown in Equation 6.

$$\kappa_{\textit{client-server}} = \textit{AES}\{\mu_{key}, \beta_{\textit{data}}\}. \tag{6}$$

In Equation 6, β_{data} is encrypted with μ_{key}. This encrypted data can only be deciphered by the client which possesses a valid λ_i and μ_{key}.

In Figure 4, an intruder A eavesdrops on a full-duplex wireless link between the server S and a client C1. It captures μ_{key}, seizes the data packets and replay them to C1, C2 and C4 along the way by posing to be a legitimate server. To detect a replay attack, each client compares the message ID and token of an incoming notification against the similar parameters of an RRM. If these parameters match, the client checks the sequence number of the incoming notification. To do so, the client can make use of a simple logic by comparing the incoming notification with the previously received notification from a server as shown in Equation 7.

$$\begin{aligned}\Omega_{freshness} = fresh \qquad when[(V_i < V_j) \wedge (V_j - V_i < 2^{23})] \vee \\ [(V_i > V_j) \wedge (V_i - V_j > 2^{23})]. \quad (7)\end{aligned}$$

In Equation 7, we have used the freshness of the notification updates, $\Omega_{freshness}$, from a server as a measure to detect a replay attack. In this equation, V_j and V_i are the 24-bit sequence numbers of an *observe* option in the incoming and the previously received notifications. A notification is fresh and latest if V_j is greater than V_i and their difference is less than 2^{23}. An incoming notification is also fresh if V_j is smaller than V_i and their difference is greater than 2^{23}. The latter is the case when the value of V_j rolls over [33].

In Figure 4, the intruder may or may not alter the sequence numbers of the incoming notifications from a server. Irrespective of the alteration, the clients C2 and C4 can easily detect a replay attack because these incoming notifications are intended for C1 only. There will be a mismatch between the token and message ID of an RRM and those of the incoming notifications at C2 and C4 respectively. If the sequence numbers of the notifications are altered, C1 can easily detect it by comparing the notifications with the previous one. However, if the intruder replays the notifications without any modification, it becomes a tricky situation as C1 is indeed waiting for such notifications from the server. There will be a match of the tokens and message IDs between an RRM and the incoming notifications. Moreover, the sequence numbers are also in the same order as expected by this client. To solve this puzzle, C1 may use a time-stamp field in

