A SECURITY ARCHITECTURE FOR THE INTERNET OF THINGS

Reinhard Behrens[1], and Ali Ahmed[2]
[1] Liverpool University / Laureate Education, UK
[e-mail: Reinhard.behrens@online.liverpool.ac.uk]
[2] Department of Computer Science, Cairo University
Giza, Egypt
[e-mail: a.ahmed@fci-cu.edu.eg]
*Corresponding author: Ali Ahmed

Received September 27, 2016; revised February 20, 2017; accepted July 13, 2017; published December 31, 2017

Abstract

This paper demonstrates a case for an end-to-end pure Application Security Layer for reliable and confidential communications within an Internet of Things (IoT) constrained environment. To provide a secure key exchange and to setup a secure data connection, Transport Layer Security (TLS) is used, which provides native protection against replay attacks. TLS along with digital signature can be used to achieve non-repudiation within app-to-app communications. This paper studies the use of TLS over the JavaScript Object Notation (JSON) via a The Constrained Application Protocol (CoAP) RESTful service to verify the hypothesis that in this way one can provide end-to-end communication flexibility and potentially retain identity information for repudiation. As a proof of concept, a prototype has been developed to simulate an IoT software client with the capability of hosting a CoAP RESTful service. The prototype studies data requests via a network client establishing a TLS over JSON session using a hosted CoAP RESTful service. To prove reputability and integrity of TLS JSON messages, JSON messages was intercepted and verified against simulated MITM attacks. The experimental results confirm that TLS over JSON works as hypothesised.

Keywords: Internet of Things, privacy, SSL/TLS, CoAP, JSON, Application Layer Security, Constrained Application Protocol Restful Environments, Key Agreement, Key Exchange, Non-Repudiation

1. Introduction

The Internet of Things (IoT) market is growing immensely [9]. Per Gartner, “In 2020, 25 Billion Connected "Things" will be in use”. Such a modern computing environment imposes new challenges that may not have been seen in traditional computing environments. For example, using traditional security measures such as typical Secure Sockets Layer (SSL) to protect connections might not be enough [32]. One can see that in the latest Payment Card Industry **Data Security** Standard (PCI DSS) compliance standard, SSL cryptography is no longer adequate [26]. Indeed, an alternative protocol should be considered. As the time of writing this paper, TLS is the better alternative [25]. IoT in low power consumption sensor environments does not use TCP/IP, but rather User Datagram Protocol (UDP) [33]. Normal TLS works over a network stream which tracks state. Using UDP, requires a different approach. To secure the UDP connection a derivative called the Datagram Transport Layer Security (DTLS) is used for IoT connections [34]. Comprehensive security in these environments are a major challenge due to low resource availability. DTLS provides point-to-point security, which could expose sensor data (or private user information) during a decryption operation between endpoints occurs. In addition, DTLS uses stateless encryption, thus no steam ciphers [18]. Fragmentation at the UDP level uses more device resources [35], which impacts on IoT devices. Lastly, reemission is more complex in DTLS. Based on the latter, this paper envisages a solution that uses an application layer to secure IoT communications. The challenge in such a proposal is that the solution needs to provide the means, the format and the design of how an application layer security can be used in different system architectural configurations using various IoT devices and providers. The research carried out in this paper attempts to provide a practical approach that leverages existing technologies within its design. IoT devices needs the capability to send sensor data to online systems using the existing web infrastructures and protocols. An example of such protocols is TLS, which is used for security and integrity.

An application layer security solution for IoT communications is proposed. The proposed system utilises CoAP [39] and JSON [38] technologies. JSON is preferred to the Concise Binary Object Representation (CBOR) format, due to its smaller footprint as JSON lacks data types. JSON is, thus, smaller in size when transmitted over the wire. The average IoT device, such as a light bulb, runs a 48MHz processor and has up to 32KB of SRAM. This leaves little room for error and requires optimised programming and resource utilization. To test the theory if TLS over JSON is plausible, the following output is needed:

1. Establish if an IoT device can hold such a message in memory,
2. Compute/Estimate the Central Processing Unit (CPU) overhead, and
3. Determine the required Storage space for the program at rest and at runtime.

This research tries to address the following questions:

- Why use TLS over JSON in RESTful, when DTLS is available?
- Are there any security advantages to using layer 7 security?
- Does using this method benefit IoT restrictive environments?
- Will TLS behave “naturally” over CoAP/RE with large messages?

The organisation of this paper is as follows:

1. Section 2 discusses the challenges and the need for a new way of securing the emergent IoT environment.
2. Section 3 introduces the architectural design of the proposed system. It also highlights the prototype and its testbed.
3. Section 4 verifies the new system by testing cases to evaluate the research hypothesis.
4. Section 5 concludes the paper and lists future work.

2. Challenges and Opportunities

The original internet has only taken off in the 1990's [36]. This means that technology has matured very rapidly leading to the use of outdated standards. For example, the HyperText Markup Language (HTML) used for formatting web pages was only developed in 1995 [37]. It means that the HTML standard has not even been around for one generation at the date of writing this paper. This newly born internet had little or no security. Millions of dollars were spent since on anti-virus programs, software patches, and system downtime. To use an enhanced IP stack, IPv6 was developed. The dilemma is that TCP/IPv6 was developed "over a decade ago" in the late 1990's [21]. With the delay in implementing IPv6 to market, created a further problem. By the time IPv6 reached the market, the standard was outdated. Clearly, its developers were unable to foresee the level of security threats of today. The industry is using an outdated RFC standard to address modern security threats. Every networked system today, whether this is a mobile phone, internet server or a desktop computer, uses TCP/IP as its primary network communication stack.

The current infrastructure is suffering large losses of money due to vulnerabilities of existing systems [20]. A better way of securing the entire communication stack might solve the problem. This requires learning from the previous design mistakes that plagued the internet since its conception. As the research clearly shows, older security models are no longer strong enough "as is" and contain very little security on the "inside". Thus, for each protocol, security measures are an afterthought which is the case of IPSec and SSL. Furthermore, the SSL market is at its weakest point, posing risks to businesses in the global e-commerce market [5]. The world is now approaching a new era where technology use will only exponentially increase. The 'old way' of approaching security might no longer be adequate.

As the internet is now considered a society, legislation will be required at some point soon [11]. This might be needed to regulate and record offences flowing on the judicial system and the WWW, which is to be presented in the court of law. This is a problem in the modern digital era, where digital evidence has very little weight; or require a rigorous vetting process to establish authenticity [12]. At present, in my own experience, most judges will not likely consider digital evidence in a courtroom; due to it being a "grey" area. But, if a legally binding digital solution exists for authentication on digital wiretaps, then this long-held objection against digital evidence might change. TLS over JSON might provide one way of providing key material to the target devices to digitally sign data using applications. This could be applicable to web apps as well as IoT devices. Using app-to-app transfers using TLS with mutual trust, digital contracts could naturally evolve using digital cryptography. For example, modern cloud computing system complex to secure and have many management and compliance requirements [13]. Adding security and integrity on individual application calls could help protect data calls.

Modern system architectures are shifting from a client-server computing model to grid cloud computing [2]. Traditional technologies for securing systems might not be able to adapt

to these ever-changing system architectures. This means, that the entire concept of how computer systems are designed is rapidly changing. For example; point-to-point communication channels suffer from limitations when transmitting data across multi-tier architectures. This includes TLS/SSL offloading scenarios. By moving the TLS protocol into the OSI Application Layer, allows the transmission of data end to end, bypassing almost all offloading scenarios.

Using TLS over HTTP or RESTful on the service level could be beneficial. It can provide an alternative to establishing confidential links. ALS TLS is very lightweight and proposes an easy transition for mainframes as well as existing web technologies and high-end technologies to converge securely. Especially as network security pertains to grid computing. ALS TLS is not a replacement for NSI, for example, in grid computing and related technologies; but simply a way to establish pure end-to-end security. The theory for using ALS TLS protocol is further strengthened through the introduction of JSON. JSON is cross platform and service independent [24]. This means that JSON data is not bound to an HTTP SOAP service standard. This concept allows for a loosely coupled message process, allowing for cross platform and cross device communication channels.

Layer 7 (L7) firewalls have greatly enhanced overall security on application layer requests [4]. L7 technology incurs a heavy processing cost during its packet analysis phase [14]. Layer 7 firewalls is still a solution derived from a bottom-up view, due to its OSI nature. This means that security designers are patching/analysing from the physical layer upwards. Never actually solving the problem, but rather responding to new threats, building on an unstable foundation. Using this approach is reactive like most modern systems [15]. By implementing layer 7 security from a top-down approach changes how the problem is addressed. Using ALS L7 virtual channels, each application request has enforced integrity and confidentiality since each transmission is on a securely established channel via CoAP. This approach, potentially problematic, negates the high-security measures at layer 7. The data is now only decrypted above layer 7. Although the data still needs to be inspected, it becomes an application layer problem, keeping the networking stack clear. This approach creates a flat model, where most data encryption ad inspection is bubbled to the top.

IoT by its very definition requires sensor devices, mobile phones, household appliances and cloud services to communicate with one another and the cloud [23]. IoT Security has not been rolled out due to its poor security implementation and lack of established standards [10]. This is largely due to the increase of connected devices to the WWW and lack of proper security mechanisms. To ensure that we define the industry's existing state of security pertaining to IoT we will include research from various bodies of knowledge such as IEEE, IETF, and ACM. The design this paper presents is IoT application layer security using TLS over JSON. If the concept proves to be feasible, then TLS will provide the following security benefits to ALS:

- Mutual Trust
    - Data Transmission Integrity [7, 3]. Integrity is built-in, not optional
- Compulsory Confidentiality. Encryption is not optional.
- Potential Long-term data archiving of each individual data request. Allows for long-term archiving of all data packages sent, as each is individually encrypted and digitally signed.
- Hop-by-Hop issues. No communication rewriting of original payload, or data decryption as its pure app-to-app

- Interoperability between server, mobile, desktop and smaller IoT devices. Standards-based data formats allows interconnectivity between devices
- Device communication segmentation. Accomplished using different sessions to various providers

Ultimate protection for IoT remains a topic for debate and research. The current research currently shows how to improve the IoT communication security posture and how to outsource cryptographic tasks using an IoT gateway proxy [6] and even implement DTLS. When one considers outsourcing processing tasks of cryptographic keys, it risks exposing the keys themselves. Using SSL Offloading has one such privacy concern [1]. An attacker could potentially intercept key material from the key generation proxy. Similarly, serious concerns arise when CoAP-enabled devices communicate outside its perimeter (i.e. such as a house or grid). The mutual trust could resolve point-to-point problems, however, routing and switching still allow data to be decrypted at potential untrusted end-points. Keeping in mind that sensor devices should not be connected directly to the internet [16]. Gateway devices already introduce privacy concerns due to SSL offloading or decryption. This is due to the nature of the IP stack and UDP sockets. UDP is converted to TCP, for reliable communication across the internet. There is an opportunity for another way to securely communicate to remote Cloud computers and computer Grids directly from the source. Thus, IoT application-to-application connectivity needs to be considered.

