LIGHTWEIGHT IOT SECURITY MIDDLEWARE FOR END-TO-END CLOUD-FOG COMMUNICATION

A Thesis presented to

the Faculty of the Graduate School

at the University of Missouri

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

BIDYUT MUKHERJEE

Dr. Prasad Calyam, Thesis Supervisor

MAY 2017

The undersigned, appointed by the Dean of the Graduate School, have examined the thesis entitled:

LIGHTWEIGHT IOT SECURITY MIDDLEWARE

FOR END-TO-END CLOUD-FOG COMMUNICATION

presented by Bidyut Mukherjee,

a candidate for the degree of Master of Science and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Prasad Calyam

Dr. Grant J. Scott

Dr. Timothy Middelkoop

ACKNOWLEDGMENTS

I am extremely thankful to have Dr. Prasad Calyam as my advisor and mentor. He has imparted a lot of his enthusiasm and love for this field to me, and I would like to thank him for all the guidance and direction that has proven crucial to me as a student of computer science and as a researcher. He has been very patient, supportive and encouraging throughout my time here, and his keen insight and out-of-box pedagogy has taught me how to approach scientific research with the excitement that it deserves. Next, I would like to express my gratitude to Dr. Grant Scott and Dr. Tim Middelkoop for taking interest in my research, dedicating time to it, and for consenting to be a part of my committee. In addition, I would like to extend my gratitude to my friend and research partner Roshan Neupane, without whose help this research would have been far from ready. Our shared vision for this research and few others before it have brought them all to fruition.

Next, I would like to thank my fellow team members Songjie Wang, Wenyi Lu, Daniel Dunn, Yijie Ren, and Qi Su, who have dedicated hours and worked hard to push the boundaries in this research and give it a new angle. This research has made leaps and bounds because of their joint effort.

I would also like to thank all the members of VIMAN Lab, who have guided and helped me selflessly. Their dedication to the lab and their research is commendable. I have great respect for all the professors who have taught me concepts at such an advanced level, or guided me in other ways. I am grateful for the opportunity that this department and University have offered to me at so many different levels. I would like to thank Elizabeth Jackson, who has been extremely loving and supportive since the very beginning. And finally, I would like to thank my parents, whom I know to be the best people in the world. None of this would be possible without their love and guidance. Thank you, everyone.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

IoT (Internet of Things) based smart devices such as sensors and wearables have been actively used in edge clouds i.e., ‘fogs’ to provide critical data during scenarios ranging from e.g., disaster response to in-home healthcare. Since these devices typically operate in resource constrained environments at the network-edge, end-to-end security protocols have to be lightweight while also being robust, flexible and energy-efficient for data import/export to/from cloud platforms. In this thesis, we present the design and implementation of a lightweight IoT security middleware for end-to-end cloud-fog communications involving smart devices and cloud-hosted applications. The novelty of our middleware is in its ability to cope with intermittent network connectivity as well as device constraints in terms of computational power, memory and network bandwidth. To provide security during intermittent network conditions, we use a Session Resumption concept in order to reuse encrypted sessions from recent past, if a recently disconnected device wants to resume a prior connection that was interrupted. The primary design goal is to not only secure IoT device communications, but also to maintain security compatibility with an existing core cloud infrastructure. Experiment results show how our middleware implementation provides fast and resource-aware security by leveraging static pre-shared keys (PSKs) for a variety of IoT-based application requirements. Thus, our work lays a foundation for promoting increased adoption of static properties such as Static PSKs that can be highly suitable for handling the trade-offs in high security or faster data transfer requirements within IoT-based applications.

Chapter 1

Introduction

1.1 Security in IoT Applications

Internet of Things (IoT) systems typically comprise of a network of connected devices with limited computation and networking capacity. The term "*thing*" here can constitute any smart device ranging from sensor devices in automobiles, bio-chemical sensing devices in homeland security, to heart monitoring devices inside of a human body. In fact, any object that has the ability to collect and transfer data across the network can be a part of the IoT system.

As mentioned in [1], emerging IoT trends are set to completely change the way businesses, governments, and consumers interact with each other, and act in a data-driven economy. *The use-case range of IoT applications is very broad.* IoT devices are used in various fields e.g., Geo Sensors collect all sorts of geographical information related to soil, forest terrains, and weather, and transmit related data sets to nearby fog computing platforms for aggregation and analysis/visualization. They can also provide critical data during e.g., disaster response scenarios or in-home healthcare.

An example of disaster response system utilizing IoT is Panacea's Cloud [2, 3]. Such systems aid in providing medical triaging and quick response during emergency disaster situations. The system involves many remote and distributed IoT devices transmitting data from various locations in a disaster region to a responding personnel's handheld device (IoT), which in turn utilizes a common core cloud system to provide visualization through a dashboard. On the other hand, in-home healthcare systems such as [4] are aimed at providing emergency services to the elderly, if the need arises. These involve IoT sensors and cameras keeping track of movements made by the elderly, and notifying their primary care contact if their movement signature appears to be an anomaly.

Hence, the sheer number and range of IoT applications is vast and varied. 'Ubiquitous Computing', as it is also referred to as, has been on the rise. As it gets more and more intrusive into our lives, a serious facet to consider is the security of such systems and applications. Considering the nature of confidentiality that many of the use-cases might require, it is imperative to ensure secure transmission and reception between the devices in an IoT systems, and between IoT systems and connected centralized services. Any loss of data, confidentiality, integrity, or availability can be catastrophic to the system and people involved, especially for applications of critical importance. At every stage there need to be protocols in place to facilitate secure and reliable working.

1.2 Need for a Middleware

IoT devices typically operate in resource-constrained (computing, memory, storage, energy) environments at the network-edge. Consider the aforementioned case of disaster response systems [3]; the edge network here comprises of IoT devices and network gateways to the cloud. Such systems could become highly unstable and unreliable due to physical infrastructure damage. On top of good level of security, more importance has to be placed on the speed or quickness of transmitting the available emergency data from IoT sensors to the responders' devices. The availability of IoT devices could be infrequent and intermittent, and there is no certainty to when the devices might run out of energy before successfully sending or receiving critical data. To make matters worse, security association and handshake between the IoT devices and the gateways could consume a lot of the limited computing, memory, network, and energy resources available. Frequent drops in connectivity would result in frequent handshakes, making loss of critical data a high probability.

On the other end of the IoT use-case spectrum, the fog might be used for providing ElderCare-as-a-Service, as in [4], that requires significant amount of resources to handle the big data generated from patient homes. In this case, security needs to be configured for data confidentiality and integrity, even if data transfer speeds are affected due to security overhead. More importance has to be placed on securing the data end-to-end than to transmit it with utmost speed. Any loss of confidentiality could result in extreme breach of privacy for the patients involved, considering how invasive the systems can be.

Hence, IoT-based applications almost always have to tackle trade-offs, and can have extremely contrasting use-cases which are hard to be generalized. Unfortunately, most implementations exist in an ad-hoc manner, forcing developers or IoT users to fully tailor their program around their needed application while keeping speed, security, and many other parameters in mind to suit their case. This is not ideal, specially since every IoT application should be guaranteed some level of security, regardless of the developer's choice or implementation. Moreover, there exists a fundamental difference between systems involving full desktop/server capabilities, and systems constituting remote, often battery-powered, IoT devices. The needs and challenges of both different, and so are many of the protocols involved.

To alleviate most of these issues, and to bridge the gap between the different kinds of network systems connecting various devices of differing capabilities, there can be two obvious ways of approaching the solution. (i) Develop a full set of *protocols*, or, (ii) Develop a *middleware* to wrap existing protocols to simulate the behavior of cross-compatibility. Both have their advantages and disadvantages, and so considering that, we have decided to go with developing a new middleware to make our solution more extensible, simpler to implement, and to allow customizing the requirements, keeping the basic fundamental requirements of security intact.

Hence, in this paper, we address the above challenges and propose the design and implementation of an end-to-end IoT security middleware for cloud-fog communication that can be suitably used with most IoT-based applications.

1.3 Intermittent and Flexible Security for Edge-Cloud

The core features of our end-to-end IoT security middleware, and the main thesis contributions are: (i) Intermittent Security, and (ii) Flexible Security.

1.3.1 Intermittent Security

Our middleware uses a *Session Resumption* concept in order to reuse encrypted sessions from recent past. If a recently disconnected device wants to resume a prior connection that was interrupted due to an unreliable network, this key feature has the potential to save a lot of time, energy, computation, memory, and bandwidth, during the reconnection phase of the associated devices.

1.3.2 Flexible Security

Our middleware has the ability to allow users to flexibly configure required security based on the application resource-awareness, versus blindly following a rigid security configuration. Every application has its own specific requirements, and implementing a trustable, reliable security is often a challenge to developers. This enables the user to consider the trade-offs for their IoT-based application, and thus configure higher security, prioritize faster data transfer, or simply use the most energy efficient solution.

1.4 Thesis Outline

The remainder of this thesis is organized as follows: In Chapter 2, we describe the thesis background and literature review, that provide context to the solution approach. In Chapter 3, we elaborate on our solution and provide a detailed description of our approach with reference architecture. Chapter 4 evaluates the effectiveness of our middleware and compares the state of an IoT-based application with and without the middleware. Chapter 5 discusses future work and provides information on extending the many facets of the middleware. Finally, Chapter 6 concludes the thesis.

Chapter 2

Background and Literature Survey

In this chapter, we provide some background information on fundamental concepts behind the middleware. We then discuss the various literature work that have led to the idea and implementation of this research.

2.1 Edge-Cloud Architecture

The architecture for IoT-based network, including Edge-Cloud systems, vary on a per-application basis. But in general, the overview logical diagram can be visualized as shown in Figure 2.1. The essential elements are (i) Core Cloud, (ii) Gateway, (iii) IoT Edge.

The *Core Cloud* is the most computationally powerful element in the system. It performs data aggregation from various edges or gateways, as well as acts as the server for most systems. It provides a scalable solution in the form of *services* to be used by gateways or IoT edges. A few examples of Cloud can be seen in Amazon Web Services, Google Cloud, Microsoft Azure, GENI, etc.

The *Gateway* acts as a network and data bridge between the IoT Edge and the Core Cloud. It is usually used to collect data from the IoT devices wirelessly connected to it and

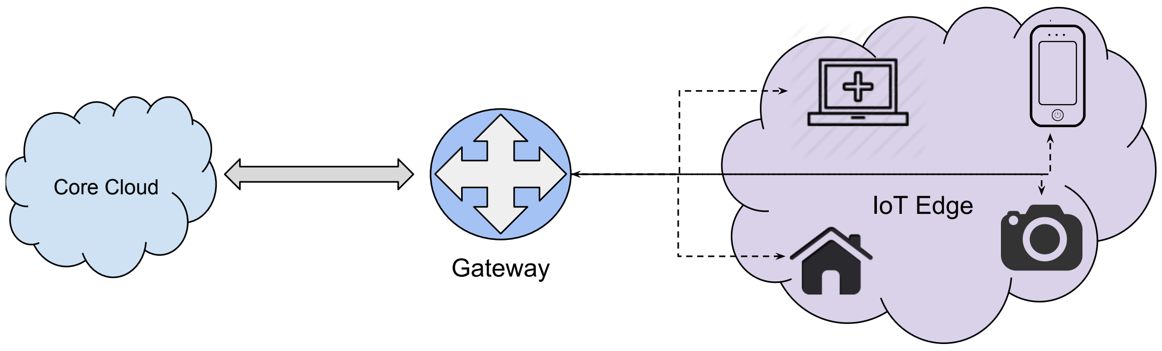


Figure 2.1: Typical IoT-based application architecture

allows a translation of protocols between its two network ends. More often than not, the gateway is geographically closer to the IoT Edge than to the Core Cloud. It can perform various tasks, including routing, forwarding, buffering, etc.

And finally, the *IoT Edge* consists of a bunch of IoT devices all either interconnected, or connected through the Gateway as a hop. Devices such as smartphones, cameras, sensors, wearables, etc. are a few examples of IoT devices. The actual data collection at the source is done through these devices. Commonly, Core Cloud might have multiple gateways interacting with it, with each Gateway having many IoT devices in its network.

Typically, the cloud-gateway network has a set of protocols in use, while the gateway-edge has its own set of protocols, due to the difference in the nature of the two networks. Hence, the gateway has an additional task of translating the protocols to ensure exchange between the two networks.

2.2 Security Protocols

Security is an extremely important requirement for most devices and applications. It is commonly implemented using base set of instructions, called Security Protocols. These protocols exist in many different forms, often in combinations. The types can include: (i) Key Agreement, (ii) Entity Authentication, (iii) Application-level data encryption, (iv)

Symmetric Encryption, (v) Asymmetric Encryption, (vi) Message Authentication, and many others. It is often a challenge to decide on the level of best protocols to be used in a given application. In addition, making them work together is not always easy.

Often, security is implemented using TLS (Transport Layer Security), or DTLS (Datagram TLS). This has replaced the older SSL (Secure Sockets Layer) technology. There's even more available options in the form of RTP (Real Time Protocol), and its new extension SRTP (Secure Real Time Protocol). All these protocols have a few things in common, namely, (i) They ensure secure, encrypted communication among connected devices, (ii) They work using a timed session, (iii) They undergo *handshake*, which *usually* involves exchanging keys or certificates to establish authenticity, (iv) Data transmitted through them is encrypted. Conversely, they also have a few key differences, such as (i) use-cases, (ii) default underlying transport protocols in place, etc.

On top of this, protocols also differ based on the type of device it is aimed for. For instance, a full fledged desktop computer might have all levels of crypto protocols in place, but the same would not be feasible for an IoT-based application in an IoT Edge.

2.3 Panacea's Cloud

Panacea's Cloud [2, 3, 5] is an example of IoT-based application utilizing the power of Cloud-Edge system. Its goal is to aid in emergency medical traiging, as part of incidence response during any natural or man-made disaster. The architecture for this application looks like Figure 2.2.