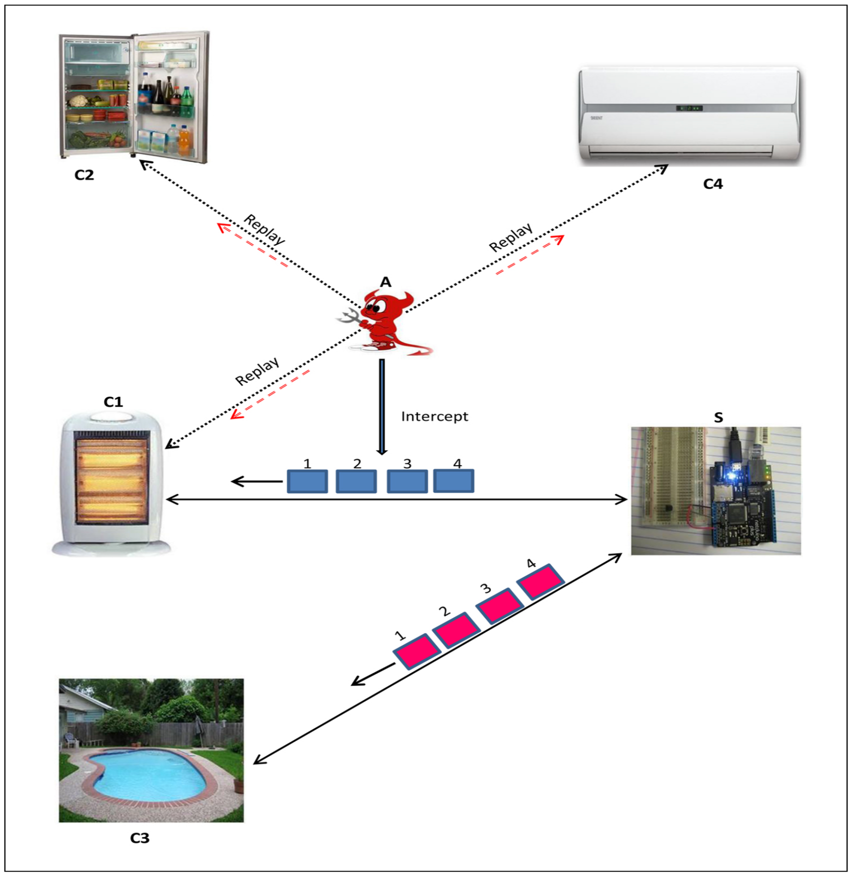


Figure 4: The Replay Attack in an IoT Environment

the RRM and the notification updates [34]. Alternatively, C1 may store in a queue the notifications previously received from a server within the acknowledgement timeout. As the incoming notifications at C1 are replayed by an intruder, their timeout values play a crucial role. At this point of time, C1 does not know whether the incoming notifications are replayed or not. Therefore, it checks the sequence number of these notifications against the previously received notifications in a sequential order. For example, the incoming notification, V_n, is checked against V_{n-1} to determine if it was received within the acknowledgement timeout after the successful reception of V_{n-1}. The entire mechanism of replay attack detection is shown in the flowchart of Figure 5.

In our proposed scheme, an intruder can only launch a replay attack during the

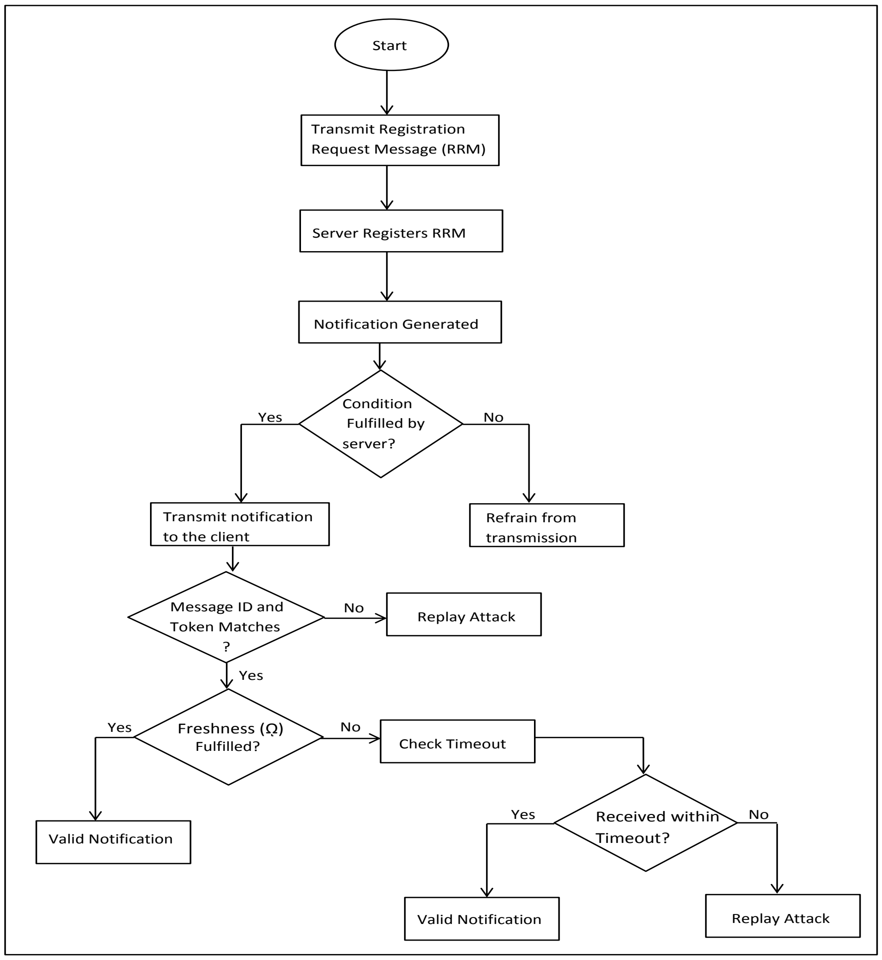


Figure 5: Flowchart for the Replay Attack Detection

exchange of data between a legitimate client and a server. Replay attack is not possible at the time of establishing a session, i.e., mutual authentication, because each client has its own λ_i known only to the given client and the server. An intruder may capture the server challenge payload and retrieve η_{server} and μ_{key} from it. However, it still require λ_i for encrypting the payload. The lack of an authentic λ_i will generate a suspicious payload which will be identified easily by the clients.