UDP is a transport layer protocol, and stateless [27]. UDP does not represent a stream of data (stateless) and requires a special implementation of TLS called DTLS. However, none of these research topic uses a top-down approach to implement security. This means pure application level implementation of TLS. The primary focus of this paper will, thus, be to test CoAP using TLS over JSON to other external RESTful services. The design will attempt to be inclusive of both small IoT devices, such as an IoT lightbulb (running an Atmel SAM R21 48MHz chip with 8KB RAM) as well as larger devices such as a Laptop or Server. Most tests will be emulated for the sake of proving the hypothesis, so no actual hardware tests will be conducted. An RSA key pair can be created by an IoT light bulb (running at 48MHz) in approximately 5 secs [8]. This is the worst case, as ECC provides better performance and fewer CPU cycles to calculate. Asymmetric keys, on the other hand, would only be necessary once for normal certificate enrolment scenarios and protecting symmetric key material. Cryptography does remain an ongoing resource utilisation problem, even for DTLS scenarios. ECC as an encryption scheme exponentially outperforms its RSA counterpart and should be considered instead. By using TLS over JSON on the CoAP layer, one could potentially reduce complexities found in DTLS. This is from a design perspective, not cryptographically. Disabling DTLS in certain communication scenarios could prove ideal. Pure TLS over JSON will include all the required security services. The aim is not to replace DTLS, but simply to do a study using TLS over JSON to consider pure application level security. This research attempts to unify communications between IoT sensor devices, mobile applications, and cloud providers while not depending on any lower level security (such as DTLS) and without the use of gateway services.

The requirement for an application security layer is initiated for the need to secure cloud and IoT device communication [22]. The implementation should be ideally built to protect layer 7 against attack. The theory is that existing IoT devices should be able to establish TLS over JSON with other networks. The question that may arise here is, “why not use DTLS?” [28]. DTLS is UDP based and requires complex gateway systems to help with key management and encryption [29]. This may not be the best use case for high-security environments. By using only the application layer security (TLS over JSON), the system can

establish a secure connection directly to the outside world. For CoAP services, a UDP-to-TCP gateway will suffice. UDP data (encrypted at the application level) can be transmitted in clear text over TCP to its intended recipient. This greatly reduces costs and complexities by not installing additional equipment or services. As this approach is new, this research tries to ensure that the TLS over JSON message structure can be implemented in an open standards approach. The work done by Xu et al [40] is an interesting one. In this work, the researchers dealt with the problem of randomly distributed eavesdroppers by secure relay communications in IoT networks. The optimal power allocation and code word rates were derived in this research for both a specific case study and in a generic one where eavesdroppers are equipped with multiple antennas. The research suggests the use of relay transmission enhance the secrecy throughput and extend the secure coverage area. The work focuses on 5G networks while our hypothesis focus on an application layer solution that could fit any IoT restricted device. The focal point of that work is theoretical and does not provide a means of implementing the solution in on a restrictive IoT device. This is the stark difference between our research line and the work done by Xu et al [40].

The main contribution of this research is the proof that using TLS in the application tier solves basic by-hop problems currently present in TLS/DTLS. In fact, the expected outcome, irrespective of lower level communication protocols, would ideally lead to findings such as, better end-to-end communications for the web. These include the use of application services, security, and protection of privacy. A unique benefit is the reuse of key material for a given application session indeed. The solution demonstrated in this paper provides a way for systems to interconnect securely using TLS in an unusual way. TLS (Transport Layer Security) becomes ALS (Application Layer Security). This paper describes how open standards such as JSON and CoAP is used to accomplish the transport of such a system.

3. The Architectural Design

The precept of moving TLS data over JSON is illustrated in **Fig. 1**. The figure explains how the SSL/TLS layer should be moved from its present placement within the networking stack upwards. Thus, the TLS stack should be configured to communicate over HTTP and RESTful services [30]. To have a stronger security credential, one should not rely on lower layers. This way, at least key negotiation is done securely to provide a means to secure a variety of new and existing web applications. On a mobile or IoT device, the application layer programmer can initiate the TLS call. Using the on-board RSA key generator or software keys, the same result can be accomplished as in our tests. The TLS record is simply transformed from a computer code object to a JSON formatted object.

This change does not, by any means, require the amendment of any existing protocols or network designs. This merely requires the change of thinking on how to apply TLS within IoT design. Instead of encapsulating the data by feeding it to the socket and intercepting this stream via TLS, another approach is needed. Due to the point-to-point limitation of using sockets, perceiving the solution from the top-down, one rather encapsulates the data at the source. App-to-app security is realised using existing and proved key exchange protocols. Furthermore, TLS is designed to be abstracted, as a network stream could easily be exchanged for HTTP web stream or as a communication stream. Hereby, the full extent of the TLS and its intended design is fully utilised.

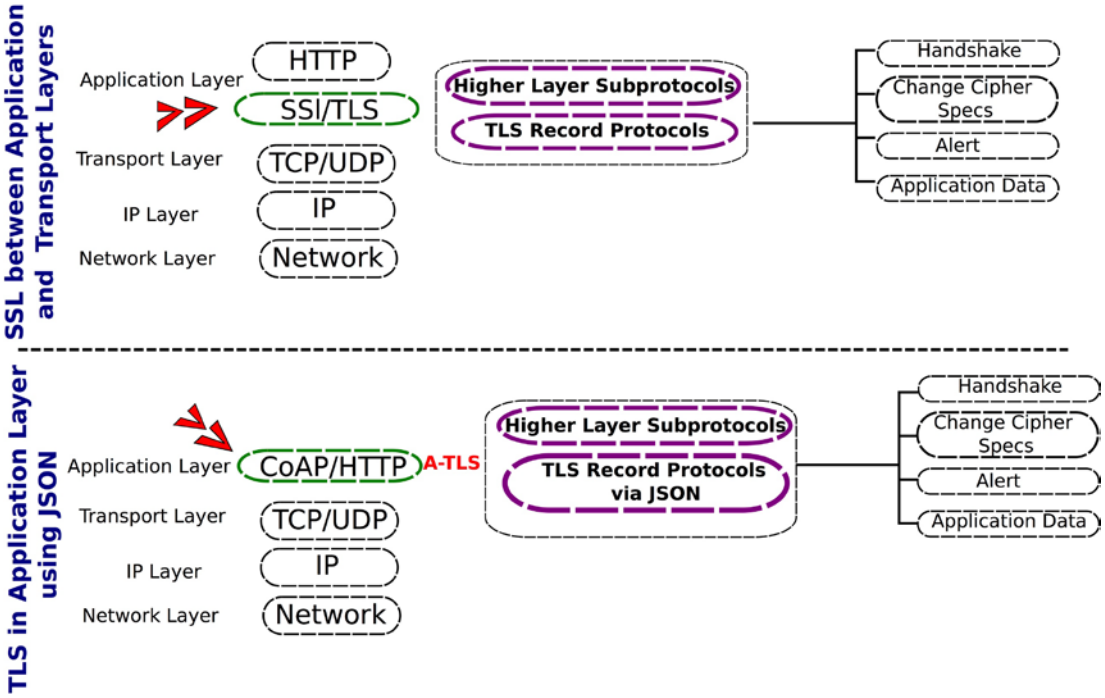

Fig. 1. OSI Placement of A-TLS Vs TLS

TLS mainly communicates data over the wire using TLS records. Once TCP/IP has finished its 3-way handshake and a network stream becomes available, TLS begins its negotiation [31]. Each TLS communication block is defined as a record. Each Record, as seen in **Fig. 2**, has a type to indicate to the recipient what the call relates to. For TLS, this is defined as type Handshake, ChangeCipherSpec, Data or Alert. The TLS protocol is less complex than its SSL ancestor. After analysing the TLS protocol stack, the flexibility of TLS became apparent.

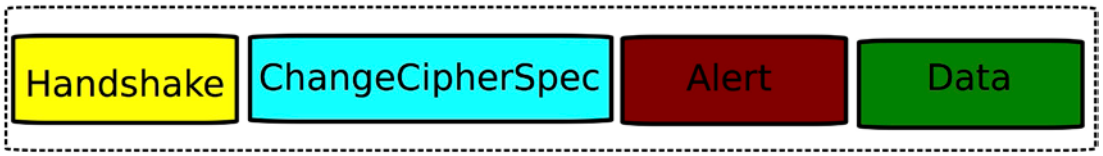

Fig. 2. Basic TLS Record

To translate the TLS communication records, one must ensure the protocol data integrity within the communication flow. The test cases that will be introduced later in this paper will serialise the record structure to JSON and de-serialise the structure into an object. The fragments that are transmitted with each TLS record contains the content type within each part or the system. This is used during the session negotiation phase as seen in **Fig. 3**.