Similar to the basic Cloud-edge architecture shown in Figure 2.1, Panacea's Cloud has all the components from the previous architecture, but has an additional component, the Edge Cloud. The application is novel in the sense that it utilizes a network mesh of IoT devices to collect data from patients and transmit to the private Edge Cloud, for processing and visualization.

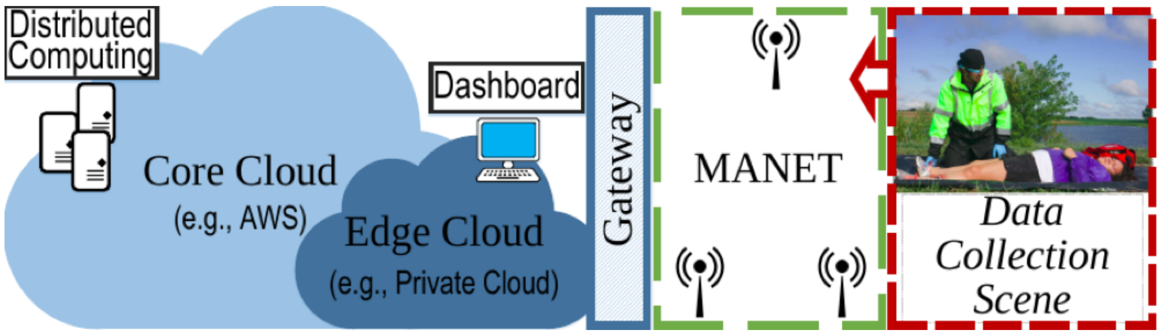


Figure 2.2: Panacea's Cloud Disaster Response System architecture

The system has a major challenge to overcome, in the form of security trade-off. The application deals with human patients, so confidentiality is crucial. But even more important is the speed of data to be transferred, i.e., there is a need to quickly get the data into the edge or core cloud so as to allow timely response. Hence, both speed and security need to be as high as possible. Unfortunately, it's almost impossible to have both of those characteristics maxed-out. And so, there is a need of decision making to select the higher priority and operate accordingly.

2.4 Literature Review

2.4.1 Constrained and Unconstrained Network

IoT-based application deployment is a relatively new trend. However, methods to secure networked IoT devices have been explored in the past. Work in [6] is based upon the categorization of IoT-based network into two kinds: (i) *Constrained Network* (CN), and (ii) *Unconstrained Network* (UCN). Constrained Network is said to typically consist of network of low-resource devices, i.e., devices with limited computation, network, bandwidth, etc. Such networks are usually designed to support low bandwidth transfers. An example of a Constrained Network would be a few small IoT devices interconnected through a gateway. On the other hand, Unconstrained Network involves network of high-resource devices. Such networks are not constrained by bandwidth or energy limitations. An example of this would be our home desktops connected to the Internet.

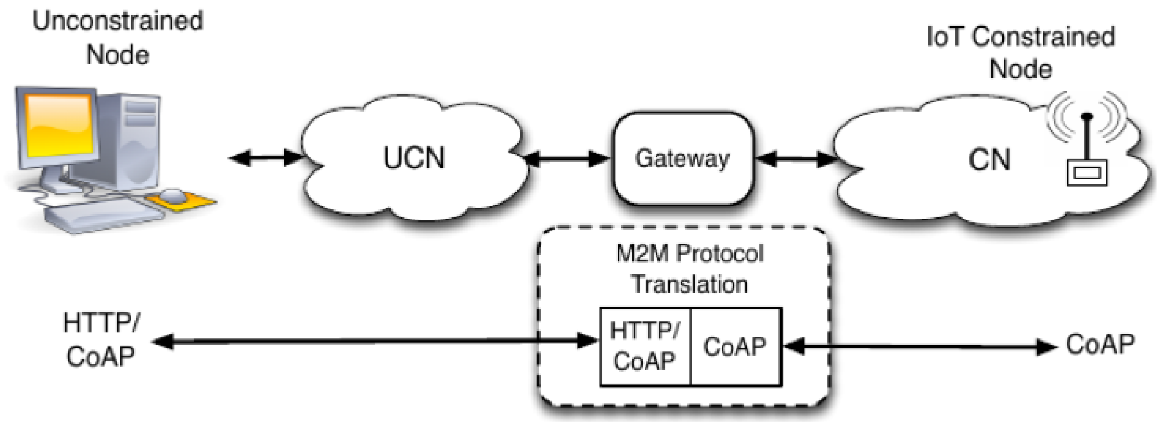


Figure 2.3: Constrained and Unconstrained Network

Figure 2.3 shows the role of gateway in connecting CN and UCN. Due to the fundamental differences in CNs and UCNs, use of full-fledged security suites is not feasible on both network types. And so, the cited paper utilizes a separate protocols for CN and UCN. In their case, the gateway acts as the bridge and performs translation from one protocol to the other to allow end-to-end communication. This is a useful idea that has been extended further in our middleware.

Although setting up of secure end-to-end channels isnt directly applicable to Constrained Network, the cited work provides a suitable security architecture for IoT discussing the following objectives: a) No modifications in the security suites employed within the Unconstrained Networks such that the unconstrained nodes do not notice any deviations from their standard procedures. b) Security handshakes handled within the Constrained Networks so that the constrained nodes handle their complexity being able to establish end-to-end secure channels. In essence, an architecture to offload computation intensive tasks to the gateway is proposed, which helps in reducing the cost of security encryptions at the IoT node side.

However, offloading at a large scale is a tedious task as mentioned in [7]. In addition, the implementation itself is tailored around IPsec, and does not discuss usage with OpenVPN and similar tunneling suites. Moreover, due to the stringent nature of the research, it might not be flexible enough for unstable or dynamic topologies which might face infrequent connectivity or many temporary nodes connecting transiently on a need-to basis.

Overall, the logical division of Constrained Network and Unconstrained Network is a key takeaway from the paper, in addition to the idea of using gateways to offload computation-intensive tasks.

2.4.2 Lightweight IoT Authentication Scheme

Publication [8] provides a lightweight authentication scheme for heterogeneous wireless sensor networks in the context of IoT. The scheme uses nonce and Keyed-Hash Message Authentication (HMAC) to check the integrity of authentication exchange. The authors use a 3-step set up of the communication. Firstly, registration between nodes and gateway, and then registration between gateway and the cloud/remote server. Secondly, authenticating the connection. The authentication required directly link of the sensor to the remote user before the gateway comes into play. The sensor creates a nonce N of not more than 8 bytes and sends it to the remote user. The remote user then creates a nonce of its own, M, appends with that of the sensor and then sends it to the gateway creating ID for the sensor. The gateway computes HMAC and checks if the information is valid. Then it processes it with its own nonce, S, and sends it to the remote user. The remote user node then sends the information for the sensor node to confirm. Thirdly, there is a session key establishment between the sensor node and the remote user. The paper deals with different security issues and present performance evaluation and analysis to support its approach. Impersonating the sensor, replay message and so on. The mutual authentication is an advanced approach that enhance security. Along with this, the papers talk about the identity protection, data integrity, and other features they cover. The work is scalable. Similar form of architecture is used in [9] for creating an eHealth application based on the approach discussed in [8]. The properties of scalability, mutual authentication feature, lightweight schemes and direct communication to the remote user/cloud are directly relevant to our work.

Since most of the encryption computation is offloaded, the beacons/nodes consume less energy in computation of the connection establishment. It allows both users and Base stations to authenticate each others. However, there is multi-phase encryption and decryption which increases overhead and takes a lot of time.

In comparison, our work investigates a security middleware which makes use of static Pre-Shared Keys (PSKs) that is different from the paper's multi-phase encryption and decryption. A major advantage in doing this is that we are able to phase down to just one iteration for authentication, and thus, our approach reduces security overhead and is quite less time consuming.

2.4.3 Physical Unclonable Functions

We encountered use of different encryption techniques. Work in [10] makes use of Physical Unclonable Functions (PUF) over public key cryptography, which takes advantage of existing physical properties of the device. For encryption and authentication services, they combine this technology with Physical Key Generation (PKG) over wireless communication that use physical properties of the communication channel. PKG uses same components as PUF making the combination cheaper than common public-key cryptography. The authors show how the approach exploits random physical nature of noisy wireless channel between devices by tapping the wireless channel itself for secret key generation.

Figure 2.4 shows that the cited paper utilizes a Hub/Gateway to keep track of various IoT Devices $\{T_1 ... T_i\}$. Using PUF has the added benefit of making security association

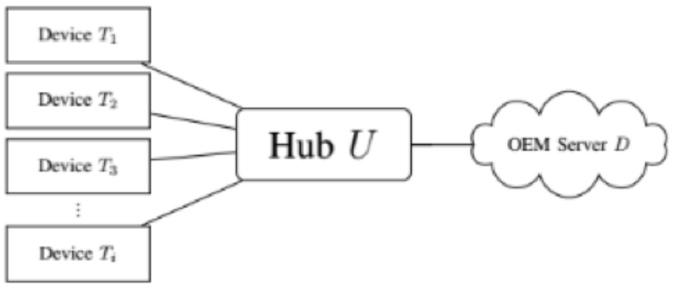


Figure 2.4: Physical Unclonable Functions

faster. The idea of using physical properties of the device is quite useful, especially for IoT-based applications, since it helps increase security. But, since the process of authentication involves full re-enrollment from scratch, it turns into an expensive solution for Constrained Network environments. Fortunately, we are able to go one step further in our middleware while using physical static properties of the device in conjunction with session ID to provide quick re-enrollment.

2.4.4 Secure End-to-End Protocol with Offloading

Paper [11] is a closely related finding. It provides a secure end-to-end protocol for resource-constrained devices, especially in context of health-care sensors. Their work uses same security functionality as Unconstrained Devices, but without computationally intensive operations. Heavy computation at constrained devices are offloaded to the neighboring trusted nodes/devices. The session, however, is secured using a session key. The key generated has a short-time validity and hence is ephemeral in nature. This is great for security, but at the expense of having to renegotiating handshake and security association all over again, in case of any disconnection.

This paper also proposes a selection criteria based on trust level to select the assisting nodes. Their protocol is compatible with other end-to-end security protocols, allowing extensibility. Our work builds on top of this work, and uses an easier, yet effective key management scheme. Specifically, we allow creating static keys which are not short-lived, significantly reducing the key exchange cost and time.

2.4.5 HIP and Session Resumption

Paper [12] provides a comprehensive session resumption mechanism. The work uses HIP DEX i.e., Host Identity Protocol Diet EXchange, a key management protocol specially designed for Constrained Networks, that provides secured end-to-end connections in IoT. To this end, the authors present complementary lightweight protocol extension for HIP DEX, i.e., a comprehensive session resumption mechanism. Ephemeral Diffie-Hellman keys and digital signatures are forfeited and are replaced by a refined session establishment handshake based on DH keys for mutual peer authentication and key agreement. Perfect forward secrecy and non-repudiation properties of HIP result in significantly decreased protocol handshake overhead and reduced handshake run-time. The peers only perform expensive operations once during the initial sessions establishment. Storage of session state after session tear-down enables efficient re-authentication and re-establishment of a secure payload channel in an abbreviated session resumption handshake.

However, it is not the most secure solution. Our work utilizes the concept of session resumption but makes a few changes for broader compatibility. Instead of HIP, we utilize Device ID, which can additionally act as a static unique device property to quickly and securely fetch stored sessions for re-use.

2.4.6 Security Configuration Frameworks for LLNs

Authors in [13] give a standard security complaint framework to secure the IEEE 802.15.4 networks in low power lossy network (LLNs). The framework aims at proposing: a) different kinds of security architectures, b) an efficient mechanism for initializing a secure IEEE

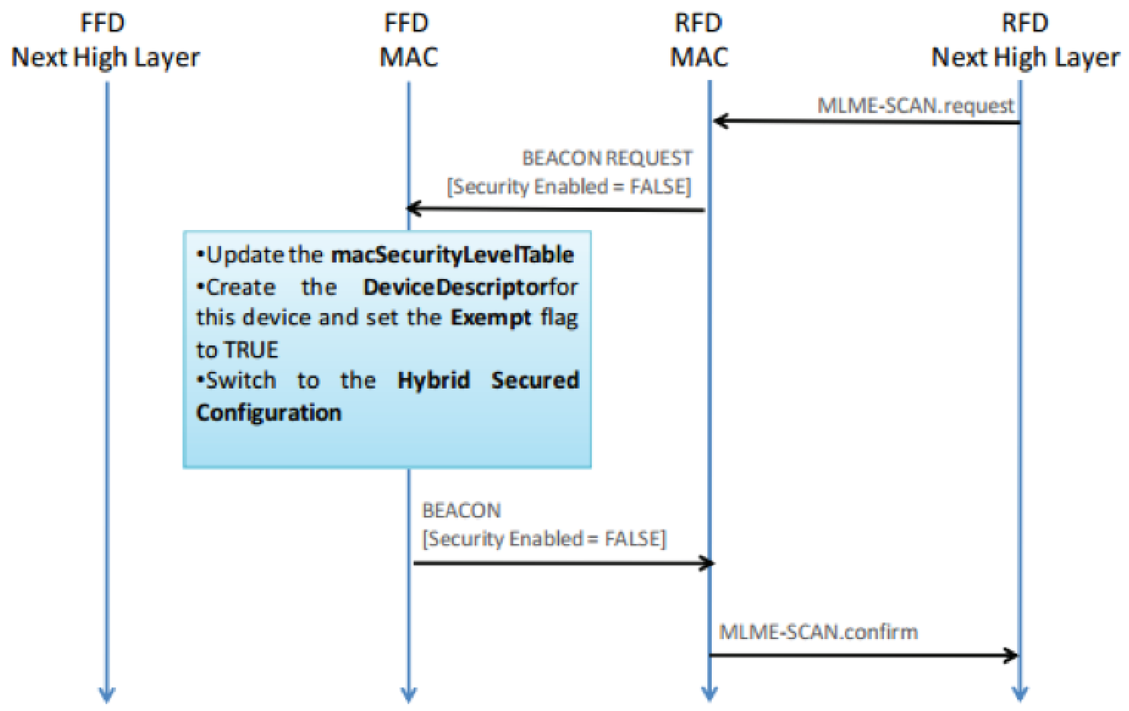

FFD
Next High Layer
FFD
MAC
RFD
MAC
RFD
Next High Layer
MLME-SCAN.request
BEACON REQUEST
[Security Enabled = FALSE]
•Update the **macSecurityLevelTable**
•Create the **DeviceDescriptor** for this device and set the **Exempt** flag to TRUE
•Switch to the **Hybrid Secured Configuration**
BEACON
[Security Enabled = FALSE]
MLME-SCAN.confirm

Figure 2.5: Interaction between FFDs and RFDs under the Framework

802.15.4 domain and c) a lightweight scheme to negotiate link keys among the devices. The standard considers two types of nodes that can build peer-to-peer or star networks: *Full Function Devices* (FFD) and *Reduced Function Devices* (RFD). The RFDs have limited resources, and low computational capabilities, while the FFDs have full computational capabilities. The FFDs are able to coordinate the network and hold references to the RFDs.