5. Experimental Results

In this section, we provide the experimental results for our proposed scheme. We used NetDuino Plus 2 boards for the client and server interaction model. We performed our experimental work using .NET Micro Framework, a .NET Framework platform for resource-constrained devices with at least 256 KB of flash and 64 KB of RAM. The NetDuino server uses a DS18B20 temperature sensor to obtain the temperature readings. Upon successful authentication, the clients are allowed to observe the temperature resource. We use the CoAPSharp[6] library which provides a very basic communication model for resource exchanges. For the authentication provisioning, we have created our own library, CoAPMicro, which runs on top of the CoAPSharp and uses its communication model for security provisioning. Our library is efficient and robust, and consumes relatively less amount of resources. To justify our claim, a detailed comparison against various existing schemes is presented here.

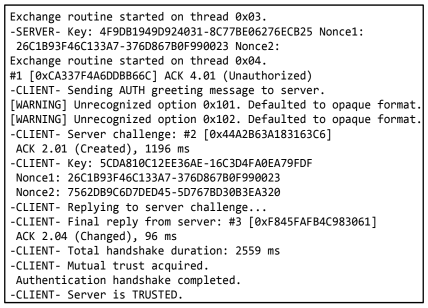
5.1. Authentication

Each client needs to authenticate itself for observation of the resources. In Figure 6(a), the successful handshake between a client and the server is shown. The *Auth* and the *Auth-Msg-Type* are the extended options used only for our authentication purpose. They are yet to be assigned numbers by IANA, so they appear as unrecognized. We use temporary numbers $0x101(257)$ and $0x102(258)$ for these options in our library using the *option registry* mechanism. The client and the server have successfully agreed upon a common session key and the handshake duration along with the round-trip response time is obtained. In Figure 6(b), the client is unable to decipher the server challenge as it does not possess the required secret, λ_i. The outcome is an incorrect session key which causes the denial of access to the resource. In both cases, the presence or absence of the pre-shared secret determines the outcome of the authentication.

5.2. Handshake Duration

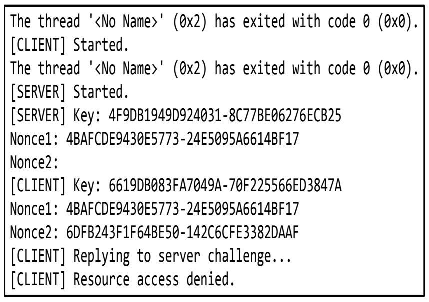
The handshake duration is computed as the sum of time taken by two round-trip messages, i.e., the session initiation request and the client response & challenge. The

[6]http://www.coapsharp.com/



```
Exchange routine started on thread 0x03.
-SERVER- Key: 4F9DB1949D924031-8C77BE06276ECB25 Nonce1:
 26C1B93F46C133A7-376D867B0F990023 Nonce2:
Exchange routine started on thread 0x04.
#1 [0xCA337F4A6DDBB66C] ACK 4.01 (Unauthorized)
-CLIENT- Sending AUTH greeting message to server.
[WARNING] Unrecognized option 0x101. Defaulted to opaque format.
[WARNING] Unrecognized option 0x102. Defaulted to opaque format.
-CLIENT- Server challenge: #2 [0x44A2B63A183163C6]
 ACK 2.01 (Created), 1196 ms
-CLIENT- Key: 5CDA810C12EE36AE-16C3D4FA0EA79FDF
 Nonce1: 26C1B93F46C133A7-376D867B0F990023
 Nonce2: 7562DB9C6D7DED45-5D767BD30B3EA320
-CLIENT- Replying to server challenge...
-CLIENT- Final reply from server: #3 [0xF845FAFB4C983061]
 ACK 2.04 (Changed), 96 ms
-CLIENT- Total handshake duration: 2559 ms
-CLIENT- Mutual trust acquired.
 Authentication handshake completed.
-CLIENT- Server is TRUSTED.
```