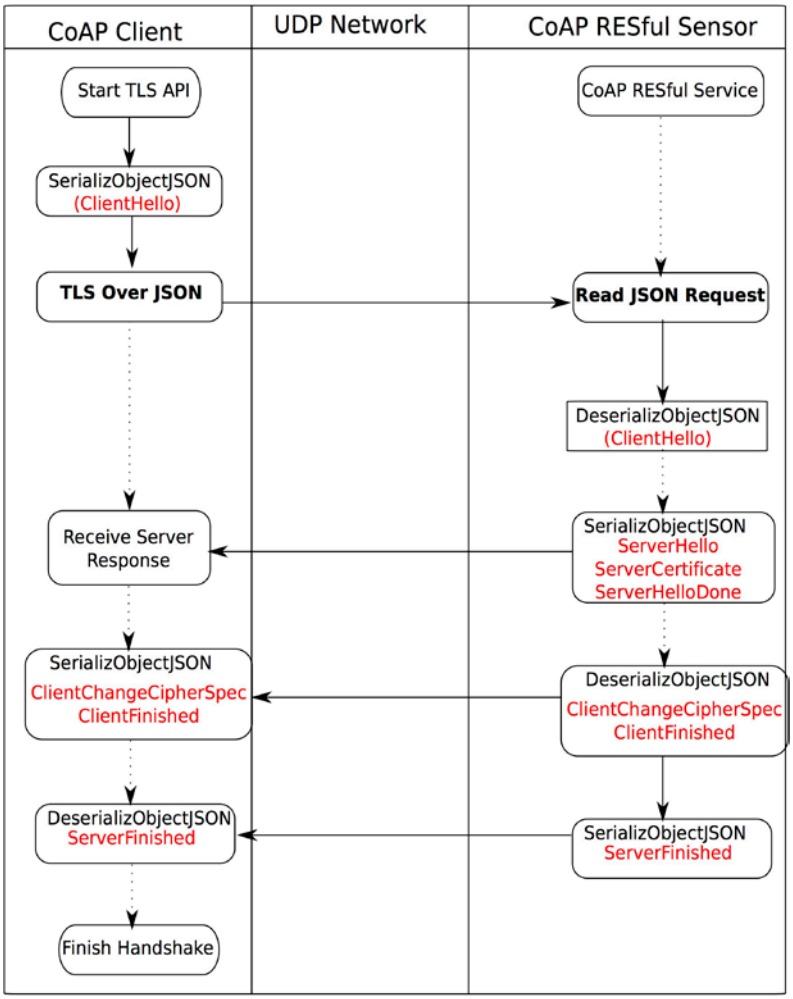

Fig. 3. TLS Session Handshake using JSON

To realise this vision and as a proof-of-concept, a prototype was developed. The testbed of the implementation is using an Intel Core i7-3770 with 16GB of RAM and a 256GB SSD drive for storage. The operating system is Windows 10-64bit. All IoT software is run on an IoT device emulator. To test the hypothesis in practice, a Visual Studio project containing a client TLS system was setup. Subsequently, a Visual Studio server project was created to listen for any TLS client calls. To refer to the normal TLS client setup, we will refer to this as TLS-Client-A and the TLS server as TLS-Server-A. These services use the default TLS communication process, by sending records using a normal TLS socket connection over TCP/IP. A test if the TLS 1.2 connection works outside the lab environment, a single test will be done to a third-party web server. Once the TLS communications stack works as expected, adjustments will be made to test the message implementation for IoT devices. The TLS stack will be amended to work asynchronously over CoAP.

The first test will be to keep the client server using TLS-Client-A and TLS-Server-A using default implementations. During a TLS session setup, the TLS Record will be serialized into JSON in real-time. This transformed TLS Record, now encoded within a JSON structure, will be sent to TLS-Server-A via CoAP. Part of the implementation is to keep message sizes below 1024 bytes, due to the CoAP Payload size. If the TLS handshake is completed correctly, then the serialization process is a success. At each transmission of a TLS record, the JSON TLS record version will be exported to a file. This will allow for an analysis of each record, and identification of the content type of each after transmission.

The content of these captured packets will also be used in a simple hard-coded CoAP server and CoAP client. The test is to see whether these data structures can be serialized and de-serialized within an IoT environment. The CoAP Server will be referred to as CoAP-Server-Sensor and the client as CoAP-Client.

The output of the project is to show that the empirical results should show that the CoAP RESTful Service will receive the JSON data, and successfully de-serialize the TLS record into a binary object. The TLS handshake data will be extracted from the TLS data fragment received. At this stage, a conclusion will be made if the CoAP A-TLS using JSON is practical and should be perused. The primary programming language that will be used is C#. C# and the libraries used are mostly compatible with PC and IoT frameworks.

Primary libraries used in the testing process:

- https://github.com/mweimer/Json.NetMF. A JSON library for .Net Micro Framework
- https://github.com/juhovh/AaltoTLS. An Open Source TLS source written in C#
- http://www.bouncycastle.org/csharp/. A cryptographic library to support TLS client communication.

The results are measured by primarily focusing on the behaviour of the TLS Record exchange. Each record is captured by writing its JSON equivalent to a log file. The JSON file contains a full TLS record as transmitted by either the client or the server. During the implementation, a problem is expected related to the serialisation of the JSON object due to a ProtocolVersion struct in the AaltoTLS project. A custom JSON formatter is required to deal with this problem. In the IoT device emulation standalone test the PreviousProtocolVersion iterations is to be removed from the TLS message structure to reduce CoAP payload size. The design, in general, remains the same.

The implementation model as depicted in Figure 4 represents the complete end-to-end test. There is, however, a foundational preparation that needs to be done to first obtain a working TLS process.

Fig. 4. The Basic System Model for TLS over CoAP

As seen in **Fig. 5**, the records are sent for each part of the handshake. We need to capture these three core message structures. The organization of the tests will be a simple client and server. Because we need access to an underlying TLS library, each Record call will be able to be serialized to JSON. In addition, based on the TLS record structure, one can derive the required object code for the software as shown in **Fig. 6**.

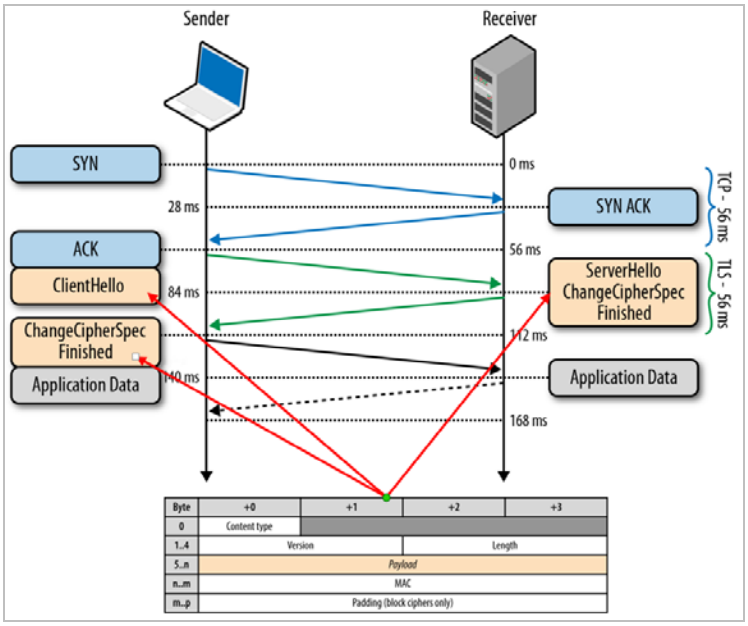

Fig. 5. SSL connection, and capture of TLS packets

Byte	+0	+1	+2	+3
0	Content type			
1..4	Version		Length	
5..n	*Payload*			
n..m	MAC			
m..p	Padding (block ciphers only)			

Fig. 6. TLS record structure

In **Fig. 7**, an example of a TLS record structure is defined. The first byte in our example holds the content type for the TLS record. The TLS version can also be identified (For example TLS 1.2 which is denoted by v3 major version and v3 minor version) in the second and third byte of the record. This is not an extensive investigation of TLS itself, but rather to display our test design can revalidate the handshake message sent via our adjusted TLS stack.

```
const unsigned char TLS_output[] = {
   // TLS record
   0x16, // Content Type: Handshake
   0x03, 0x01, // Version: TLS 1.0
   0x00, 0x6c, // Length (use for bounds checking)
      // Handshake
      0x01, // Handshake Type: Client Hello
      0x00, 0x00, 0x68, // Length (use for bounds checking)
      0x03, 0x03, // Version: TLS 1.2
      // Random (32 bytes fixed length)
      0xb6, 0xb2, 0x6a, 0xfb, 0x55, 0x5e, 0x03, 0xd5,
      0x65, 0xa3, 0x6a, 0xf0, 0x5e, 0xa5, 0x43, 0x02,
      0x93, 0xb9, 0x59, 0xa7, 0x54, 0xc3, 0xdd, 0x78,
      0x57, 0x58, 0x34, 0xc5, 0x82, 0xfd, 0x53, 0xd1,
      0x00, // Session ID Length (skip past this much)
      0x00, 0x04, // Cipher Suites Length (skip past this much)
         0x00, 0x01, // NULL-MD5
         0x00, 0xff, // RENEGOTIATION INFO SCSV
      0x01, // Compression Methods Length (skip past this much)
         0x00, // NULL
      0x00, 0x3b, // Extensions Length (use for bounds checking)
         // Extension
         0x00, 0x00, // Extension Type: Server Name (check extension type)
         0x00, 0x0e, // Length (use for bounds checking)
         0x00, 0x0c, // Server Name Indication Length
            0x00, // Server Name Type: host_name (check server name type)
            0x00, 0x09, // Length (length of your data)
            // "localhost" (data your after)
            0x6c, 0x6f, 0x63, 0x61, 0x6c, 0x68, 0x6f, 0x73, 0x74,
         // Extension
         0x00, 0x0d, // Extension Type: Signature Algorithms (check extension type)
         0x00, 0x20, // Length (skip past since this is the wrong extension)
         // Data
         0x00, 0x1e, 0x06, 0x01, 0x06, 0x02, 0x06, 0x03,
         0x05, 0x01, 0x05, 0x02, 0x05, 0x03, 0x04, 0x01,
         0x04, 0x02, 0x04, 0x03, 0x03, 0x01, 0x03, 0x02,
         0x03, 0x03, 0x02, 0x01, 0x02, 0x02, 0x02, 0x03,
         // Extension
         0x00, 0x0f, // Extension Type: Heart Beat (check extension type)
         0x00, 0x01, // Length (skip past since this is the wrong extension)
         0x01 // Mode: Peer allows to send requests
};
```

Fig. 7. Sample TLS record

4. Experimental Results and Analysis

4.1 Test Case 1 – TLS transposed as JSON baseline

The aim of this test case is to verify that a TLS record can be submitted over UDP using CoAP. The size limitation for this test is a concern, as each CoAP payload is limited to 1024 bytes. This means as soon as a record reaches a size over 1024 bytes, the Record is broken into 1-n records. In other words, this test verifies that the basic hypothesis of the handshake message itself is possible.