The framework supports 5 different levels of security with their proposed security configurations (i.e., *Fully Secured, Unsecured, Partial Secured, Hybrid Secured and Flexible Secured*). Flexible secured configuration has the potential to change the level of security based on requirements when needed, and shifts from full secured state to hybrid secured state. Figure 2.5 shows a small instance of cross-interaction between RFDs and FFDs. However, the approach is not quite scalable since re-entry of device request is not supported. This is a gap that can be filled by the presence of a flexible, dynamic security middleware to act as an interface for fast or secure encrypted communication. Hence, our work takes the framework a step further towards a more practical layer for use within IoT-based applications.

Chapter 3

IoT End-to-End Security Scheme

3.1 Overview

As mentioned earlier, our work closely follows the idea of flexible security framework described in [13]. We build upon the previous framework by adapting their implementation for our core features suited for a variety of application use cases.

3.1.1 Physical Infrastructure

Figure 3.1 illustrates a physical infrastructure perspective of our middleware in action. The infrastructure primarily constitutes of a core cloud infrastructure, a gateway, and several edge IoT devices. The core cloud has a communication channel with gateways at the edge. The edge gateways form an interface to the network for the IoT devices. Parts of the primary middleware are installed on both the gateway and the fog IoT devices. The fog network can constitute any number or type of IoT device, such as heart monitor, beacon, geo sensor, etc. The primary middleware allows secure and fast data transfer between the sensor devices and the gateway, using security schemes chosen through our ”flexible security” module, and ensuring robustness using the ”intermittent security” feature to accommodate frequent disconnections. This is possible because the middleware stores and tracks sessions, certificates, or keys,

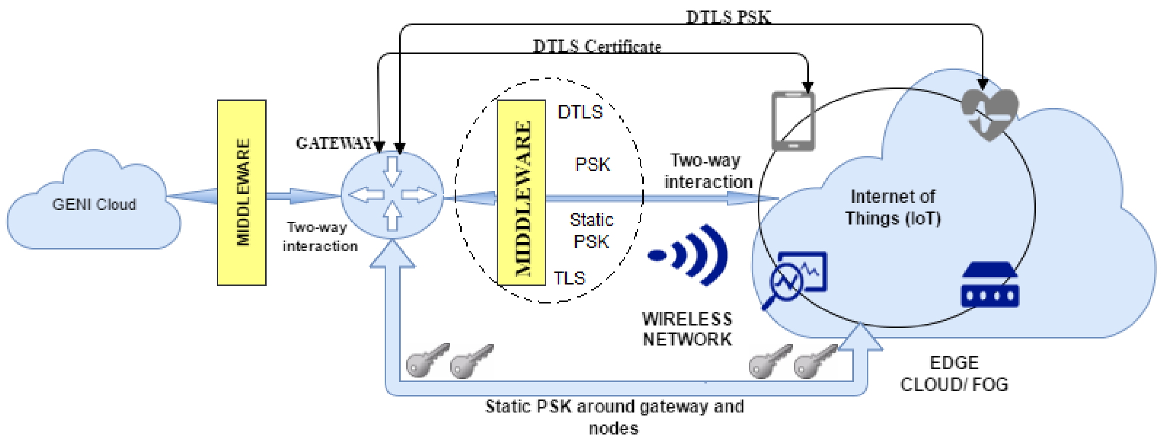


Figure 3.1: Edge Cloud/Fog system architecture

between both the fog and the gateway. Once data has been securely transferred to the gateway, it handles the translation of protocols to allow compatibility between the core cloud protocols and the IoT protocols.

Optionally, secondary middleware can exist between the gateway and the core cloud to provide flexibility and robustness, if needed. There is a key benefit of having this setup. The presence of intermediate gateways allows for decoupling of services and protocols between the cloud-gateway and gateway-iot subnets, essentially paving way for end-to-end security via our intermediate middleware. Our model consists of a middleware in the core cloud network side, and another middleware at the fog network side. Each middleware consists of a server-client pair interacting with just each other. At the gateway, the client of cloud interacts with the server of edge, allowing end-to-end secured communication. The middleware supports flexible security by allowing different protocols for individual nodes in the fog network, in addition to being ready to use static PSKs for quick encryption setup, and support for intermittent security through session resumption.

3.1.2 Logical Modules

A modular diagram of our proposed middleware is shown in Figure 3.2. The involved devices keep track of (D)TLS sessions, PSKs, and the Device IDs. The security association occurs first by letting the Intermittent Security module try and resume a past connection, by first verifying session existence and validity. If the resumption fails, Flexible Security module acts as an interface to allow configuration of required security schemes. These two modules in conjunction form the middleware.

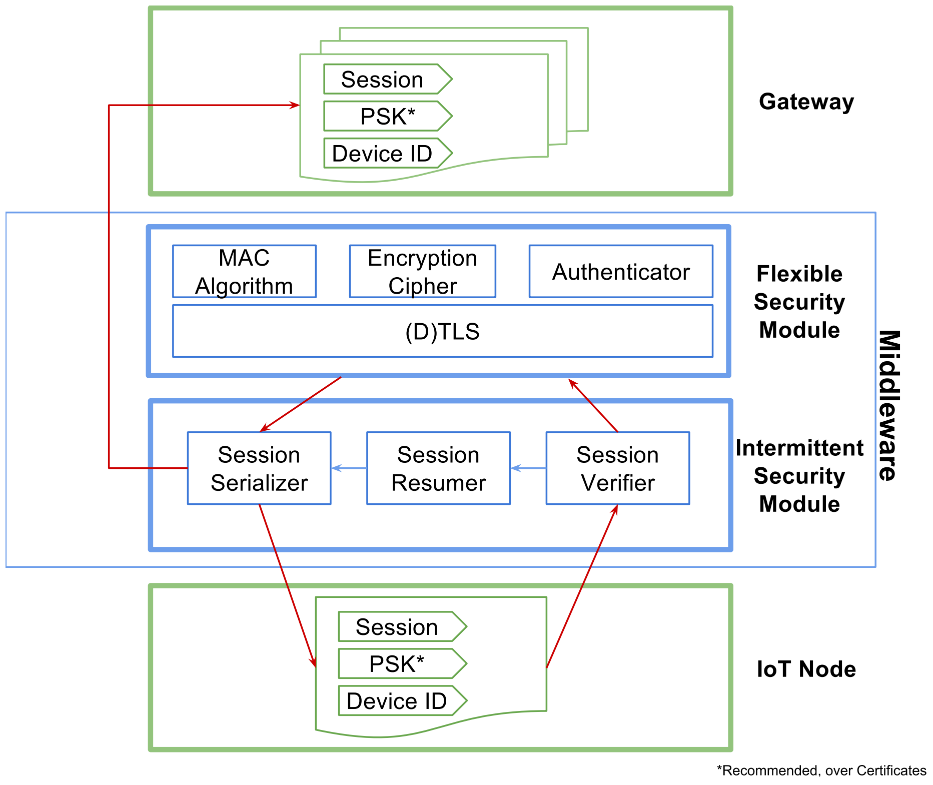


Figure 3.2: End-to-End IoT Security Middleware Module Diagram

Our middleware allows flexibility of security through various available protocols. The reasoning behind providing flexibility is because all applications and devices are not built with same level of security in mind. There is a trade-off between security and speed when it comes to a preset of a security protocol. High level of security is usually desired, but not always needed. Based on the application, it might be detrimental to have full-fledged security. For instance, if an edge beacon (based on e.g., iBeacon technology) is transmitting confidential medical information, the data security is a major priority. However, if the same beacon is to be reused for emergency medical triage, the priority for speed and low power consumption goes up, at the expense of high security.

3.1.3 Deployment

In practical IoT-based deployments, the middleware can be installed at various levels, including *System Level* and *Application Level*. A System Level installation could involve integrating the features of the middleware into the Operating System services of the device, ideally by the device manufacturer or software developer. This approach allows an application developer to incorporate the features of the middleware for customization by users. On the other hand, Application Level installation can allow an application developer to directly integrate the middleware features into the application logic. This approach could be useful if full device control is not available at the Application Level.

3.2 Intermittent Security

Intermittent security utilizes session resumption to quickly bring a disconnected device back in the network when next needed. This concept closely follows the ideas proposed in [12], and modifies a few factors. Our middleware implements intermittent security using "Device ID", instead of Host Identity Protocol (HIP). This allows compatibility with a broader range of device types. The device IDs of the edge nodes are managed by the nearest hop gateway. (D)TLS sessions are stored by the devices on disconnection for future use. If such a recently disconnected edge node attempts to make a connection with the gateway, the gateway uses the client Device IDs to determine the session to resume for that requesting node. This scheme allows security handshake steps and time to be minimized, and the data to be transmitted can still successfully be transferred, in intermittent chunks.

A possible major concern in a session resumption implementation is the possibility of Replay Attacks [14]. Given that the serialized sessions are tied to the property of the device, i.e, the Device ID, replaying using the same session is made extremely difficult by any malicious device, almost certainly having a different Device ID. To prevent an active session from being replayed by a spoofing device, a simple flag is sufficient to block such replay requests.

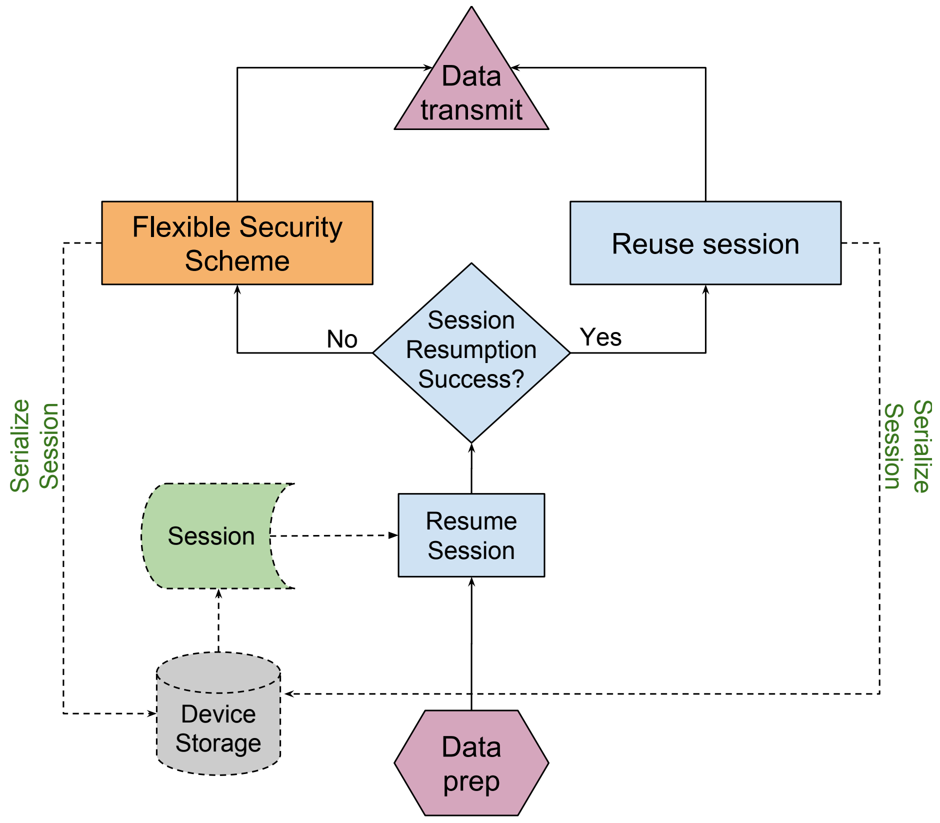


Figure 3.3: Illustration of Intermittent Security handling with Session Resumption

Data Encryption is commonly done using keys established through Public Key Cryptography (PKC). Instead of using PKC, our middleware chooses to go with static elements such as Static Pre-shared Key (PSK) or Certificates. This is because PKC can be quite slow, in addition to being intensive in terms of time, computation, bandwidth, and memory resources, [10]. Despite lacking in nonce and entropy compared to ephemeral schemes, static PSKs still are capable of providing a reasonable level of encryption using the user's choice of cipher, such as block or stream ciphers. Hence, it is a preferred scheme for most use cases, allowing a tradeoff balance between speed and security requirements of an IoT-based application [15].