(a) Resource Access Granted



```
The thread '<No Name>' (0x2) has exited with code 0 (0x0).
[CLIENT] Started.
The thread '<No Name>' (0x2) has exited with code 0 (0x0).
[SERVER] Started.
[SERVER] Key: 4F9DB1949D924031-8C77BE06276ECB25
Nonce1: 4BAFCDE9430E5773-24E5095A6614BF17
Nonce2:
[CLIENT] Key: 6619DB083FA7049A-70F225566ED3847A
Nonce1: 4BAFCDE9430E5773-24E5095A6614BF17
Nonce2: 6DFB243F1F64BE50-142C6CFE3382DAAF
[CLIENT] Replying to server challenge...
[CLIENT] Resource access denied.
```

(b) Resource Access Denied

Figure 6: The Authentication Process

client's session initiation request is acknowledged through a server challenge whereas the client response & challenge is acknowledged through a server response. The handshake duration, $d_{handshake}$, is computed at the client-end using Equation 8.

$$d_{\mathit{handshake}} = \mathbb{T}_{session} + \mathbb{T}_{challenge} + \delta_{proc}. \tag{8}$$

In this equation, $T_{session}$ is the round-trip time taken by session initiation request, $T_{challenge}$ is the round-trip taken by client response & challenge and the δ_{proc} is the processing time taken by the client. The processing time at the server is part of the two round-trip messages.

To compute the handshake duration, we perform 20 random handshakes between the clients and the server. To determine the variability and accuracy among the readings, we compute the standard deviation using Equation 9.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}. \tag{9}$$

Here, σ is the standard deviation, N is the total number of readings, μ is the mean value and x_i is the actual handshake duration of each individual reading.

In Figure 7, we have compared our scheme with the CoAP-based DTLS implementation (INDIGO) for smartphones. DTLS* represents the handshake between a smartphone and a standard computer. In this case, the smartphone acts as a client whereas the standard computer acts as a server. DTLS$^+$ represents the smartphone as a server and the computer as a client. Both these implementations use DTLS on the client and the server. The creation of stateless cookie at the server and the exchange of computationally complex certificates and raw-public keys contribute to a higher handshake duration for DTLS* and DTLS$^+$ respectively. Moreover, the simultaneous execution of multiple processes inside an android device contributes to a much higher standard deviation values for these schemes.

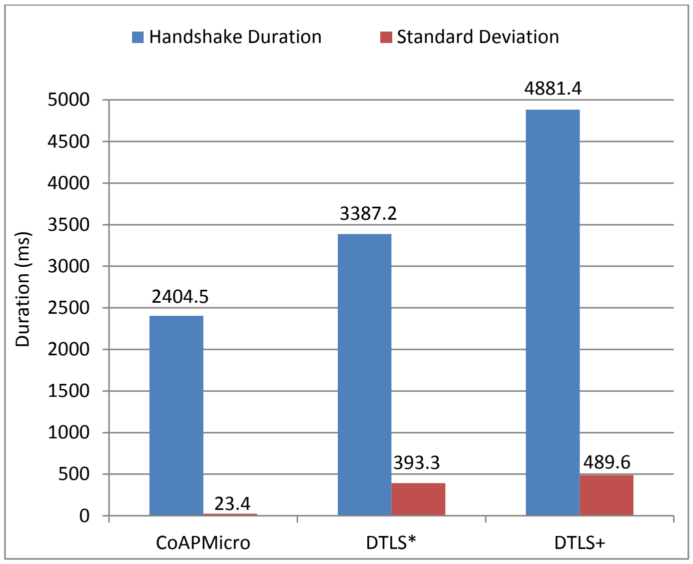
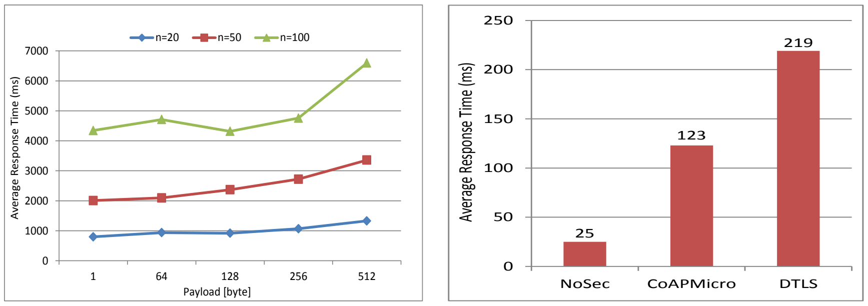