Data is collected from the ClientHello and ServerHelloDone states of the TLS protocol handshake. As Figures 8 shows, both ClientHello and ServerHello messages are successful. The result for the ServerDone is a total size of 2340 bytes and the client message a total size of 299 bytes. The message has a roundtrip value of 7 seconds for the ServerDone message and 1 second for the ClientHello message. Time measurement is done on the file timestamp created by the software during the time of transmission and reception. This action can be accomplished using the CoAPTLS-HostA client software. Each part of the TLS transaction is managed using a key input control.

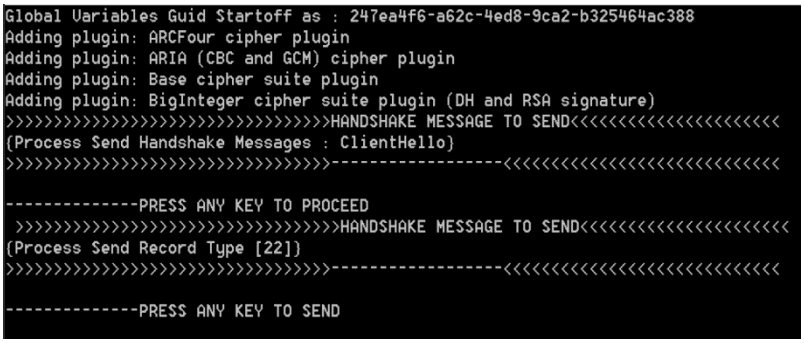

Fig. 8. Successful ClientHello Message

Fig. 9 shows the output of a CoAP response from the server to acknowledge the receipt. The benefit here with the block transfer mechanism is that the delivery is not guaranteed over UDP, where pure UDP would not have provided any value.

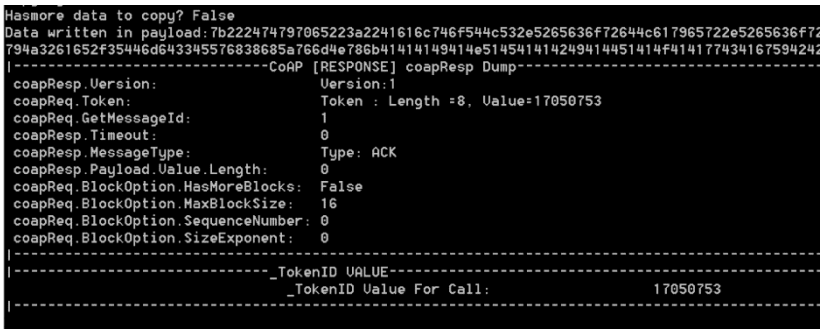

Fig. 9. CoAP Server Response

The server side software CoAPTLS-HostB lists the necessary responses. This software needs to be run with CoAPTLS-HostA client side software. The CoAP ACK captured into a dmp file with the unique CoAP 8-byte token ID 17050753 as shown in Figure 10. In that figure, the server receives the ClientHello message and prepares ServerHello, Certificate and ServerHelloDone. **Fig. 11** shows the CoAP Content (.CON) of the three CoAP messages using the PUT method of each block of the split ServerHelloDone Record response. In fact, the output here was larger than 1024 bytes and a screenshot of the section of interest is seen in **Fig. 12**. The Base64 representation of the Record Fragment Data has an actual byte size of 1087 during transmission. The actual JSON text measures at 2340KB, which means that three Block transfers are required and was observed during the initial transmission. Based on that, the handshaking is proved to be successful validating the partial hypothesis that handshake message is possible for the TLS record using JSON baseline.