Algorithm 1: Intermittent Security Handler

```
Data: Device ID $dID$. Protocol $p$, either DTLS or TLS
Data: Authentication Scheme $auth$, Encryption Scheme $en$
Data: Message Authentication Code $mac$
Data: $session$ variable holds encrypted session info
Data: $stored\_session$ holds deserialized session fetched from device storage
Data: $first\_connect$ is true if this is the first time connecting

Result: The latest session is stored on the respective devices to be quickly resumable

function initSession ()
    /* Creates a new session from specified configuration */
    $session \leftarrow flex\_security\_vector$(dID, $\{p, auth, en, mac\}$)
end
function resume ()
    /* Pulls the stored session and uses it as new session */
    $session \leftarrow stored\_session$
end
function serializeSession (x)
    /* Store the session in storage of member devices */
    while true do
        sleep (x)
        $stored\_session \leftarrow session$
    end
end
function main ()
    /* Decide and create or resume a session */
    if $firstConnect$ or $stored\_session.isNull$ then
        $initSession()$
    else
        $resume()$
    end
    $serializeSession()$
    $transmit()$
end
```

Figure 3.3 shows a flowchart illustration of how our middleware leverages intermittent security with session resumption for quick data transmission. A contingent flexible security scheme is used to quickly establish lost connections with the fastest possible way, based on the security needs of an IoT-based application.

Algorithm 1 shows our pseudocode for providing intermittent security. The *main()* function gets executed first, to check whether the connection between the associated devices is being made for the very first time. Or, if there already is a valid session corresponding to these devices. If so, we can simply fetch the stored session from device storage and attempt to resume it, allowing quick reconnection between them. If not, a new connection has to be established, plugging into the flexible security scheme, i.e., based on chosen protocol, authentication scheme, encryption scheme, and message authentication code algorithm, a new session would be initiated.

Once a session has been found (either new or resumed), two operations occur in parallel: First, *serializeSession()* ensures that the current session state is serialized to the device storage every few seconds, as represented by variable *x*. Based on the need, the value of *x* can be made higher or lower. Higher value of *x* would result in more frequent writes to the storage, providing more reliability for future session resumptions at the expense of using higher computation and storage. Conversely, less frequent writes would be less reliable, but faster and resource conservative. Second, *transmit()* keeps data flow active between the connected devices.

3.3 Flexible Security

The first step for security association and communication initiation is selection of the security protocols to be used. Our middleware supports different kinds of protocols and allows switching between them. The possible choices all select one of the options in each cate-

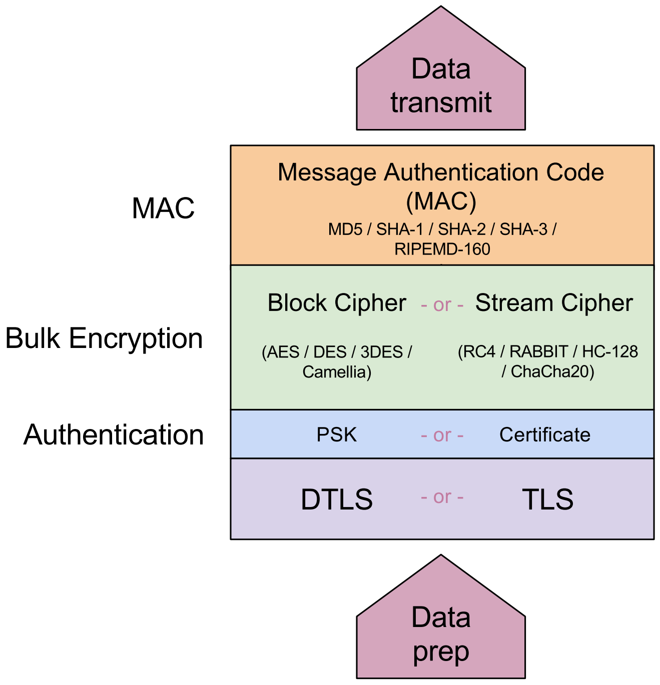

Data transmit
MAC
Message Authentication Code (MAC)
MD5 / SHA-1 / SHA-2 / SHA-3 / RIPEMD-160
Bulk Encryption
Block Cipher
- or -
Stream Cipher
(AES / DES / 3DES / Camellia)
(RC4 / RABBIT / HC-128 / ChaCha20)
Authentication
PSK
- or -
Certificate
DTLS
- or -
TLS
Data prep

Figure 3.4: Illustration of Flexible Security handling with Protocol Selection

gory. The categories include (i) Protocol, (ii) Authentication Scheme, (iii) Bulk Encryption Scheme, and (iv) Message Authentication Code (MAC) algorithm as shown in Figure 3.4.

3.3.1 Protocol Selection

The Protocol selection allows a choice between {TLS, DTLS}. TLS (Transport Layer Security) and DTLS (Datagram Transport Layer Security) are both extremely secure protocols enforcing network encryption between participants. Both of these protocols ensure confidentiality and integrity of data. DTLS is a better choice for stream-based applications, and can work over UDP (User Datagram Protocol). For use with TCP based applications, TLS is the preferred choice. The difference in performance and bandwidth requirements of TLS and DTLS can get noticeably high when adding the impact of the ideal authentication, encryption, and MAC schemes.

3.3.2 Authentication Scheme Selection

Next, Authentication Scheme can be $\{$PSK, Certificate$\}$. Furthermore, each of these entities can either $\{$Ephemeral, Static$\}$. Ephemeral PSKs or Certificates require full security handshake and key exchange before use. On the other hand, Static authentication elements do not require repeated key exchange, and hence can save a lot of time and bandwidth. In our middleware, if PSK is found to be an ideal candidate, the default setup goes for Static PSK. In fact, Static PSK can exist as a device property on the IoT devices, allowing many benefits, such as quick connection, resumption, low memory footprint, low bandwidth consumption, low CPU usage. If the requirement is for even higher security, ephemeral PSK or Certificate can be generated using Key Exchange algorithms, such as RSA, DH (Diffie-Hellman), etc.

3.3.3 Bulk Encryption Scheme Selection

Once authentication is chosen, the Bulk Encryption scheme is the next option. Encryption can be done using either Block Ciphers, or Stream Ciphers. Block Ciphers are useful for sending large chunk of data, and can consume a lot of bandwidth and memory if the payload is small. This is due to the padding added to each block of data being sent. For example, AES uses 128-bit (16-byte) padding by default. If the data being transmitted is 1024-bit in length, then the total packets sent would be $\lceil 1024/128 \rceil = 8$. But, if the data size is 130-bit, the number of packets sent would be $\lceil 130/128 \rceil = 2$. The second packet would have 126 empty reserved bits. Hence, for small payload applications (such as video streams) it is better to opt for Stream Cipher, which encrypts small chunks of data before sending. The most common Stream Cipher is RC4, but ChaCha20 is starting to take over as the next generation of much faster and more secure stream ciphers. All ciphers can be configured to various key sizes (if applicable), including 128-bit, 224-bit, 256-bit, and so on.

3.3.4 Message Authentication Code Selection

Lastly, the chosen Message Authentication Code algorithm is used to generate checksum, to ensure integrity of data being sent. The available options are MD5, SHA {1/2/3}, and a few lesser-used options. SHA2 or SHA3 should be used whenever possible, since MD5 and SHA1 have been found vulnerable to various checksum attacks [16, 17] and collision attacks [18]. Through permutation and combination, the possible choices for the security scheme can be many. Table 3.1 shows a few of the possible schemes.

Table 3.1: Different security schemes for *flexible security*

Security Scheme	Protocol	Authentication	Encryption	MAC	Description
DTLS_PSK_WITH_CHACHA20_SHA256	DTLS	Static PSK	ChaCha20	SHA2(256)	Very fast, secure. Excellent for secure video streaming
DTLS_DHE_WITH_NULL_SHA384	DTLS	Certificate	-	SHA2(384)	Fast scheme, high security
DTLS_DHE_PSK_WITH_3DES_EDE_SHA	DTLS	PSK	3DES (EDE)	SHA1	Fast, but risk of integrity loss due to SHA1.
TLS_PSK_WITH_AES_128_CBC_SHA	TLS	Static PSK	AES128(CBC)	SHA1	Fast, highly secure, suitable for moderately heavy data
TLS_PSK_WITH_CHACHA20_POLY1305	TLS	Static PSK	ChaCha20	POLY1305	Fast, highly secure, suitable for quick bulk data transfer
TLS_ECDHE_WITH_AES_256_GCM_SHA384	TLS	Certificate	AES256(GCM)	SHA2(384)	Very high security, suitable for confidential data on a reliable network

3.4 Optimal Scheme Decider

As discussed in the previous sections, our middleware allows picking and choosing pieces of security protocols as per the user's requirement. But when it comes to actually choosing the best scheme for an application, the decision is hard to make. It becomes even more of a challenge due to the fact that not every user might be well versed with the various security components, or have a strong understanding of the differences between the schemes, their advantages, disadvantages, etc. Hence, just having the option to choose is not enough.

This is where the optimal scheme decider comes in. With close to 200 possible security scheme choices, the decider is able to find the optimal choice of security scheme for any application, IoT-based or otherwise. This is done in two phases: (i) *Offline Phase*, and (ii) *Online Phase*.

3.4.1 Offline Phase

The Offline Phase is a step used to narrow down the searchable space of security schemes by filtering a database of around 200 schemes. Although schemes can be quite varied, but there exist many schemes that have close similarities in all practical aspects. Moreover, it is possible to find alternative schemes to any chosen scheme when the requirements and priorities are adjusted according to the needs of the application. Hence, it turns out, every