Figure 7: The Handshake Duration

5.3. Average Response Time

In our scheme, the CoAP messages are exchanged asynchronously over the UDP sockets. Each client maintains the record of the transmitted CON request messages to keep track of their transit through the network. When a matching ACK or an RST response is received for such messages, the exchange is considered as successful.

In Figure 8(a), the average response time for each message is computed for a server handling multiple requests at a given time. The response time increases with the increase of simultaneously transmitted messages. When the number of such requests, n, is 100, the response time is significantly higher which can ultimately cause congestion, scarcity of resources and denial of services to the clients in the network. The response time is considerably lower for 20 and 50 of multiple requests of such type and the payload size has little impact on the average response time for these values of n. In Figure 8(b), the average response time for a single confirmable message of 1 byte is compared with those of the DTLS and the CoAP protocol with no security. As explained previously, the presence of certificates, raw-public keys and expensive flight-based authentication makes DTLS as an expensive choice for the objects of an IoT.



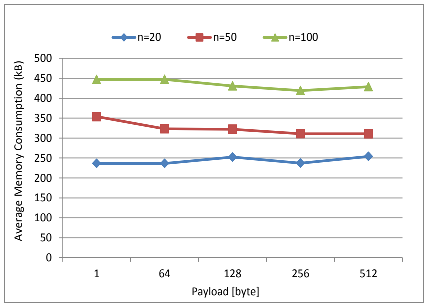
(a) Simultaneous Transmissions (b) Response Time at 1 Byte Payload

Figure 8: The Average Response Time

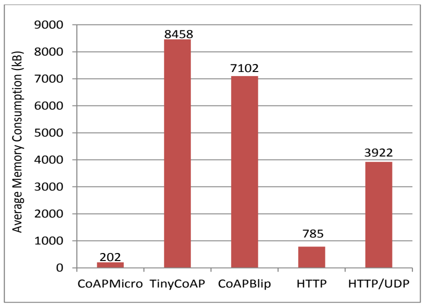
5.4. *Average Memory Consumption*

The average memory consumption of a message at the compile time is obtained by using the Debug.GC() method of the Microsoft.SPOT.Native assembly. In Figure 9(a), the average memory consumption is computed for varying payload sizes. The number of request messages has a significant impact on the memory consumption whereas the payload of each message has a minor impact. This is due to the fact that the server allocates memory on per message basis rather than per byte basis.

In Figure 9(b), we compare our scheme with the existing schemes for a confirmable message of 500 bytes. CoapBlip [35] and its variant TinyCoAP [36] allocate a substantial amount of memory to the messages at the compile time. CoapBlip is an adaptation of the standard C libraries which require TinyOS component for its installation on a sensor node. The use of C libraries is too complex for the resource constrained sensors embedded in a physical object. On the other hand, HTTP/UDP has a low memory footprint as it does not provide a reliability mechanism or a request/response matching. Our



(a) Simultaneous Transmissions



(b) Consumption at 500 Byte Payload

Figure 9: The Average Memory Consumption

scheme ensures that sufficient memory is available to other tasks at the compile time. The allocated memory is immediately released upon a successful message exchange.

5.5. Detection of Replay Attacks

As an application layer protocol, CoAP is exposed to various denial-of-service (DoS) attacks. Even in the presence of an authentication scheme, intruders always try to sneak into a network to conduct malicious activities. The number of such activities increases with the increase in the number of intruders in the network. In Table 2, the number of replay attacks at the time of session initiation and data exchange is computed over a period of 60 seconds.