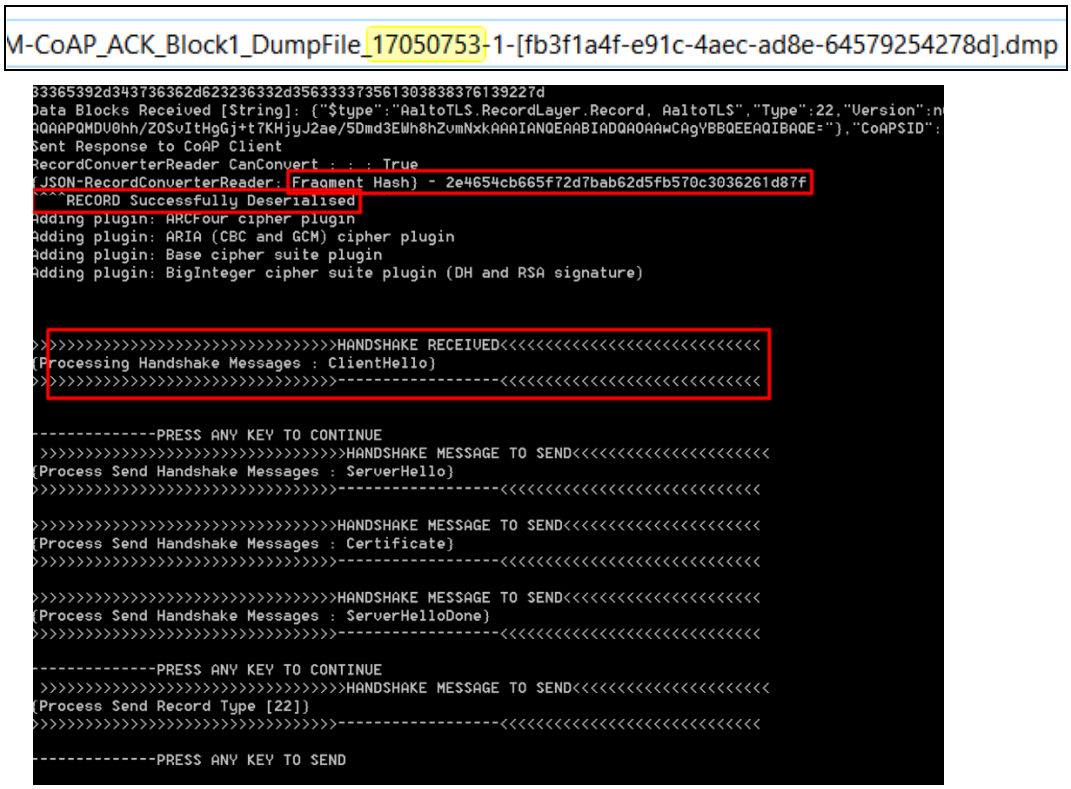

Fig. 10. CoAP Token ID

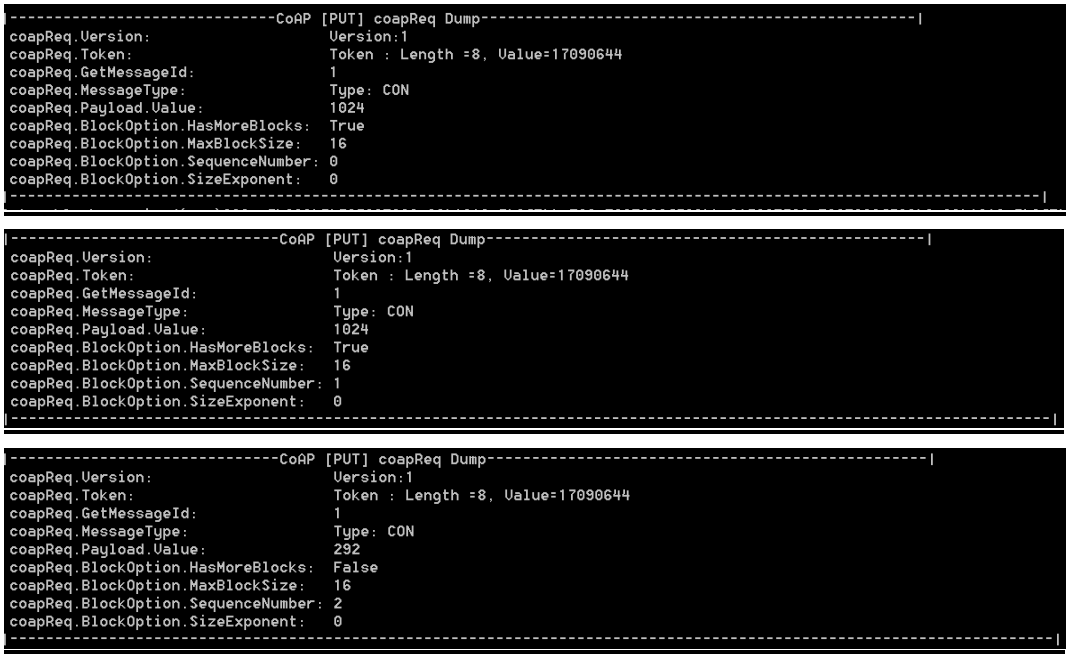

Fig. 11. CoAP Content (.CON)

```
,"HasSelectablePRF":false,"HasSelectableSighash":false,"HasExtensions":true
,"HasExplicitIV":false,"HasVariablePadding":true,"HasVerifiablePadding":true
,"HasAeadSupport":false}]}]}],"Epoch":0,"SequenceNumber":0,"Fragment"
:{"$type":"System.Byte[], mscorlib","$value":"AgAAJgMDV0hnSMq9nzG+Sd45gkGB4iE
1KMTlRSSThOPe07bIAJwAADUACwAEDQAECgAEBzCCBAMwggLroAMCAQICCQCGbl0qppBm7TANBgkq
hkiG9w0BAQUFADBeMQswCQYDVQQGEwJGSTEQMA4GA1UECBMHVXVzaW1hYTEOMAwGA1UEBxMFRXNwb
28xGTAXBgNVBAoTEEFhbHRvIFVuaXZlcnNpdHkxEjAQBgNVBAMTCWxvY2FsaG9zdDAeFw0xMTA5MT
YxNDMwMzZaFw0xMjA5MTUxNDMwMzZaMF4xCzAJBgNVBAYTAkZJMRAwDgYDVQQIEwdVdXNpbWFhMQ4
wDAYDVQQHEwVFc3BvbzEZMBcGA1UEChMQQWFsdG8gVW5pdmVyc2l0eTESMBAGA1UEAxMJbG9jYWxo
b3N0MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr88L3GXh/vChLZ5FdhcOV4BtHmS8F
ZpvG1MnPEQK4uROzoUW2LZtgehOzdGRZdJzV/8hkpiliR3Ww9/VYVXqZcpPgh8Q01aMjvDVx3kQc5
QaXNjBEwtijtgMQr8GZecpq2bpj+ovCrYFI69xtCc3HxIvCH7Ywkk597bZwCdbJdFQb+EHPlU9tO4
Z4jUDtAKh+A8NNuLorvgCrTIOxo7GeD1lvmODVN7anRHlrld0eR+iSuBdhsXQGzZsJDFuaUob
/lkahZkrrbG91USlcFOou/xky97HxWd4E5Md+NBeSh47tIBvDg0JfPLnHjSxPOxZ0tlHgu1TyVo1C
cDVeFW/0QIDAQABo4HDMIHAMB0GA1UdDgQWBBTQFUgn6sQIdi8WtF70sUgayFNMVzCBkAYDVR0jBI
GIMIGFgBTQFUgn6sQIdi8WtF70sUgayFNMV6FipGAwXjELMAkGA1UEBhMCRkkxEDAOBgNVBAgTB1V
1c2ltYWExDjAMBgNVBAcTBUVzcG9vMRkwFwYDVQQKExBBYWx0byBVbml2ZXJzaXR5MRIwEAYDVQQD
Ewlsb2NhbGhvc3SCCQCGbl0qppBm7TAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBBQUAA4IBAQCDO
TIrh36ClY2mj/49LHsHG4p9A01Q+s1TZR8rZ2vXtHMaSYKRMCWQ24tHxwoteQEZ4Yy1tdp
/zAgKgKvQGIJSH8uaKipD/Wu/nWkFGR7edb6Cx1+VAHfiqmjseRhs5uu3PPhT0fLNzvSsDBBVSd71
w5+NEkrLOGDwQra5sxotqwoEDOFcmqe2cSzduKhsxX6pJzNJ3NX0JgWZp2A3uCrKpe+RBRb9vR3BG
0oTgWWkm0qGX5ts0/ciTCGfkosPZ8ko/dfV33JKUQNWhig11hxDHsFFbjsMFxE5AIh78Qa9e34NWY
XCuQZgScjbRijkMr6eADhBVR5DeSs92qcCJJqgDgAAAA=="},"CoAPSID":null}
```


Fig. 12. The Request Output

4.2 Test Case 2 – Establish TLS Secure Session over CoAP

In this test case, the aim is to establish a secure TLS session above layer 7 over UDP and CoAP using a JSON as the application layer format. It is worth noting that if the Record Fragments' MAC's cannot be validated, the session will not work. Thus, the key material exchanged to establish a secure connection is built on cryptographic principles, which if not followed correctly this test will fail. The test will attempt to establish a TLS secure session and to provide the output dump of the master secret generated within the session. The software used to create the tests are CoAPTLS-HostA (Client) and CoAPTLS-HostB (Server). By running the main executable of each, the client software will start with the call sequence to the server software. Each part of the TLS call sequence has a breaking point, which can be continued using the space key. The process can be repeated multiple times.

The TLS session is started using ClientHello, and its reception of the TLS JSON record is received by the target TLS RESTful Server as seen in **Fig. 13**. The acknowledgement of the packet was received as well as the ServerHello content. This proved to be adequate for the initial request. The service acknowledged all packets as seen in **Fig. 14**. One can observe a stateful communication over UDP and CoAP Block Transfer. Each transfer has an ACK packet, indicating that the data is delivered correctly, thus acting as a guaranteed service (i.e. for reliability purpose). In **Fig. 14**, the screenshot on the right, lists the output of the TLS record transacted in JSON format. This allowed for a careful study to ensure that the record was properly constructed. This is important from a design perspective, TLS needs a form of confirmation on each message delivery, otherwise, the handshake will fail, as each message in the process needs to be received in the correct order. The CoAP's design provides a client channel with a delivery service, which makes the implementation possible. Once the finished call is received on the server side, the Cipher Specification is received and the finished call compares the session's handshake messages. In this way, the data is validated. The MASTER SECRET is calculated, and the Handshake is successful as we can see in Table 1 below. The table shows all 48-bytes server master secrets generated during the experiment and the 48-bytes master secrets generated by the client after KDF. The values of the keys are the same confirming that the handshake process is feasible and successful. The server secret and client secret is calculated during the TLS handshake. To ensure the TLS process is working correctly, the server and master secret was extracted. The hex values represent a 48byte value,

represented in a hex data string. We can see that the hex values match up, which means that the TLS session establishment process is accurately working to this point.

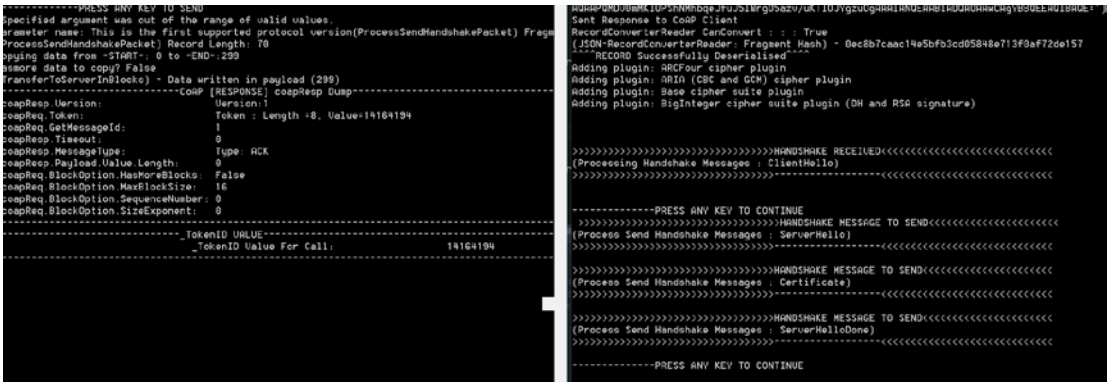

Fig. 13. Starting TLS Session

Fig. 14. Packet Acknowledgement

Table 1. The Master Secret Verification

SERVER MASTER SECRET (48byte)		CLIENT MASTER SECRET (after KDF) (48byte)
[0]	0x66	0x66
[1]	0x2b	0x2b
[2]	0x6c	0x6c
[3]	0xc0	0xc0
[4]	0x85	0x85
[5]	0x6f	0x6f
[6]	0x8c	0x8c
[7]	0x87	0x87
[8]	0x4c	0x4c
[9]	0x03	0x03
[10]	0x6e	0x6e
[11]	0xb9	0xb9
[12]	0x72	0x72
[13]	0xec	0xec
[14]	0xf7	0xf7
[15]	0xe6	0xe6

[16]	0x1c	0x1c
[17]	0x23	0x23
[18]	0x71	0x71
[19]	0xa7	0xa7
[20]	0xa7	0xa7
[21]	0x3f	0x3f
[22]	0xd8	0xd8
[23]	0xd4	0xd4
[24]	0x7d	0x7d
[25]	0x6c	0x6c
[26]	0xc5	0xc5
[27]	0x2a	0x2a
[28]	0x0f	0x0f
[29]	0x3b	0x3b
[30]	0x67	0x67
[31]	0xd2	0xd2
[32]	0x1d	0x1d
[33]	0x5e	0x5e
[34]	0x23	0x23
[35]	0xc3	0xc3
[36]	0x31	0x31
[37]	0x35	0x35
[38]	0xed	0xed
[39]	0xd2	0xd2
[40]	0x0e	0x0e
[41]	0xf6	0xf6
[42]	0x48	0x48
[43]	0xbc	0xbc
[44]	0xc3	0xc3
[45]	0x46	0x46
[46]	0xd4	0xd4
[47]	0x6e	0x6e

The final call is a Record type of 23, which is a TLS data Record. The test case was unable to process the Message Authentication Code correctly while trying to send data. This error manifested constantly after successful TLS handshake when trying to perform a data call. The cipher suite and cryptographic key materials were correct. This impacts negatively on the hypothesis but does not completely disprove the theory. The problem is with the test harness and not with the overall TLS protocol process. This needs further research to identify the core cause of such failure. Apart from that, this test case proved the overall hypothesis and is practically observable.

4.3 Test Case 3 – TLS Session via CoAP to HTTP Proxy

This test is to investigate whether the data packet can traverse via the proxy (as an offload example) and hit the HTTP server. The server, in this test case, should receive the ClientHello

fragment untouched from its original inception. Normal SHA1 hash on the original message and target message will be used to verify the Base64 fragments. The TLS Record structure, excluding the Fragment property, is not sensitive to slight changes but requires binary data to be unchanged. The TLS protocol fails if the TLS Record protocol sequence or Handshake data is changed. But for the sake of our Layer 7 data packet, the primary goal here is to measure if the TLS Record reaches beyond a typical Web Proxy, unchanged. TLS and DTLS will both ultimately "fail" this part of the process. When an SSL Off-loader or Gateway appliance is reached, SSL is terminated, and the traffic inspected at this point. Even in cloud servers, the processing farm web cluster might be dealing with a private communication between an IoT device and one of its servers. Depending on the data, a strict end-to-end communication might be required. This could be a banking transaction, a voting ticket from an e-Government system, or simply data that requires protection due to its sensitivity.

The first endpoint was the client, the second the HTTP proxy system; and the third the HTTP server endpoint. A change was made to add a specific GET CoAP Transfer call to the prototype.