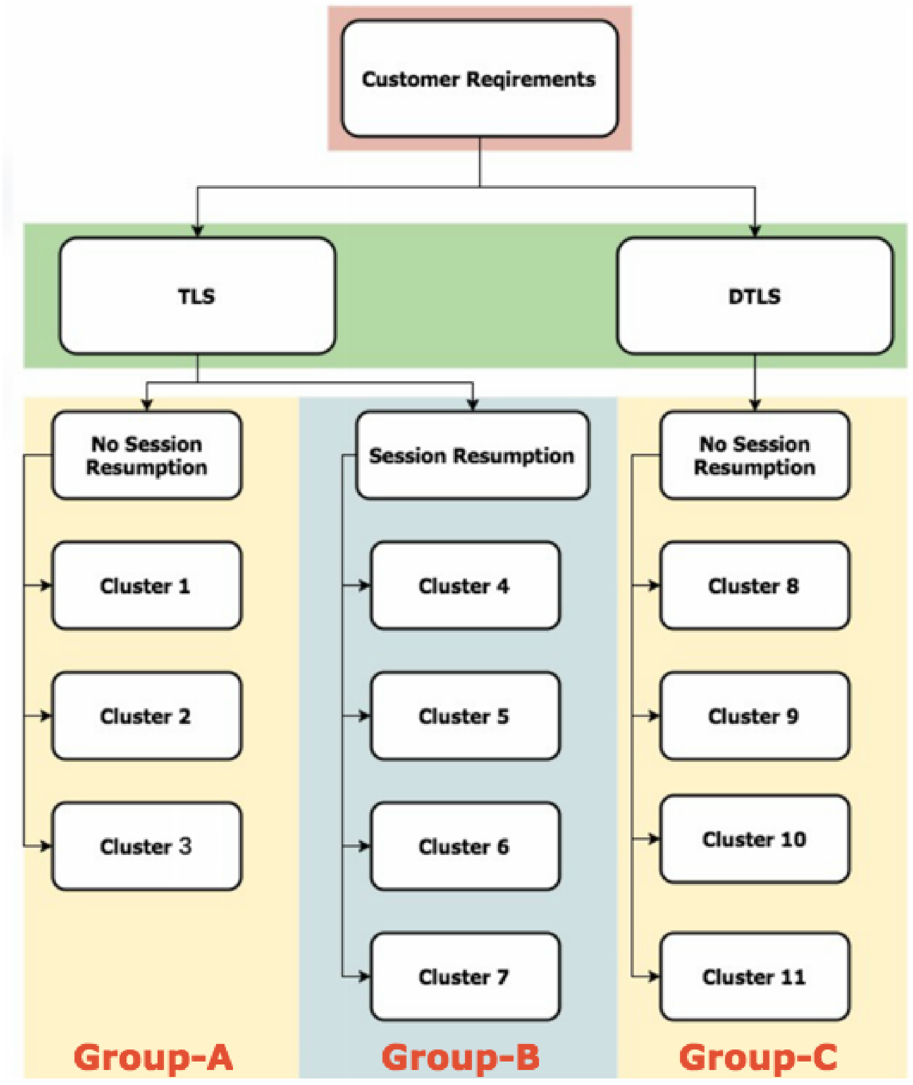


Figure 3.5: Scheme Groups based on configuration

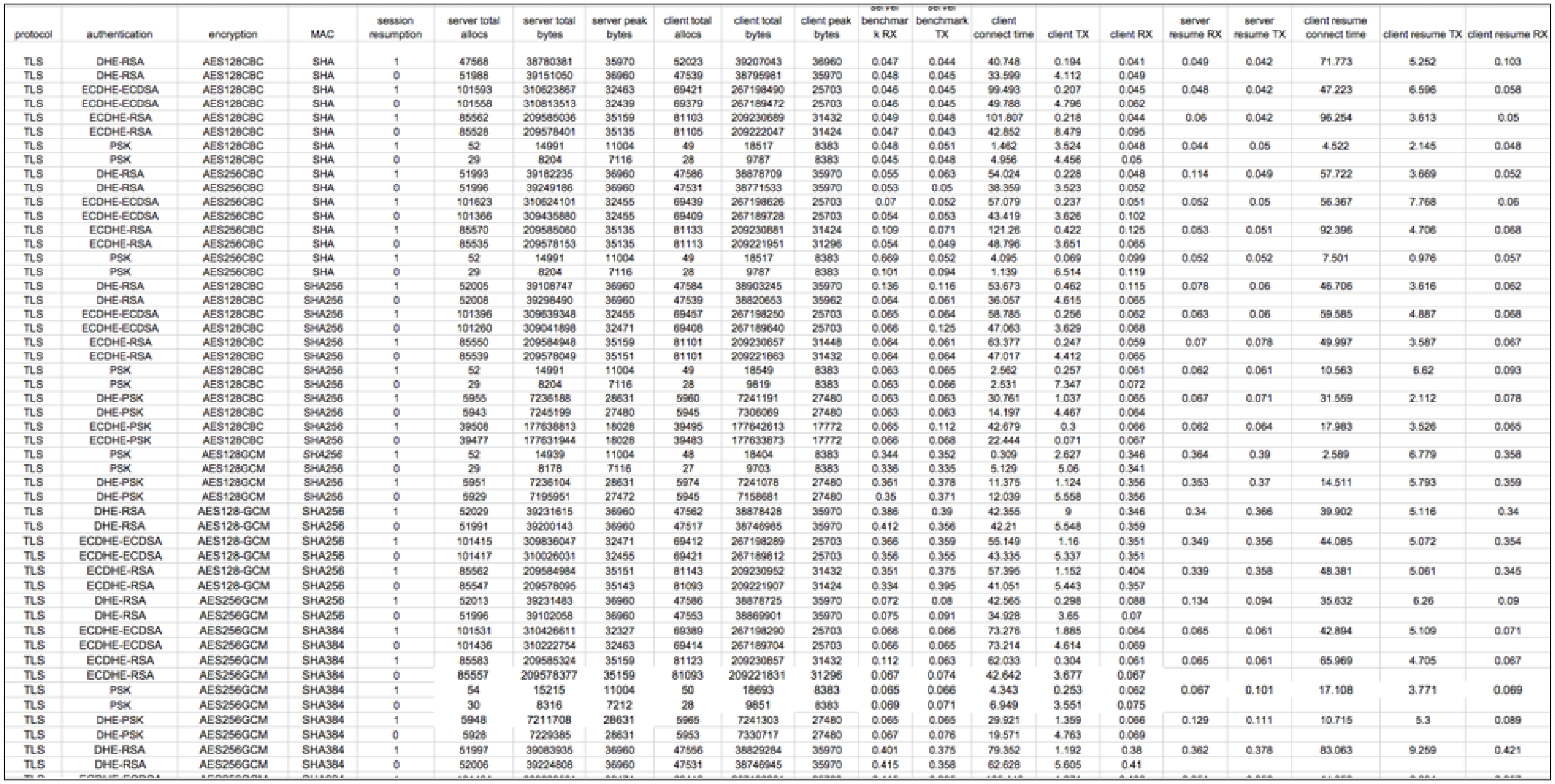
permutation or combination of protocols is not a necessary option for all IoT application developers. In fact, as shown in Figure 3.5, we are able to group security schemes based on their protocol choices.

In the current implementation, three groups {Group-A, Group-B, Group-C} have been formed. Group-A represents all configurations utilizing TLS as the base protocol, and using no session resumption. Group-B represents all configurations under TLS, but with support for session resumption. Group-C houses all DTLS schemes with no session resumption. A fourth group, Group-D can additionally be formed for all DTLS schemes with session resumption. The logic from start to end of the *offline-phase* will be the same for all groups, and hence will also be applicable for Group-D.

The reason why Session Resumption has been made an optional configuration is because there is a trade-off of having the feature. The resumption feature itself requires some periodic disk I/O to serialize active session. In addition, the serialized files are stored on the device, utilizing some disk space. This process also ends up consuming some energy. In most cases, these factors are trivial. But for certain extremely limited IoT devices, the middleware's option to not support session resumption can prove beneficial.

Figure 3.6 shows a part of a table with raw benchmark data of many possible scheme choices. The data is organized by protocol {TLS, DTLS}, authentication schemes {DHE-RSA, PSK, ECDSA, etc.}, encryption schemes {AES, ChaCha20, etc.}, MAC {SHA, MD5, etc.}, and session resumption support. The benchmarking is done for total memory allocations on server and client, total bytes transferred on server and client, connect times, resume times, and many similar parameters.

Since our server is considered a FFD (Full Function Device), it can be safely assumed that client benchmarks are the bottleneck, and hence sufficient for forming clusters. Since the clusters are to choose the optimal security scheme targeted for the IoT-devices involved, we are able to discard the benchmark values for the server side without any negative impact on the decider.



protocol	authentication	encryption	MAC	session resumption	server total allocs	server total bytes	server peak bytes	client total allocs	client total bytes	client peak bytes	server benchmar k RX	server benchmark TX	client connect time	client TX	client RX	server resume RX	server resume TX	client resume connect time	client resume TX	client resume RX
TLS	DHE-RSA	AES128CBC	SHA	1	47568	38780381	35970	52023	39207043	36960	0.047	0.044	40.748	0.194	0.041	0.049	0.042	71.773	5.252	0.103
TLS	DHE-RSA	AES128CBC	SHA	0	51988	39151050	36960	47539	38795981	35970	0.048	0.045	33.599	4.112	0.049					
TLS	ECDHE-ECDSA	AES128CBC	SHA	1	101593	310623867	32463	69421	267198490	25703	0.046	0.045	99.493	0.207	0.045	0.048	0.042	47.223	6.596	0.058
TLS	ECDHE-ECDSA	AES128CBC	SHA	0	101558	310813513	32439	69379	267189472	25703	0.046	0.045	49.788	4.796	0.062					
TLS	ECDHE-RSA	AES128CBC	SHA	1	85562	209585036	35159	81103	209230689	31432	0.049	0.048	101.807	0.218	0.044	0.06	0.042	96.254	3.613	0.05
TLS	ECDHE-RSA	AES128CBC	SHA	0	85528	209578401	35135	81105	209222047	31424	0.047	0.043	42.852	8.479	0.096					
TLS	PSK	AES128CBC	SHA	1	52	14991	11004	49	18517	8383	0.048	0.051	1.462	3.524	0.048	0.044	0.05	4.522	2.145	0.048
TLS	PSK	AES128CBC	SHA	0	29	8204	7116	28	9787	8383	0.045	0.048	4.956	4.456	0.05					
TLS	DHE-RSA	AES256CBC	SHA	1	51993	39182235	36960	47586	38878709	35970	0.055	0.063	54.024	0.228	0.048	0.114	0.049	57.722	3.669	0.052
TLS	DHE-RSA	AES256CBC	SHA	0	51996	39249186	36960	47531	38771533	35970	0.053	0.05	38.359	3.523	0.052					
TLS	ECDHE-ECDSA	AES256CBC	SHA	1	101623	310624101	32455	69439	267198626	25703	0.07	0.052	57.079	0.237	0.051	0.052	0.05	56.367	7.768	0.06
TLS	ECDHE-ECDSA	AES256CBC	SHA	0	101366	309435880	32455	69409	267189728	25703	0.054	0.053	43.419	3.626	0.102					
TLS	ECDHE-RSA	AES256CBC	SHA	1	85570	209585060	35135	81133	209230881	31424	0.109	0.071	121.26	0.422	0.125	0.053	0.051	92.396	4.706	0.068
TLS	ECDHE-RSA	AES256CBC	SHA	0	85535	209578153	35135	81113	209221951	31296	0.054	0.049	48.796	3.651	0.065					
TLS	PSK	AES256CBC	SHA	1	52	14991	11004	49	18517	8383	0.669	0.052	4.095	0.069	0.099	0.052	0.052	7.501	0.976	0.057
TLS	PSK	AES256CBC	SHA	0	29	8204	7116	28	9787	8383	0.101	0.094	1.139	6.514	0.119					
TLS	DHE-RSA	AES128CBC	SHA256	1	52005	39108747	36960	47584	38903245	35970	0.136	0.116	53.673	0.462	0.115	0.078	0.06	46.706	3.616	0.062
TLS	DHE-RSA	AES128CBC	SHA256	0	52008	39298490	36960	47539	38820653	35962	0.064	0.061	36.057	4.615	0.065					
TLS	ECDHE-ECDSA	AES128CBC	SHA256	1	101396	309639348	32455	69457	267198250	25703	0.065	0.064	58.785	0.256	0.062	0.063	0.06	59.585	4.887	0.068
TLS	ECDHE-ECDSA	AES128CBC	SHA256	0	101260	309041898	32471	69408	267189640	25703	0.066	0.125	47.063	3.629	0.068					
TLS	ECDHE-RSA	AES128CBC	SHA256	1	85550	209584948	35159	81101	209230657	31448	0.064	0.061	63.377	0.247	0.059	0.07	0.078	49.997	3.587	0.067
TLS	ECDHE-RSA	AES128CBC	SHA256	0	85539	209578049	35151	81101	209221863	31432	0.064	0.064	47.017	4.412	0.065					
TLS	PSK	AES128CBC	SHA256	1	52	14991	11004	49	18549	8383	0.063	0.065	2.562	0.257	0.061	0.062	0.061	10.563	6.62	0.093
TLS	PSK	AES128CBC	SHA256	0	29	8204	7116	28	9819	8383	0.063	0.066	2.531	7.347	0.072					
TLS	DHE-PSK	AES128CBC	SHA256	1	5955	7236188	28631	5960	7241191	27480	0.063	0.063	30.761	1.037	0.065	0.067	0.071	31.559	2.112	0.078
TLS	DHE-PSK	AES128CBC	SHA256	0	5943	7245199	27480	5945	7306069	27480	0.063	0.063	14.197	4.467	0.064					
TLS	ECDHE-PSK	AES128CBC	SHA256	1	39508	177638813	18028	39495	177642613	17772	0.065	0.112	42.679	0.3	0.066	0.062	0.064	17.983	3.526	0.065
TLS	ECDHE-PSK	AES128CBC	SHA256	0	39477	177631944	18028	39483	177633873	17772	0.066	0.068	22.444	0.071	0.067					
TLS	PSK	AES128GCM	SHA256	1	52	14939	11004	48	18404	8383	0.344	0.352	0.309	2.627	0.346	0.364	0.39	2.589	6.779	0.358
TLS	PSK	AES128GCM	SHA256	0	29	8178	7116	27	9703	8383	0.336	0.335	5.129	5.06	0.341					
TLS	DHE-PSK	AES128GCM	SHA256	1	5951	7236104	28631	5974	7241078	27480	0.361	0.378	11.375	1.124	0.356	0.353	0.37	14.511	5.793	0.359
TLS	DHE-PSK	AES128GCM	SHA256	0	5929	7195951	27472	5945	7158681	27480	0.35	0.371	12.039	5.558	0.356					
TLS	DHE-RSA	AES128-GCM	SHA256	1	52029	39231615	36960	47562	38878428	35970	0.386	0.39	42.355	9	0.346	0.34	0.366	39.902	5.116	0.34
TLS	DHE-RSA	AES128-GCM	SHA256	0	51991	39200143	36960	47517	38746985	35970	0.412	0.356	42.21	5.548	0.359					
TLS	ECDHE-ECDSA	AES128-GCM	SHA256	1	101415	309836047	32471	69412	267198289	25703	0.366	0.359	55.149	1.16	0.351	0.349	0.356	44.085	5.072	0.354
TLS	ECDHE-ECDSA	AES128-GCM	SHA256	0	101417	310026031	32455	69421	267189612	25703	0.356	0.355	43.335	5.337	0.351					
TLS	ECDHE-RSA	AES128-GCM	SHA256	1	85562	209584984	35151	81143	209230952	31432	0.351	0.375	57.395	1.152	0.404	0.339	0.358	48.381	5.061	0.345
TLS	ECDHE-RSA	AES128-GCM	SHA256	0	85547	209578095	35143	81093	209221907	31424	0.334	0.395	41.051	5.443	0.357					
TLS	DHE-RSA	AES256GCM	SHA256	1	52013	39231483	36960	47586	38878725	35970	0.072	0.08	42.565	0.298	0.088	0.134	0.094	35.632	6.26	0.09
TLS	DHE-RSA	AES256GCM	SHA256	0	51996	39102058	36960	47553	38869901	35970	0.075	0.091	34.928	3.65	0.07					
TLS	ECDHE-ECDSA	AES256GCM	SHA384	1	101531	310426611	32327	69389	267198290	25703	0.066	0.066	73.276	1.885	0.064	0.065	0.061	42.894	5.109	0.071
TLS	ECDHE-ECDSA	AES256GCM	SHA384	0	101436	310222754	32463	69414	267189704	25703	0.066	0.065	73.214	4.614	0.069					
TLS	ECDHE-RSA	AES256GCM	SHA384	1	85583	209585324	35159	81123	209230857	31432	0.112	0.063	62.033	0.304	0.061	0.065	0.061	65.969	4.705	0.067
TLS	ECDHE-RSA	AES256GCM	SHA384	0	85557	209578377	35159	81093	209221831	31296	0.067	0.074	42.642	3.677	0.067					
TLS	PSK	AES256GCM	SHA384	1	54	15215	11004	50	18693	8383	0.065	0.066	4.343	0.253	0.062	0.067	0.101	17.108	3.771	0.069
TLS	PSK	AES256GCM	SHA384	0	30	8316	7212	28	9851	8383	0.069	0.071	6.949	3.551	0.075					
TLS	DHE-PSK	AES256GCM	SHA384	1	5948	7211708	28631	5965	7241303	27480	0.065	0.065	29.921	1.359	0.066	0.129	0.111	10.715	5.3	0.089
TLS	DHE-PSK	AES256GCM	SHA384	0	5928	7229385	28631	5963	7330717	27480	0.067	0.076	19.571	4.763	0.069					
TLS	DHE-RSA	AES256GCM	SHA384	1	51997	39063935	36960	47556	38829284	35970	0.401	0.375	79.352	1.192	0.38	0.362	0.378	83.063	9.259	0.421
TLS	DHE-RSA	AES256GCM	SHA384	0	52006	39224808	36960	47531	38746945	35970	0.415	0.358	62.628	5.605	0.41					