Time (sec)	Intruder A^*	Intruder B^*	Intruder A^+	Intruder B^+
0-10	3	5	8	2
11-20	6	2	4	4
21-30	3	4	11	7
31-40	4	0	4	3
41-50	1	2	8	3
51-60	7	6	0	2

Table 2: Number of Detected Replay Attacks

Here, A^* and B^* represent the number of replay attacks launched by intruders A and B during session initiation request, i.e., during mutual authentication phase. The number of replay attacks during data exchange is represented by A^+ and B^+. In Table 2, the number of replay attacks during data exchange is launched on packet flows, where each flow may contain several packets. For example, during the first 10 seconds, the intruder A^+ replays 8 flows, i.e., 8 attacks, where each attack contains a series of replayed data packets.

6. Conclusion

In this paper, we have proposed a payload-based authentication scheme to verify the identities of the communicating clients and server in the network. It uses a simple handshake procedure to exchange the session key for the resource observation. The proposed scheme uses pre-shared secrets for the identity verification of objects. These secrets are known only to the legitimate clients and server and they cannot be obtained illicitly in view of the Internet Threat Model. Our experimental results show significant improvement over the existing schemes for the resource-constrained objects. Our proposed scheme is efficient against eavesdropping, DoS, replay and resource exhaustion attacks. However, it is yet to be determined how efficient and robust the scheme is against other types of attacks. Furthermore, the proposed scheme allocates pre-shared keys at the provisioning phase, so it is a ideal choice for static deployment. If the pre-shared key of an incoming mobile client is not present in the server lookup table, the client will not be able to communicate with the server. For this type of scenario, a dynamic key allocation is an optimal choice. This is specifically the case with a large scale IoT environment. The scalability and mobility may further improve the proposed scheme and broaden its scope. The resource-constrained nature of the sensor nodes will be a challenging task in provisioning of the above facilities for such a lightweight authentication scheme.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: A survey on enabling technologies, protocols, and applications, IEEE Communications Surveys & Tutorials 17 (2015) 2347–2376.

[2] A. Botta, W. De Donato, V. Persico, A. Pescapé, Integration of cloud computing and internet of things: a survey, Future Generation Computer Systems 56 (2016) 684–700.

[3] J. Granjal, E. Monteiro, J. S. Silva, Security for the internet of things: a survey of existing protocols and open research issues, IEEE Communications Surveys & Tutorials 17 (2015) 1294–1312.

[4] Z. Shelby, K. Hartke, C. Bormann, The constrained application protocol (coap) (2014).

[5] E. Rescorla, N. Modadugu, Datagram transport layer security version 1.2 (2012).

[6] R. T. Fielding, R. N. Taylor, Principled design of the modern web architecture, ACM Transactions on Internet Technology (TOIT) 2 (2002) 115–150.

[7] J. Kim, J. Lee, J. Kim, J. Yun, M2m service platforms: Survey, issues, and enabling technologies., IEEE Communications Surveys and Tutorials 16 (2014) 61–76.

[8] K. Hartke, Observing resources in the constrained application protocol (coap) (2015).

[9] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, C. K.-Y. Tan, Performance evaluation of mqtt and coap via a common middleware, in: 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP),, IEEE, 2014, pp. 1–6.

[10] P. Saint-Andre, Extensible messaging and presence protocol (xmpp): Address format (2015).

[11] M. Becker, T. Pötsch, K. Kuladinithi, C. Goerg, Deployment of coap in transport logistics, in: Proceedings of the 36th IEEE Conference on Local Computer Networks (LCN), 2011.

[12] S.-C. Son, N.-W. Kim, B.-T. Lee, C. H. Cho, J. W. Chong, A time synchronization technique for coap-based home automation systems, IEEE Transactions on Consumer Electronics 62 (2016) 10–16.

[13] M. Centenaro, L. Vangelista, A. Zanella, M. Zorzi, Long-range communications in unlicensed bands: The rising stars in the iot and smart city scenarios, IEEE Wireless Communications 23 (2016) 60–67.