Once the transmission was executed, the message is intercepted via the CoAP proxy service. At this point, we will present the output to indicate the transparent throughput between the ClientHello and target HTTP web-server. This will provide us with the baseline test to measure data, before and after. The normal ClientHello message is sent to the target server using the Proxy-Uri URL. For this instance, we plainly present this as a CoAP-to-HTTP proxy as used in IoT systems. The SHA1 hash is captured at each state to verify integrity for the source and destination records as seen in **Fig. 15**. When the packet arrived at the Proxy system, the SHA1 for the data matched, as the Record is intercepted from the incoming CoAP request. At this point, it should be noted that the CoAP protocol is easily translated due to its similar design to HTML. They are both stateless and relay on a request/response mechanism.

Here, the application layer data is intercepted for analysis. The SHA1 received by the proxy system is, "INFO - requestData SHA1: 7a80a31eb40beb046b97f2013971338a7baade50". This value matches the original request. At the final HTTP web server, which receives the data, one can see clearly that the data matches that of the source system's ClientHello message.

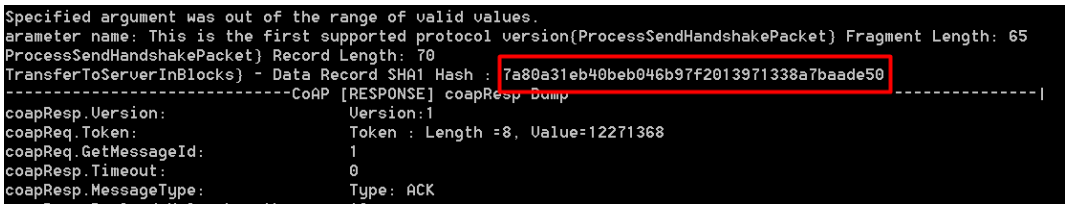

Fig. 15. Original Request SHA1

The next step is to simulate a simple man in the middle attack and to show how application layer data can be changed and basically reformatted for malicious use. This test is to prove the weakness in all modern web systems, where the data in transit is exposed. Because TLS one-way trust is used in our proposal, the data rewriting process will succeed if a proper TLS Client Record is replaced. Mutual authentication or trust will, however, solve this issue. This test is expected to prove that the data transmitted via a proxy can be changed in transit but can be also protected against such an attack (i.e. MiTM). A simple tamper program, as seen in **Fig. 16**, is created to realise the point. The software reads the TLS Record Fragment and does a bit shift on the first 10 array items. This is to maliciously modify the outgoing data to the HTTP server.

The tamperWithData flag is set in the HttpClientNIO.java file so that the tamper function is executed. The view data as received as seen in Figure 17 is now changed. Indeed, integrity is broken from the initial request and as we can see in Figure 18 that the SHA1 payload after it was changed after we injected our own packet as 7c91f7b6e722a8208a62fe14f77ae3726f6a7f50. The SHA1 as received by the Target HTTP server now shows the altered data with the same SHA1 hash of 7c91f7b6e722a8208a62fe14f77ae3726f6a7f50. Thus, the outcome of this experiment indicates and proves the threat of application layer data manipulation especially when done via proxy systems.

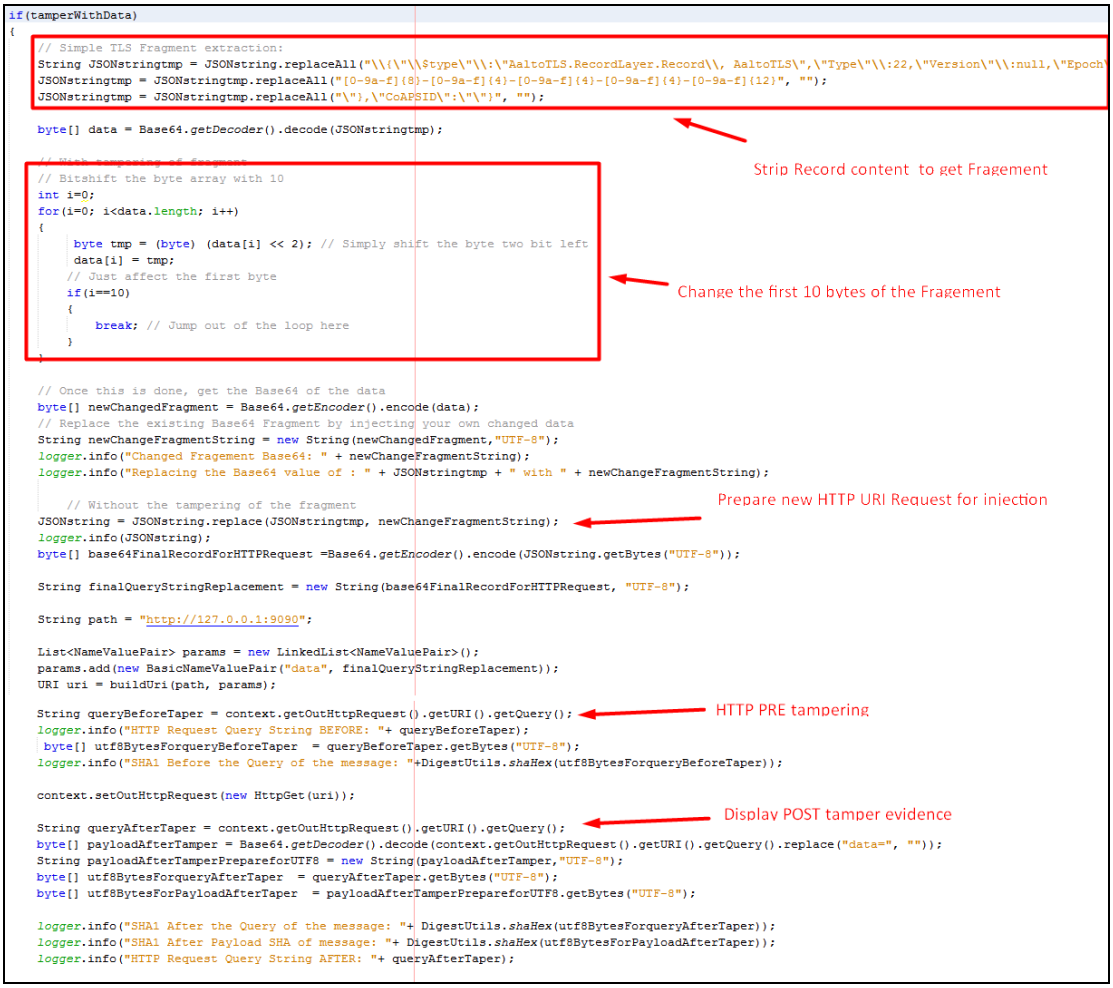

Fig. 16. MITM Attack Simulation: The Tampering Function

Fig. 17. MITM Attack realisation

```
INFO - new incomming CoAP connection
DEBUG - cache get: /127.0.0.1 9090
INFO - requestData String Value: {"$type":"AaltoTLS.RecordLayer.Record,
AaltoTLS","Type":22,"Version":null,"Epoch":0,"SequenceNumber":0,"Fragment":{"$type":"System.Byte[],
mscorlib","$value":"AQAAPQMDV0l7jygMfkQm7e0K8uvRvPzdxXubNVkmMItXfcrx5A0AAAIANQEAABIADQAOAAwCAgYBBQE
EAQIBAQE="},"CoAPSID":"d534bbb6-c0df-4eff-a2ff-e866153122d4"}
INFO - requestData SHA1 : 37767ea7ac34ed7665071a6a80d877f25608706c
INFO - Changed Fragement Base64: BAAA9AwMXCTsPKAMfkQm7e0K8uvRvPzdxXubNVkmMItXfcrx5A0AAAIANQEAABIADQAOAAwCAgYBBQEEAQIBAQE=
INFO - Replacing the Base64 value of :
AQAAPQMDV0l7jygMfkQm7e0K8uvRvPzdxXubNVkmMItXfcrx5A0AAAIANQEAABIADQAOAAwCAgYBBQEEAQIBAQE= with
BAAA9AwMXCTsPKAMfkQm7e0K8uvRvPzdxXubNVkmMItXfcrx5A0AAAIANQEAABIADQAOAAwCAgYBBQEEAQIBAQE=
INFO - {"$type":"AaltoTLS.RecordLayer.Record,
AaltoTLS","Type":22,"Version":null,"Epoch":0,"SequenceNumber":0,"Fragment":{"$type":"System.Byte[],
mscorlib","$value":"BAAA9AwMXCTsPKAMfkQm7e0K8uvRvPzdxXubNVkmMItXfcrx5A0AAAIANQEAABIADQAOAAwCAgYBBQEEAQIBAQE="},"CoAPSID
":"d534bbb6-c0df-4eff-a2ff-e866153122d4"}
INFO - HTTP Request Query String BEFORE:
data=eyIkdHlwZSI6IkFhbHRvVExTLlJlY29yZExheWVyLlJlY29yZCwgQWFsdG9UTFMiLCJUeXBlIjoyMiwiVmVyc2lvbiI6bnVsbCwiRXBvY2giOjAsIl
NlcXVlbmNlTnVtYmVyIjowLCJGcmFnbWVudCI6eyIkdHlwZSI6IlN5c3RlbS5CeXRlW10sIG1zY29ybGliIiwiJHZhbHVlIjoiQVFBQVBRTURWMGw3anlnT
WZrUW03ZTBLOHV2UnZQemR4WHViTlZrbU1JdFhmY3J4NUEwQUFBSUFOUUVBQUJJQURRQU9BQXdDQWdZQkJRRUVBUUlCQVFFPSJ9LCJDb0FQU0lEIjoiZDUz
NGJiYjYtYzBkZi00ZWZmLWEyZmYtZTg2NjE1MzEyMmQ0In0=
INFO - SHA1 Before the Query of the message: 65fbbf51ebe6b81165cc903fe12e77356a2aa928
INFO - SHA1 After the Query of the message: ffcdcb338542fbe90988f223eb641bda2be9426c
INFO - SHA1 After Payload SHA of message: 7c91f7b6e722a8208a62fe14f77ae3726f6a7f50
INFO - HTTP Request Query String AFTER:
data=eyIkdHlwZSI6IkFhbHRvVExTLlJlY29yZExheWVyLlJlY29yZCwgQWFsdG9UTFMiLCJUeXBlIjoyMiwiVmVyc2lvbiI6bnVsbCwiRXBvY2giOjAsIlNlcXVlbmNl
TnVtYmVyIjowLCJGcmFnbWVudCI6eyIkdHlwZSI6IlN5c3RlbS5CeXRlW10sIG1zY29ybGliIiwiJHZhbHVlIjoiQkFBQTlBd01YQ1RzUEtBTWZrUW03ZTBLOHV2UnZQe
mR4WHViTlZrbU1JdFhmY3J4NUEwQUFBSUFOUUVBQUJJQURRQU9BQXdDQWdZQkJRRUVBUUlCQVFFPSJ9LCJDb0FQU0lEIjoiZDUzNGJiYjYtYzBkZi00ZWZmLWEyZmYtZT
g2NjE1MzEyMmQ0In0=
```