Figure 3.6: Raw benchmark data for various security schemes

Table 3.2: Parameters used for client benchmark analysis

Client Parameter	Description	Desired Level for RFD
CPU Usage	The security scheme's CPU usage	Smaller value
Bandwidth Usage	The bandwidth of the security schemes network usage	Smaller value
Peak Bytes	The peak memory consumption on device	Smaller value
Connection Time	The elapsed time to successfully connect with server	Smaller value
TX	The speed of transmitting data	Higher value
RX	The speed of receiving data	Higher value

Table 3.2 shows the parameters used for client benchmark analysis. The desired level represents whether high value is optimal, or lower, when operating on a Reduced Function Device. These 6 metrics can next be used to form the clusters. To move forward, we need to perform a set of operations on the raw data to successfully filter the schemes into appropriate clusters under each group. Hence, the following steps are followed:

1. Scale the raw data
2. Identify Principal Components and generate weighted formula for clustering
3. Calculate the weight value for data elements
4. Identify number of clusters needed per group
5. Final clustering

1. **Scale the raw data**

   This step is needed to decrease the correlation between each metric, in addition to decreasing the influence of units, i.e., Normalization. Hence, we perform data scaling on the raw data in Table 3.6 using the scaling formula:

$$x' = \frac{x_{ij} - x_{mj}}{\sigma_j}$$

where

x'	: scaled value
x_{ij}	: value of i^{th} row, j^{th} column data
x_{mj}	: mean value of j column data
σ_j	: standard variance for the j column data

2. **Identify Principal Components and generate weighted formula for clustering**

Next step involves applying Principal Component Analysis (PCA) to decide the number of clusters that may exist in each for the groups {Group-A, Group-B, Group-C}. The formula for each potential cluster can be as below:

$$z_{i1} = \phi_{11} \cdot x_{i1} + \phi_{12} \cdot x_{i2} + \cdots + \phi_{1p} \cdot x_{ip}$$
$$z_{i2} = \phi_{21} \cdot x_{i1} + \phi_{22} \cdot x_{i2} + \cdots + \phi_{2p} \cdot x_{ip}$$
$$\cdots$$
$$z_{ip} = \phi_{p1} \cdot x_{i1} + \phi_{p2} \cdot x_{i2} + \cdots + \phi_{pp} \cdot x_{ip}$$

where

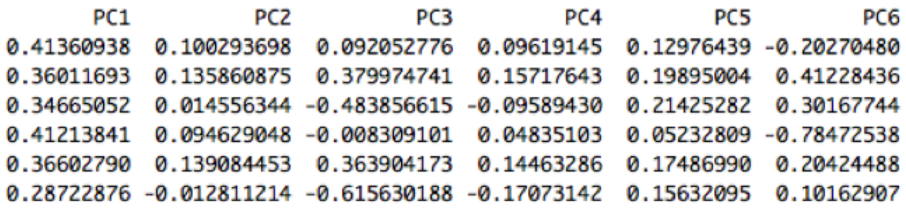
z_{ip}	: value of p^{th} principal component
ϕ_{p1}	: weight of each element
x_{ip}	: value of element

3. **Calculate the weight value for data elements**

Once we have the principal component formula, we need to calculate the value of weight ϕ of every element. This is an optimization problem to calculate the weight ϕ:

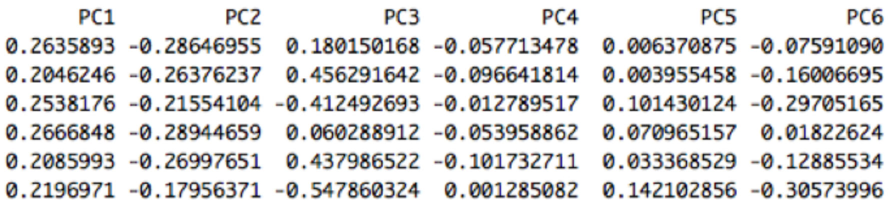
$$\begin{aligned} &\underset{\phi_{11},\cdots,\phi_{p1}}{\text{maximize}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1} \cdot x_{ij} \right)^2 \right\} \\ &\text{subject to} \qquad\qquad\qquad \sum_{j=1}^{p} \phi_{j1}^2 = 1. \end{aligned}$$

Figure 3.7 shows our results for Principal Components of Group-A. Every column is now categorized as per metric value similarity. For example, *PC1* might have all schemes with maximum CPU consumption, *PC2* might represent all schemes with maximum network bandidth, and so on. Similarly, Principal Component Analysis generates the data shown in Figure 3.8 and Figure 3.9 for Group-B and Group-C, respectively.



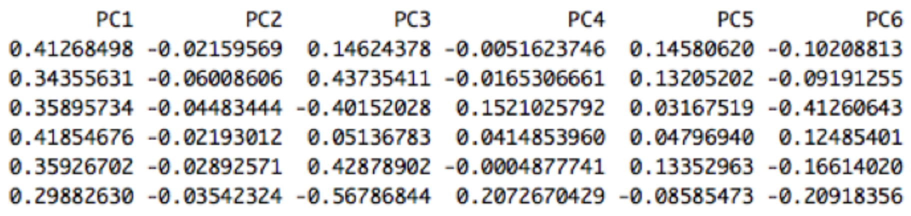
```
       PC1          PC2          PC3         PC4        PC5         PC6
0.41360938  0.100293698  0.092052776  0.09619145 0.12976439 -0.20270480
0.36011693  0.135860875  0.379974741  0.15717643 0.19895004  0.41228436
0.34665052  0.014556344 -0.483856615 -0.09589430 0.21425282  0.30167744
0.41213841  0.094629048 -0.008309101  0.04835103 0.05232809 -0.78472538
0.36602790  0.139084453  0.363904173  0.14463286 0.17486990  0.20424488
0.28722876 -0.012811214 -0.615630188 -0.17073142 0.15632095  0.10162907
```

Figure 3.7: Group-A Principal Component Results



```
      PC1         PC2          PC3          PC4         PC5         PC6
0.2635893 -0.28646955  0.180150168 -0.057713478 0.006370875 -0.07591090
0.2046246 -0.26376237  0.456291642 -0.096641814 0.003955458 -0.16006695
0.2538176 -0.21554104 -0.412492693 -0.012789517 0.101430124 -0.29705165
0.2666848 -0.28944659  0.060288912 -0.053958862 0.070965157  0.01822624
0.2085993 -0.26997651  0.437986522 -0.101732711 0.033368529 -0.12885534
0.2196971 -0.17956371 -0.547860324  0.001285082 0.142102856 -0.30573996
```

Figure 3.8: Group-B Principal Component Results



```
       PC1         PC2         PC3           PC4         PC5         PC6
0.41268498 -0.02159569  0.14624378 -0.0051623746  0.14580620 -0.10208813
0.34355631 -0.06008606  0.43735411 -0.0165306661  0.13205202 -0.09191255
0.35895734 -0.04483444 -0.40152028  0.1521025792  0.03167519 -0.41260643
0.41854676 -0.02193012  0.05136783  0.0414853960  0.04796940  0.12485401
0.35926702 -0.02892571  0.42878902 -0.0004877741  0.13352963 -0.16614020
0.29882630 -0.03542324 -0.56786844  0.2072670429 -0.08585473 -0.20918356
```

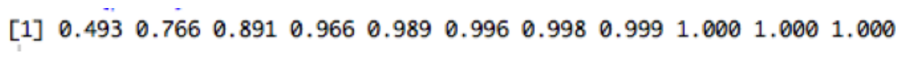
Figure 3.9: Group-C Principal Component Results

4. **Identify number of clusters needed per group**

The number of clusters that need to be generated per group can next be calculated, using Proportion of Variance Explained (PVE). We apply the following on the dataset:

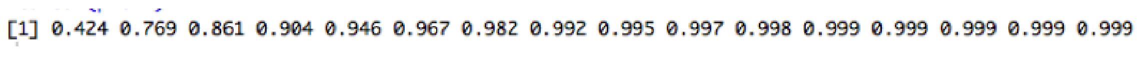
$$\sum_{j=1}^{p} \operatorname{Var}\left(X_j\right) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2$$

The PVE result for each group is shown in Figures 3.10, 3.11, and 3.12. Analyzing Group-A PVE in Figure 3.10 tells us that 89.1% ($\sim$ 90%) of the schemes in this group can be taken into consideration if we use **three** clusters. Any higher cluster count would provide only trivial advantage. For Group-B, the PVE in Figure 3.11 suggests taking **four** clusters to encompass 90.4% ($\sim$ 90%) of the schemes. Similarly, Group-C can benefit by housing **four** clusters, i.e., 90.6% ($\sim$ 90%) of the schemes. Hence, we accept 3 clusters in Group-A, and 4 clusters each in Group-B and Group-C.



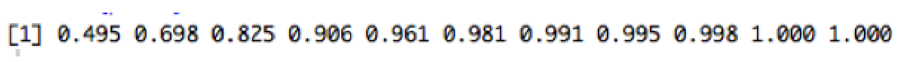
```
[1] 0.493 0.766 0.891 0.966 0.989 0.996 0.998 0.999 1.000 1.000 1.000
```

Figure 3.10: Group-A Proportion of Variance Explained results



```
[1] 0.424 0.769 0.861 0.904 0.946 0.967 0.982 0.992 0.995 0.997 0.998 0.999 0.999 0.999 0.999 0.999
```

Figure 3.11: Group-B Proportion of Variance Explained results



```
[1] 0.495 0.698 0.825 0.906 0.961 0.981 0.991 0.995 0.998 1.000 1.000
```

Figure 3.12: Group-C Proportion of Variance Explained results

5. **Final clustering**

   Once we have decided on number of clusters in each group, we can use k-means clustering by calculating Euclidean distance for each observation. This can be done by applying the following:

$$W\left(C_k\right) = \frac{1}{\left|C_k\right|} \sum_{i,i' \varepsilon C_k} \sum_{j=1}^{p} \left(x_{ij} - x_{i'j}\right)^2 . \tag{3.1}$$

   We want to cluster together the observations with minimum distance between them. Hence, we find the minimum distance in Equation 3.1:

$$\underset{C_1,\cdots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W\left(C_k\right) \right\} .$$

In essence, clustering allows us to reduce the overhead of storing hundreds of redundant security schemes, and instead finds the best alternative from smaller scope containing equivalent schemes. We are able to choose the most optimal cluster in each group, hence allowing a small subset of schemes under the hood. These subsets should ideally be able to provide all the benefits of equivalent schemes.

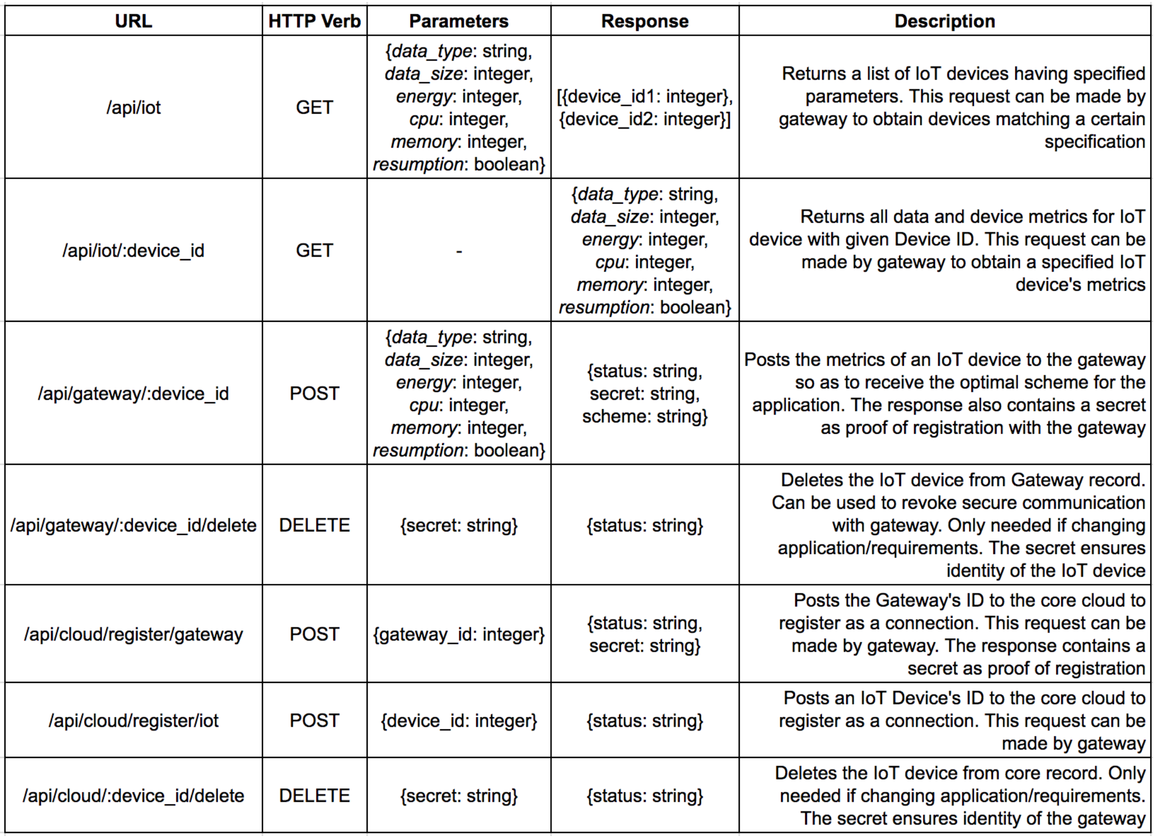
3.4.2 Online Phase

Once the offline-phase has been executed at the development stage of our middleware, its job is done. Unless any major changes are required in the available schemes, such as addition or removal of new/old schemes, the generated set of can be reused by the middleware for finding the optimal scheme, when being run by a potential user of the middleware. The online phase can be accessed using a set of RESTful APIs for the middleware on the IoT device, the gateway, and the core cloud. Figure 3.13 shows a table of available APIs for use. The *Decider* is our decision making engine which finds the optimal security scheme for our use-case. But first, it needs to collect the metrics to find the best-match security scheme as needed by the application. The metrics taken into consideration are:

- Type of Data
- Estimated size of data transfer
- Energy available to the device
- Computation power available
- Memory available
- Session Resumption requirement

The options chosen by the user are each internally mapped to a value in the range $\{1,2,\cdots,10\}$. The mapping has been done by surveying many devices ranging from lower end of the spectrum in the metric to the higher end. For example, CPU frequency of over 750 MHz can be safely categorized as a Level-4 device, considering it is higher than most IoT devices. a full description of the acceptable parameter values can be viewed in Figure 3.14.