[14] M. Becker, K. Kuladinithi, C. Görg, Wireless freight supervision using open standards, in: 5th International Workshop on Cold Chain Management. June, 2013, pp. 10–11.

[15] W. Trappe, R. Howard, R. S. Moore, Low-energy security: Limits and opportunities in the internet of things, IEEE Security & Privacy 13 (2015) 14–21.

[16] T. Dierks, The transport layer security (tls) protocol version 1.2 (2008).

[17] K. Hartke, Practical issues with datagram transport layer security in constrained environments draft-hartke-dice-practical-issues-00, IETF work in progress (2013).

[18] S. Raza, L. Seitz, D. Sitenkov, G. Selander, S3k: Scalable security with symmetric keys—dtls key establishment for the internet of things, IEEE Transactions on Automation Science and Engineering 13 (2016) 1270–1280.

[19] A. Bhattacharyya, A. Ukil, T. Bose, A. Pal, Lightweight mutual authentication for coap (wip), draft-bhattacharyya-core-coap-lite-auth-00 (2014).

[20] J. Granjal, E. Monteiro, J. Sa Silva, On the feasibility of secure application-layer communications on the web of things, in: 2012 IEEE 37th Conference on Local Computer Networks (LCN), IEEE, 2012, pp. 228–231.

[21] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, G. Carle, Dtls based security and two-way authentication for the internet of things, Ad Hoc Networks 11 (2013) 2710–2723.

[22] J. Granjal, E. Monteiro, J. S. Silva, On the effectiveness of end-to-end security for internet-integrated sensing applications, in: 2012 IEEE International Conference on Green Computing and Communications (GreenCom), IEEE, 2012, pp. 87–93.

[23] D. Trabalza, S. Raza, T. Voigt, Indigo: Secure coap for smartphones, in: Wireless Sensor Networks for Developing Countries, Springer, 2013, pp. 108–119.

[24] M. A. Jan, P. Nanda, X. He, Z. Tan, R. P. Liu, A robust authentication scheme for observing resources in the internet of things environment, in: 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE, 2014, pp. 205–211.

[25] M. C. Domingo, An overview of the internet of underwater things, Journal of Network and Computer Applications 35 (2012) 1879–1890.

[26] D. R. Raymond, S. F. Midkiff, Denial-of-service in wireless sensor networks: Attacks and defenses, Pervasive Computing, IEEE 7 (2008) 74–81.

[27] K. Pelechrinis, M. Iliofotou, S. V. Krishnamurthy, Denial of service attacks in wireless networks: The case of jammers, Communications Surveys & Tutorials, IEEE 13 (2011) 245–257.

[28] H. Liu, P. Luo, D. Wang, A distributed expansible authentication model based on kerberos, Journal of Network and Computer Applications 31 (2008) 472–486.

[29] L. Seitz, G. Selander, Design considerations for security protocols in constrained environments, draft-seitz-ace-design-considerations-00 (WiP), IETF (2014).

[30] S. K. Sood, A. K. Sarje, K. Singh, A secure dynamic identity based authentication protocol for multi-server architecture, Journal of Network and Computer Applications 34 (2011) 609–618.

[31] P. Chown, Advanced encryption standard (aes) ciphersuites for transport layer security (tls) (2002).

[32] E. Rescorla, B. Korver, Guidelines for writing rfc text on security considerations (2003).

[33] R. Elz, Serial number arithmetic (1996).

[34] J. Arkko, A. Keranen, Coap security architecture, draft-arkko-core-security-arch-00 (work in progress) Internet Draft (2011).

[35] K. Kuladinithi, O. Bergmann, T. Pötsch, M. Becker, C. Görg, Implementation of coap and its application in transport logistics, Proc. IP+ SN, Chicago, IL, USA (2011).

[36] A. Ludovici, P. Moreno, A. Calveras, Tinycoap: a novel constrained application protocol (coap) implementation for embedding restful web services in wireless sensor networks based on tinyos, Journal of Sensor and Actuator Networks 2 (2013) 288–315.