Fig. 18. MITM Attack Simulation: The New Incoming CoAP Connection

4.4 Test Case 4 – Simulated JSON messages and IoT Environments

This test case is to check AaltoTLS on normal Sockets and to do a basic IoT string message exchange test. This should confirm that TLS Record structures can traverse over CoAP on emulated IoT devices. The basic TLS connection test session was successful. The following HTTP connection string was sent from AaltoTLS client software:

"GET / HTTP/1.1\r\n Host: www.checktls.com \r\n\r\n;"

A response was received from the www.checktls.com server, which identified itself as an Apache/2.4.6 (CentOS) OpenSSL/1.0.1e-fips compliant web server with the response: "HTTP 1.1 200 OK". This confirmed that a secure session was established and data decryption of the response message is successful. As mentioned in the introduction, a simple "message only" test will be conducted to verify IoT device resource constraints. For the normal .NET Framework system, Newtonsoft's JSON parser library was used. This test is needed to verify a scenario where CoAP is using TLS over JSON via an IoT Emulator. The test will be conducted using captured JSON structures from the CoAP-Client to CoAP-Server.

During the setup and development phase, the biggest problem was the limitation of the CoAP payload section. There is a limit of 1024 bytes for the Payload section and a maximum Datagram MTU of 1280 bytes. This was used over UDP to send and receive data to-and-from the CoAP sensor (i.e. server). The process of moving the TLS data in JSON format over to constrained IoT devices became a daunting task. During the test, the data format size for the ClientKeyExchange and the ServerHello-Certificate-ServerHelloDone TLS protocol Records were simply too large for the Payload block. Optimisation is needed for the basic structure to lower message size for this specific test case. The tests were to use envelopes by sending single messages alone.

Data transferred by JSON over CoAP gave inconsistent results. For example, JSON string data transferred from the client to the server caused an error on deserialization. To circumvent this problem, a Base64 encoding step was added before sending TLS record data over the wire. This inflated the basic message structure. To shorten the message structure, the following basic optimisation was carried out:

- Newline characters were shortened to line feed only (i.e. Character Return was removed). Newlines were still required for deserialization
- Base64 data step was added to solve JSON data integrity issues. However, this step inflated the data packet size with over 25% in total (See **Fig. 19**).

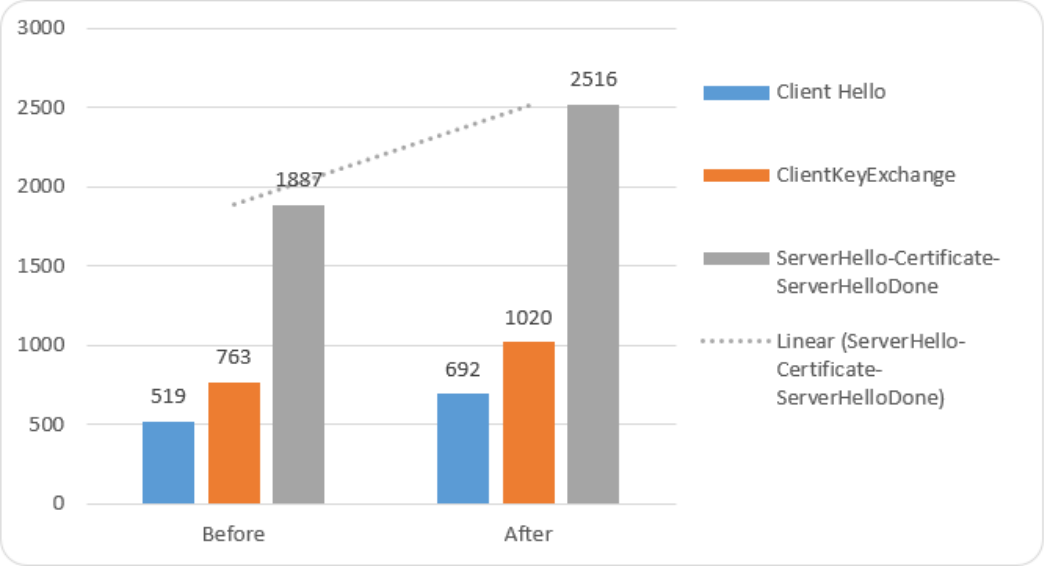

Fig. 19. Base64 added for JSON data integrity 25% increase in data structure size

After minor adjustments to the captured JSON files, the Record was loaded and sent to and from the CoAP server and IoT Client. The data size of the files was in an acceptable range of what was finally transferred over the wire. These tests were done using only the actual messages captured in previous successful tests. The memory profile of the system was quite high during the execution of the server (i.e. sensor) device. Although a Microsoft Emulator was used, the fact remains that the average memory consumption was at 311 603 bytes. This is rounded to 312KB of memory usage by the current software. Of the allocated 4 194 084 (4MB) memory, the system never went below 3 879 516 mark as seen in Figure 20. This data was captured by the emulators Garbage Collector while the CoAP server was serving requests. Such outputs theoretically confirm part of the hypothesis that messages can be sent by small IoT devices. Empirical results are still ultimately required to validate the latter hypothesis.

The formulae that was used to calculate the serialization time is as follows:

$$T = \frac{M}{500} * 20$$

Where:

- T is the total serialisation time
- M is the massage size
- 500 and 20 are constant serialisation coefficients

By using the formulae, **Table 2** shows results for the three messages used in the case study:

Table 2. TLS Message size and JSON parsing time

Message	Size	Time to Serialize (ms)	Time to De-serialize (ms)
Client Hello	*519*	*20.76*	*235.28*
ClientKeyExchange	*763*	*30.52*	*346.8*
ServerHello-Certificate-ServerHelloDone	*1887*	*75.48*	*855.44*

These measurements were taken from using only the message without any other processing requirements. The biggest message took 850 ms to parse using the current JSON parser. This reading indicates the entire handshake will take *1.564* seconds to process. This is just the JSON parsing alone. Depending on the size of the symmetric key and its data, the results may vary.

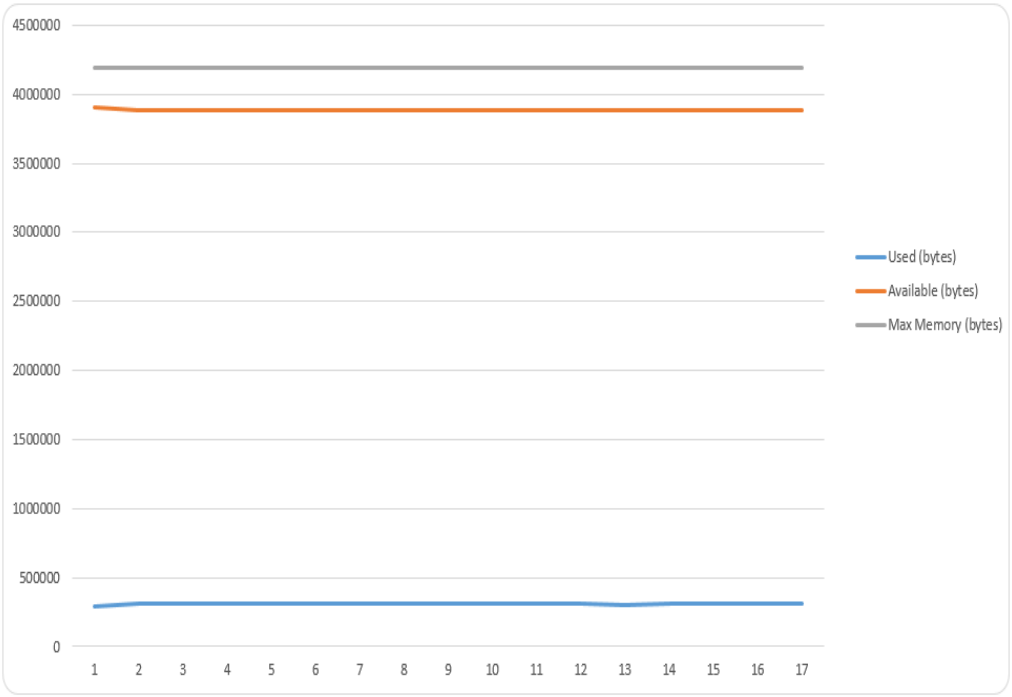

Fig. 20. Average CoAP Server memory utilization during calls

5. Conclusion and Future Work

This paper proposed a new methodology to secure communication in IoT environments using existing protocols. The building blocks chosen were CoAP as an Application Layer protocol and TLS for secure key agreement. CoAP's built-in asynchronous message parsing API provided a practical why to transport large TLS Record. This also proved that TLS could be used without a stateful TCP network stream. This was done by moving the session state management into the application layer. The drawback of using such a technology, although its proper potential in such device-constrained environment, is that CoAP is also not matured yet and could lead to unknown attacks during the key exchange. Key management is not inside the

scope of this research though and is considered one of the future directions. The test cases introduced in this research show that:

1. Establishing TLS Secure Session over CoAP is successful
2. TLS transposed as JSON baseline. In other words, the handshake phase is possible for the TLS record using JSON baseline.
3. TLS Session via CoAP to HTTP Proxy is possible but susceptible to the Man-In-The-Middle attack. The attack could be identified properly though the verification provided by CoAP. Using mutual trust could mitigate this attack.
4. JSON messages have a good potential in the IoT Environments.
5. This research also recommends that TLS to be used like a loosely coupled API and leveraged within the HTTP or RESTful service directly. Using a horizontal approach, rather than the typical funnel (stack) approach, better application layer security could be obtained

This research also recommends that TLS be used like a loosely coupled API and leveraged within the HTTP or RESTful service directly. Using a horizontal approach, rather than the typical funnel (stack) approach, better application layer security could be obtained. The following depicts the future directions of this research:

1. Formally investigate the general time and space complexity and to investigate the formal secrecy outage probability of the solution.
2. Evaluating the solution in a real-time scenario that utilises restrictive sensors.