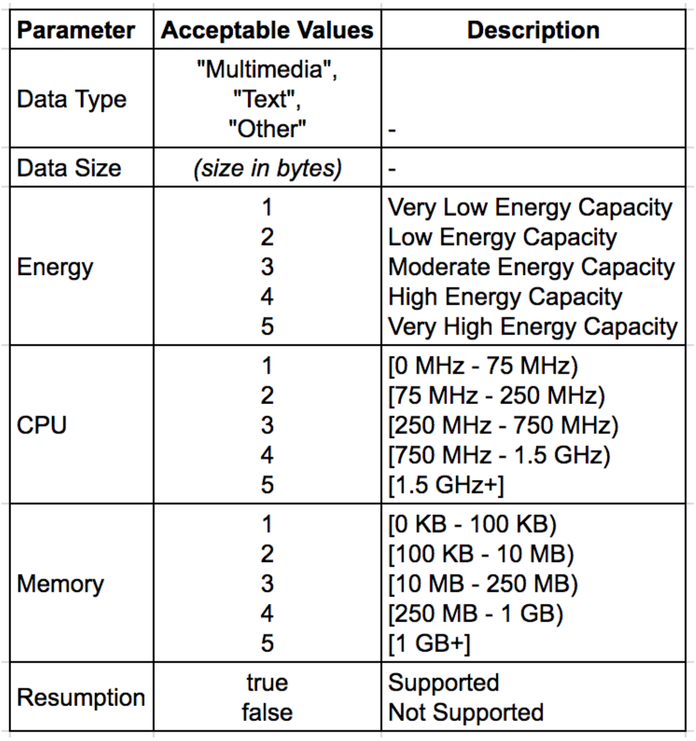
Once the internal map has been generated, the decider algorithm kicks in to filter good candidate schemes from the cluster chosen in offline-phase. Algorithm 2 describes the various filters applied to narrow down to a single security scheme from the input metrics. We have decided to prefer Stream Cipher over Block Cipher for all multimedia applications, as well as for general applications with smaller payloads. For further filtering, we have tied DTLS protocol with STREAM ciphers, and TLS protocol with BLOCK ciphers. This is because utilizing TLS with STREAM ciphers or DTLS with BLOCK ciphers would essentially undo the benefits offered by the protocol or cipher.



URL	HTTP Verb	Parameters	Response	Description
/api/iot	GET	{*data_type*: string, *data_size*: integer, *energy*: integer, *cpu*: integer, *memory*: integer, *resumption*: boolean}	[{device_id1: integer}, {device_id2: integer}]	Returns a list of IoT devices having specified parameters. This request can be made by gateway to obtain devices matching a certain specification
/api/iot/:device_id	GET	-	{*data_type*: string, *data_size*: integer, *energy*: integer, *cpu*: integer, *memory*: integer, *resumption*: boolean}	Returns all data and device metrics for IoT device with given Device ID. This request can be made by gateway to obtain a specified IoT device's metrics
/api/gateway/:device_id	POST	{*data_type*: string, *data_size*: integer, *energy*: integer, *cpu*: integer, *memory*: integer, *resumption*: boolean}	{status: string, secret: string, scheme: string}	Posts the metrics of an IoT device to the gateway so as to receive the optimal scheme for the application. The response also contains a secret as proof of registration with the gateway
/api/gateway/:device_id/delete	DELETE	{secret: string}	{status: string}	Deletes the IoT device from Gateway record. Can be used to revoke secure communication with gateway. Only needed if changing application/requirements. The secret ensures identity of the IoT device
/api/cloud/register/gateway	POST	{gateway_id: integer}	{status: string, secret: string}	Posts the Gateway's ID to the core cloud to register as a connection. This request can be made by gateway. The response contains a secret as proof of registration
/api/cloud/register/iot	POST	{device_id: integer}	{status: string}	Posts an IoT Device's ID to the core cloud to register as a connection. This request can be made by gateway
/api/cloud/:device_id/delete	DELETE	{secret: string}	{status: string}	Deletes the IoT device from core record. Only needed if changing application/requirements. The secret ensures identity of the gateway

Figure 3.13: RESTful APIs for Online Phase

In the algorithm, once the data type and size has been used to filter according to protocol and cipher type, we make the remainder of the decisions based on the device specifications. The priority is given to energy level of the device in question, since IoT-based applications are almost invariably limited by energy consumption or availability to securely handle data. Hence, low energy availability and extremely-high security scheme will not be clubbed together, being a recipe for disaster. Next, the best choice of scheme is picked based on the limiting value between available CPU and available Memory. For instance, a Level-2 CPU in conjunction with Level-9 Memory will still be incapable of running high security schemes. The same would be true if the CPU and Memory levels are reversed. Finally, the result generated is the choice of scheme to be used, which gets used by the transmitter program to generate the appropriate secure session.



Parameter	Acceptable Values	Description
Data Type	"Multimedia", "Text", "Other"	-
Data Size	*(size in bytes)*	-
Energy	1 2 3 4 5	Very Low Energy Capacity Low Energy Capacity Moderate Energy Capacity High Energy Capacity Very High Energy Capacity
CPU	1 2 3 4 5	[0 MHz - 75 MHz) [75 MHz - 250 MHz) [250 MHz - 750 MHz) [750 MHz - 1.5 GHz) [1.5 GHz+]
Memory	1 2 3 4 5	[0 KB - 100 KB) [100 KB - 10 MB) [10 MB - 250 MB) [250 MB - 1 GB) [1 GB+]
Resumption	true false	Supported Not Supported

Figure 3.14: RESTful API Parameter Values

Algorithm 2: Optimal Security Scheme Decider

```
Data: Data to be transmitted $data$
Data: Protocol to be used for transmission $protocol$
Data: Cipher to be used for encryption $mac$
Data: Energy Level classification of the device $energyLevel$
Data: CPU level classification of the device $cpuLevel$
Data: Memory level classification of the device $memLevel$

Result: The best security scheme is chosen

/* data.SIZE in bits */
 if $data.TYPE = MULTIMEDIA$ or $data.SIZE < 128$ then
    $protocol \leftarrow DTLS$
    $cipher$.TYPE $\leftarrow STREAM$
else
    $protocol \leftarrow TLS$
    $cipher$.TYPE $\leftarrow BLOCK$
end

if $energyLevel < 6$ then
    Eliminate heavy encryption schemes
end

if $\min(cpuLevel, memLevel) < 4$ then
    Choose low-level security scheme
else if $\min(cpuLevel, memLevel) < 7$ then
    Choose medium-level security scheme
else
    Choose high-level security scheme
end
```

Chapter 4

Middleware Testbed and Evaluation

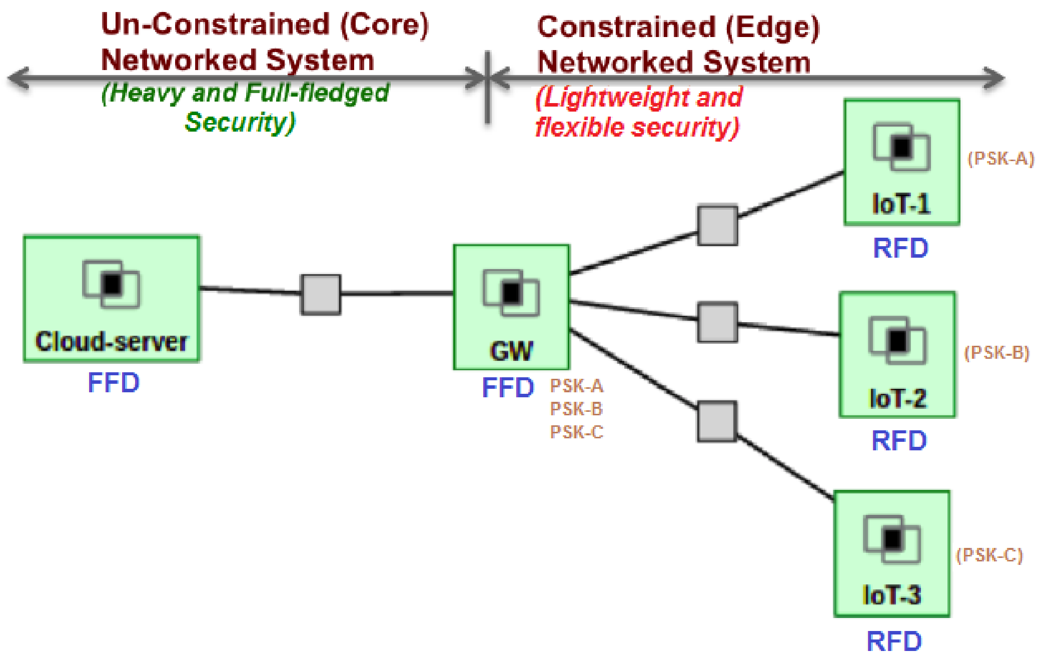
In this section, we compare the performance of various schemes accessible on our middleware to randomly selected schemes. This allows us to check the difference in impact caused by a better selection. Since the middleware has multiple submodules capable of working independently, we perform our middleware evaluation using two test cases: (i) Impact Test of Flexible and Intermittent Security, and (ii) Test of Optimal Scheme Decider.

4.1 Case I: Impact Test of Flexible and Intermittent Security

This test aims to check the impact of utilizing the middleware to switch from ephemeral, high security protocol schemes to using static properties such as PSK for secure session. This test accounts for: (i) Memory Footprint, including number of memory allocations and total size of allocation, and (ii) Time taken for security association, for initial session establishment and session resumption scenario. A minimum viable implementation of the middleware has been used on a GENI [19] Cloud testbed.

Figure 4.1 shows the network setup using the GENI Cloud infrastructure.

The first step for security association and communication initiation is selection of the



Un-Constrained (Core) Networked System
(Heavy and Full-fledged Security)
Constrained (Edge) Networked System
(Lightweight and flexible security)
Cloud-server
FFD
GW
FFD
PSK-A
PSK-B
PSK-C
IoT-1
RFD
(PSK-A)
IoT-2
RFD
(PSK-B)
IoT-3
RFD
(PSK-C)

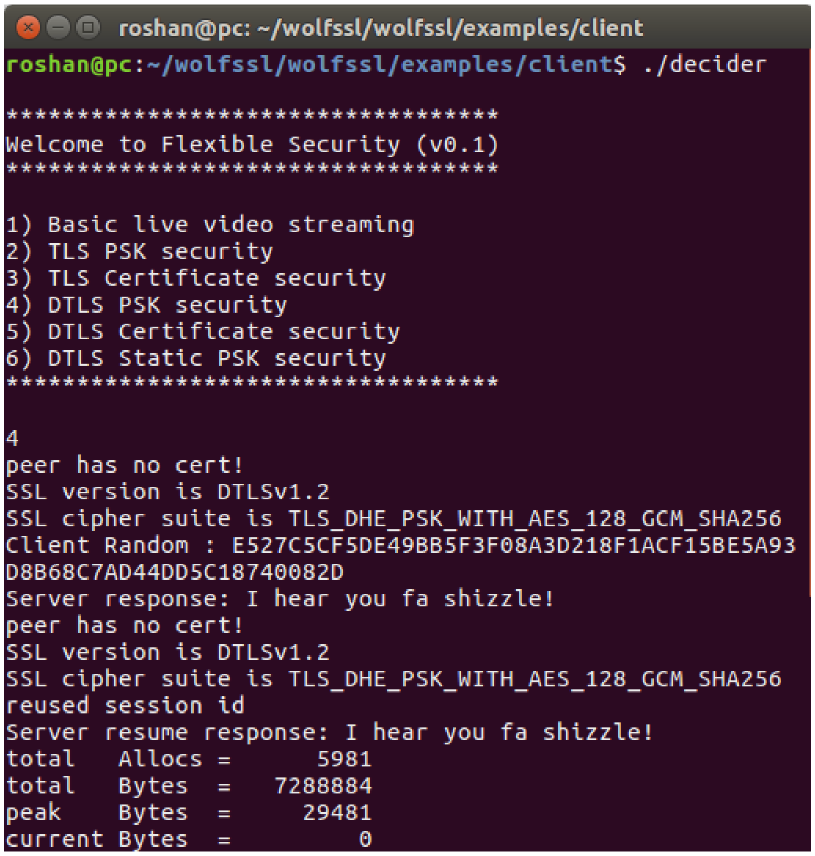
Figure 4.1: Testbed setup over the GENI Cloud Infrastructure

security protocols to be used. Our middleware supports different kinds of protocols and allows switching between them. The possible choices all select one of the options in each category. The categories include Protocol, Authentication Scheme, Bulk Encryption Scheme, and Message Authentication Code (MAC) algorithm.

Our implementation of crypto and authentication uses WolfSSL [20], an embedded SSL implementation library. Live video stream is supported using OpenCV [21]. The application itself is built completely using C/C++, using GCC compiler.