References

[1] Almudawi, N.A., "Cloud Computing Privacy Concerns in Social Networks," *International Journal of Computer (IJC)*, *22*(1), pp.29-36, 2016.

[2] Abolfazli, S.A.E.I.D., Sanaei, Z., Sanaei, M., Shojafar, M. and Gani, A., "Mobile cloud computing: The-state-of-the-art, challenges, and future research," *Encyclopedia of Cloud Computing*, *Wiley, USA*, 2015.

[3] Best, E., Data Integrity Issues: Causes and Solutions, available from https://www.pda.org/publications/pda-publications/pda-letter/latest-news/2015/03/30/data-integrity-issues-causes-and-solutions, (last access 02.08.2016)

[4] Daniel, J., El-Moussa, F., Ducatel, G., Pawar, P., Sajjad, A., Rowlingson, R. and Dimitrakos, T., "Integrating Security Services in Cloud Service Stores," in *Proc. of IFIP International Conference on Trust Management*, Springer International Publishing, pp. 226-239, May 2015. Article (CrossRef Link).

[5] Dsadasd Arnbak, A., Asghari, H., Van Eeten, M. and Van Eijk, N., "Security collapse in the HTTPS market," *Communications of the ACM*, 57(10), pp.47-55, 2014. Article (CrossRef Link).

[6] Granjal, J., Monteiro, E. and Silva, J.S., "End-to-end transport-layer security for Internet-integrated sensing applications with mutual and delegated ECC public-key authentication," in *Proc. of IFIP Networking Conference*, pp. 1-9, IEEE, May 2013. Article (CrossRef Link).

[7] Hallman, S., Stahl, A. and Ahmadov, V., "The Causes, Security Issues, and Preventive Actions Associated with Data Integrity," *Communications of the IIMA*, 11(1), p.2, 2014.

[8] Kothmayr, T., Schmitt, C., Hu, W., Brünig, M. and Carle, G., "DTLS based security and two-way authentication for the Internet of Things," *Ad Hoc Networks*, 11(8), pp.2710-2723, 2013. Article (CrossRef Link).

[9] Lee, I. and Lee, K., "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," *Business Horizons*, *58*(4), pp.431-440, 2015. Article (CrossRef Link).

[10] Orebaugh, A., "What do we need to make IoT security a reality?" *Information Security Magazine*, 16, 31-33, 2014.

[11] Graham, M. and Dutton, W.H. eds., *Society and the internet: How networks of information and*

communication are changing our lives. OUP Oxford, 2014. Article (CrossRef Link).
[12] Nilsson, JD 2010, Digital Evidence in the Courtroom. [Electronic Book], n.p.: New York: Nova Science Publishers, c2010, University of Liverpool Catalogue, EBSCOhost, viewed 7 August 2016.
[13] Krutz, R.L. and Vines, R.D., 2010. *Cloud security: A comprehensive guide to secure cloud computing*. Wiley Publishing.
[14] Basile, C. and Lioy, A., "Analysis of application-layer filtering policies with application to HTTP," *IEEE/ACM Transactions on Networking*, *23*(1), pp.28-41, 2015. Article (CrossRef Link).
[15] Schultz, K., "Padding layer 7 security," *InfoWorld*, (23), pp.30-32, 2004.
[16] Rahmani, R. and Kanter, T., "Layering the Internet-of-Things with Multicasting in Flow-sensors for Internet-of-services," *International Journal of Multimedia and Ubiquitous Engineering*, 10, 2015. Article (CrossRef Link).
[17] Vucinic, M, Tourancheau, B, Rousseau, F, Duda, A, Damon, L, & Guizzetti, R 2014, 'OSCAR: Object Security Architecture for the Internet of Things', arXiv, EBSCO*host*, viewed 3 August 2016. Article (CrossRef Link).
[18] Modadugu, N., 2015. Datagram Transport Layer Security Version 1.2, https://tools.ietf.org/html/rfc6347, Internet Engineering Task Force (IETF), viewed 4 August 2016
[19] Granjal, J., Monteiro, E. and Silva, J.S., "Security for the internet of things: a survey of existing protocols and open research issues," *IEEE Communications Surveys & Tutorials*, *17*(3), pp.1294-1312, 2015. Article (CrossRef Link).
[20] Ko, M, & Dorantes, C 2006, THE IMPACT OF INFORMATION SECURITY BREACHES ON FINANCIAL PERFORMANCE OF THE BREACHED FIRMS: AN EMPIRICAL INVESTIGATION', http://jitm.ubalt.edu/XVII-2/article2.pdf, Journal of Information Technology Management Volume XVII, Number 2, viewed 7 August 2016
[21] Li, Q Jinmei, T & Shima, K, 2006, Ipv6 Core Protocols Implementation, pp. 1-27, ScienceDirect, EBSCOhost, viewed 7 August 2016.
[22] Durao, F, Carvalho, J, Fonseka, A, & Garcia, V 2014, 'A systematic review on cloud computing', Journal Of Supercomputing, 68, 3, pp. 1321-1346, Computers & Applied Sciences Complete, EBSCOhost, viewed 7 August 2016. Article (CrossRef Link).
[23] Perera, C., Talagala, D.S., Liu, C.H. and Estrella, J.C., "Energy-Efficient Location and Activity-Aware On-Demand Mobile Distributed Sensing Platform for Sensing as a Service in IoT Clouds," *IEEE Transactions on Computational Social Systems*, *2*(4), pp.171-181, 2015. Article (CrossRef Link).
[24] Wang, C.F. and Yang, D.L., "A Study on Decoupling Flow Engines Toward Cross-Platform Interoperability," In *Information Science and Applications (ICISA) 2016*, pp. 1081-1091, Springer Singapore, 2016. Article (CrossRef Link).
[25] Wang, J., Yang, Y., Chen, L., Yang, G., Chen, Z. and Wen, L., "A Combination of Timing Attack and Statistical Method to Reduce Computational Complexities of SSL/TLS Side-Channel Attacks," in *Proc. of 2015 11th International Conference on Computational Intelligence and Security (CIS)*, pp. 402-406, IEEE, December 2015. Article (CrossRef Link).
[26] Yulianto, S., Lim, C. and Soewito, B., "Information security maturity model: A best practice driven approach to PCI DSS compliance," in *Proc. of 2016 IEEE Region 10 Symposium (TENSYMP)*, pp. 65-70, IEEE, May 2016. Article (CrossRef Link).
[27] Loo, J., Mauri, J.L. and Ortiz, J.H. eds., 2016. *Mobile ad hoc networks: current status and future trends*. CRC Press.
[28] Sheffer, Y., Holz, R. and Saint-Andre, P., 2015. *Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)* (No. RFC 7525).
[29] Han, J., 2016, March. Chaining the secret: Lightweight authentication for security in pervasive computing. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)* (pp. 1-3). IEEE. Article (CrossRef Link).
[30] Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F. and Alonso-Zarate, J., "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud Computing*, *3*(1), pp.11-17, 2015.

[31] Torres, R., Javali, N., Border, J. and Ganesan, V., Torres and Robert, *Dynamic disabling of multi-step transport layer handshake spoofing in performance enhancing proxies (peps) in broadband networks*. U.S. patent 20,150,381,752, 2015.
[32] Kieseberg, P., Frühwirt, P., Schrittwieser, S. and Weippl, E., "Security tests for mobile applications—Why using TLS/SSL is not enough," in *Proc. of Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, pp. 1-2, IEEE, April 2015. Article (CrossRef Link).
[33] Kim, H.S., Im, H., Lee, M.S., Paek, J. and Bahk, S., "A Measurement Study of TCP over RPL in Low-power and Lossy Networks," *Journal of Communications and Networks*, *17*(6), pp.647-655, 2015. Article (CrossRef Link).
[34] Capossele, A., Cervo, V., De Cicco, G. and Petrioli, C., "Security as a CoAP resource: an optimized DTLS implementation for the IoT," in *Proc. of 2015 IEEE International Conference on Communications (ICC)*, pp. 549-554, IEEE, June 2015. Article (CrossRef Link).
[35] Kwon, H., Park, J. and Kang, N., "Challenges in deploying CoAP over DTLS in resource constrained environments," *International Workshop on Information Security Applications*, pp. 269-280, Springer International Publishing, August 2015. Article (CrossRef Link).
[36] Curran, J., Fenton, N. and Freedman, D., *Misunderstanding the internet*. Routledge publication. pp. 2, 2012.
[37] Jackson, W., "HTML5 History: The Past and Future of HTML Markup," *HTML5 Quick Markup Reference*, pp. 1-4, Apress, 2016a. Article (CrossRef Link).
[38] Jackson, W., "An Introduction to JSON: Concepts and Terminology," *JSON Quick Syntax Reference*, pp. 15-20, Apress, 2016b. Article (CrossRef Link).
[39] Castro, M., Jara, A.J. and Skarmeta, A.F., "Enabling end-to-end CoAP-based communications for the Web of Things," *Journal of Network and Computer Applications*, *59*, pp.230-236, 2016. Article (CrossRef Link).
[40] Xu, Q., Ren, P., Song, H. and Du, Q., "Security enhancement for IoT communications exposed to eavesdroppers with uncertain locations," *IEEE Access*, *4*, pp.2840-2853, 2016. Article (CrossRef Link).