Figures 4.2 and 4.3 show an instance of our client and server prototype implementation. The server is hosted on core cloud and gateway, and listens for client requests from gateway and IoT nodes. The client side of the system provides an interactive interface where one can choose from five different levels of security. The image representing the server side shows the server when DTLS-PSK cipher scheme is being used. In general case, we choose and recommend the static PSK scheme.

Figure 4.4(a) gives the graph generated by comparing different cipher schemes. The

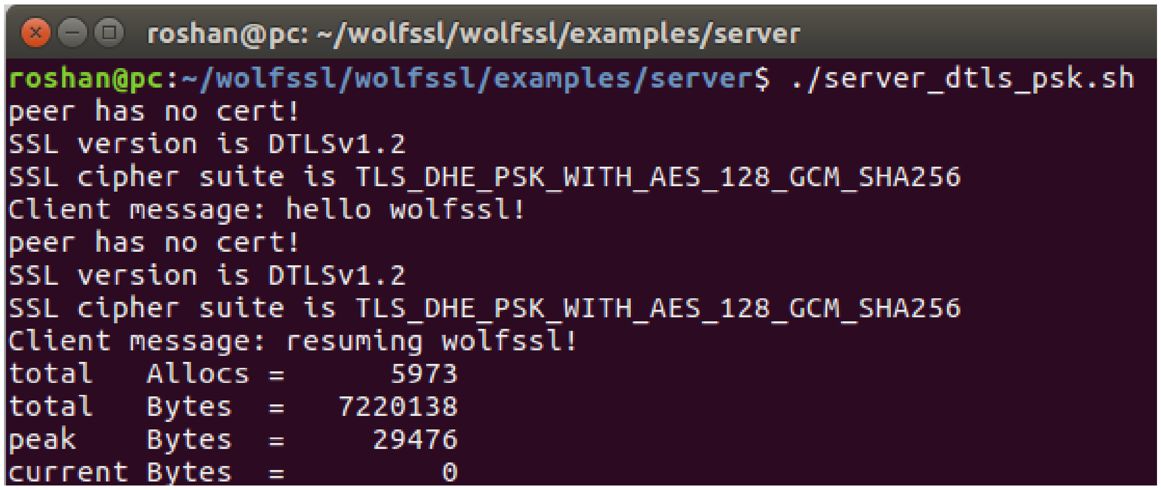


```
roshan@pc:~/wolfssl/wolfssl/examples/client$ ./decider

***********************************
Welcome to Flexible Security (v0.1)
***********************************

1) Basic live video streaming
2) TLS PSK security
3) TLS Certificate security
4) DTLS PSK security
5) DTLS Certificate security
6) DTLS Static PSK security
***********************************

4
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_DHE_PSK_WITH_AES_128_GCM_SHA256
Client Random : E527C5CF5DE49BB5F3F08A3D218F1ACF15BE5A93
D8B68C7AD44DD5C18740082D
Server response: I hear you fa shizzle!
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_DHE_PSK_WITH_AES_128_GCM_SHA256
reused session id
Server resume response: I hear you fa shizzle!
total   Allocs =       5981
total   Bytes  =    7288884
peak    Bytes  =      29481
current Bytes  =          0
```

Figure 4.2: Prototype of client UI

schemes we compared are Datagram TLS (DTLS), and TLS. The schemes were evaluated using Pre-shared Keys (PSKs) and certificates. Likewise, Figure 4.4(b) shows how much memory allocation size it takes to have the connection established. DTLS-PSK comes out to be low, by order of millions. We can see that certificate generation takes more size. Hence, choosing PSK for the resumption can be quite an excellent choice. Even better results are obtained using Static PSK, if high security is not critical to the use case.

Figure 4.5 shows results for the connection and resumption time for the four different schemes we compared in our experiments with our prototype middleware. Even though using DTLS-certificate gives consistently low time spent, we see that DTLS-PSK is the fastest scheme. DTLS can offer speed-up of over a few hundred times, regardless of cases where there is a fresh handshake or resumed session. Hence, our results show how use-



```
roshan@pc:~/wolfssl/wolfssl/examples/server$ ./server_dtls_psk.sh
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_DHE_PSK_WITH_AES_128_GCM_SHA256
Client message: hello wolfssl!
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_DHE_PSK_WITH_AES_128_GCM_SHA256
Client message: resuming wolfssl!
total   Allocs =      5973
total   Bytes  =   7220138
peak    Bytes  =     29476
current Bytes  =         0
```

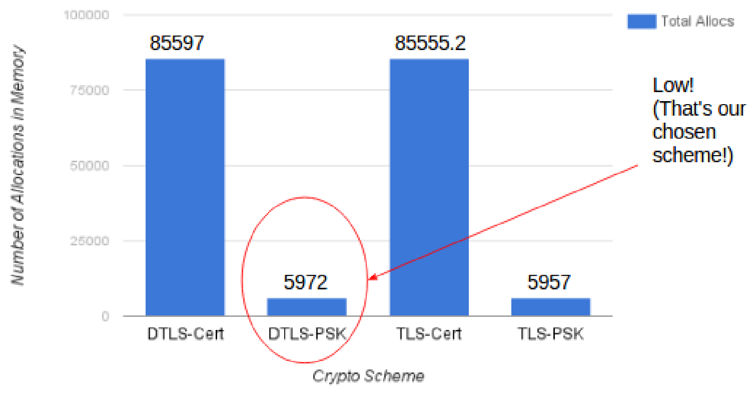
Figure 4.3: Prototype of server UI

ful Intermittent security in IoT systems can be, all the while without compromising the security, by allowing flexibility in configuration.

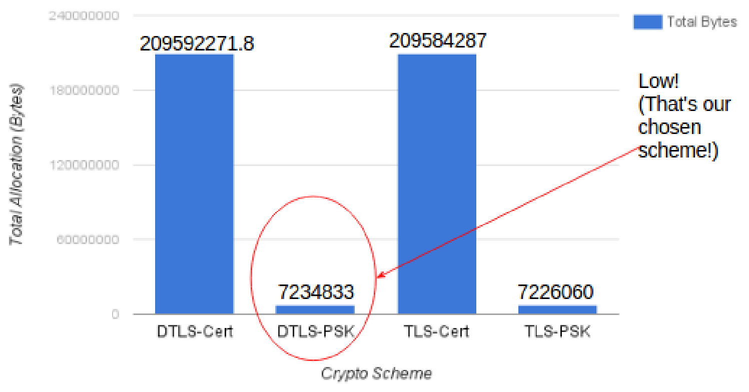
4.2 Case II: Testing Optimal Scheme Decider

In Case I test, we established the usefulness of utilizing Session Resumption, as well as having the option to choose from a few different security schemes. In Case II test, we will be taking things further and allow the middleware to form clusters from almost 200 security schemes. To test the offline phase, we check (i) Validity of the clusters formed using our method, (ii) The optimal cluster chosen, and (iii) whether limiting scope is a safe and accurate way of choosing the optimal scheme, through visualization.

In the offline phase, we attempted forming clusters to narrow down the choices to a few viable ones. Following the logic and analysis described in Chapter 3, the optimal clusters were formed. For our statistical threshold, the within-cluster sum of squares ratio of at least 60% proves that the cluster formed is valid and successful. Figures 4.6, 4.7, and 4.8 show the within-cluster sum of square ratio for the optimum cluster chosen in Group-A, B and C, respectively. As can be observed, the ratios 61.1%, 69.4%, and 62.1% are generated, and



(a) Number of memory allocations



(b) Memory allocated (in bytes)

Figure 4.4: Memory footprint for different encryption schemes

hence, the clusters are successful. This increases the certainty of optimal schemes being made available to be sifted through in the online phase. The schemes in the chosen cluster are shown in Figures 4.9, 4.10, and 4.11.

Figures 4.12, 4.13, and 4.14 show the selection of best cluster in each category, as chosen by the decider's offline phease. The figures show a plot of each of the clusters in each group, after normalization of computation index. The computation index is nothing but the benchmark metrics used for clustering and decision making. Each of the chosen clusters have their advantages and disadvantages. These can be filtered in the online phase.

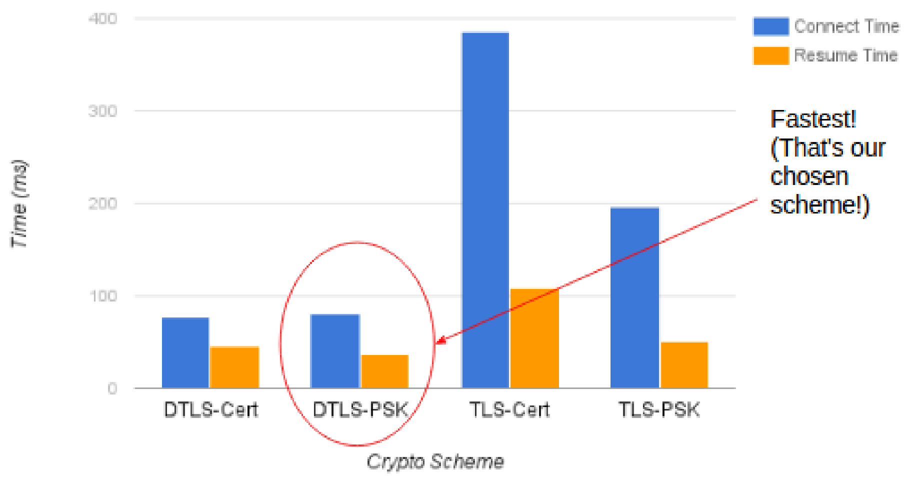
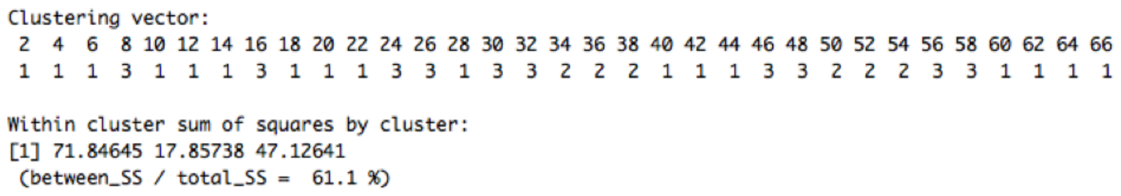
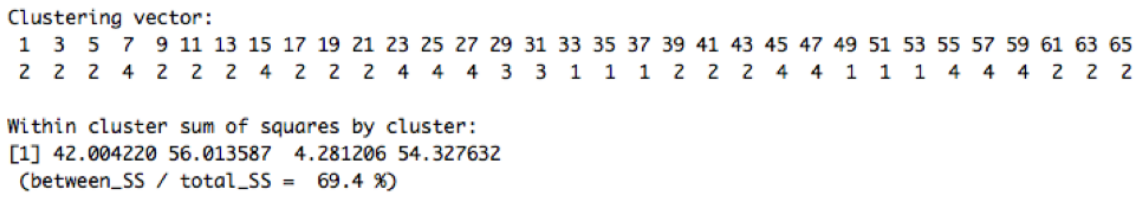


Figure 4.5: Time for connection vs. resumption for different encryption schemes



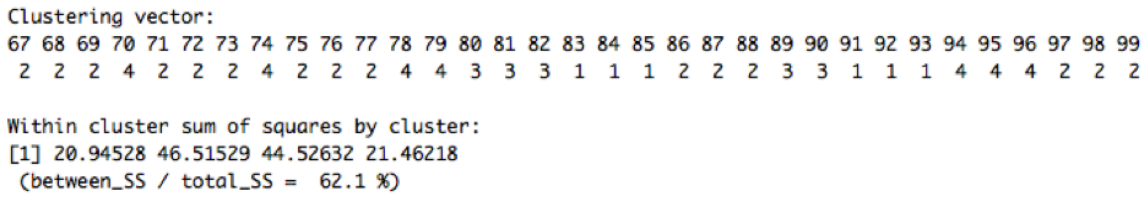
```
Clustering vector:
 2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66
 1  1  1  3  1  1  1  3  1  1  1  3  3  1  3  3  2  2  2  1  1  1  3  3  2  2  2  3  3  1  1  1  1

Within cluster sum of squares by cluster:
[1] 71.84645 17.85738 47.12641
 (between_SS / total_SS =  61.1 %)
```

Figure 4.6: Within-cluster sum of squares for Group-A



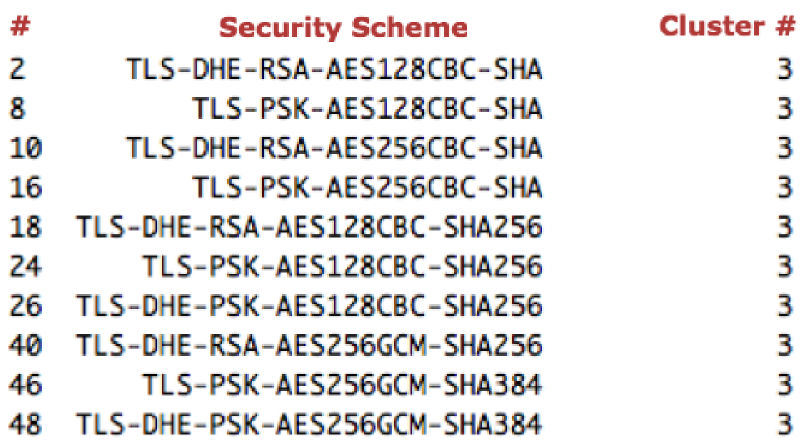
```
Clustering vector:
 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65
 2  2  2  4  2  2  2  4  2  2  2  4  4  4  3  3  1  1  1  2  2  2  4  4  1  1  1  4  4  4  2  2  2

Within cluster sum of squares by cluster:
[1] 42.004220 56.013587  4.281206 54.327632
 (between_SS / total_SS =  69.4 %)
```

Figure 4.7: Within-cluster sum of squares for Group-B



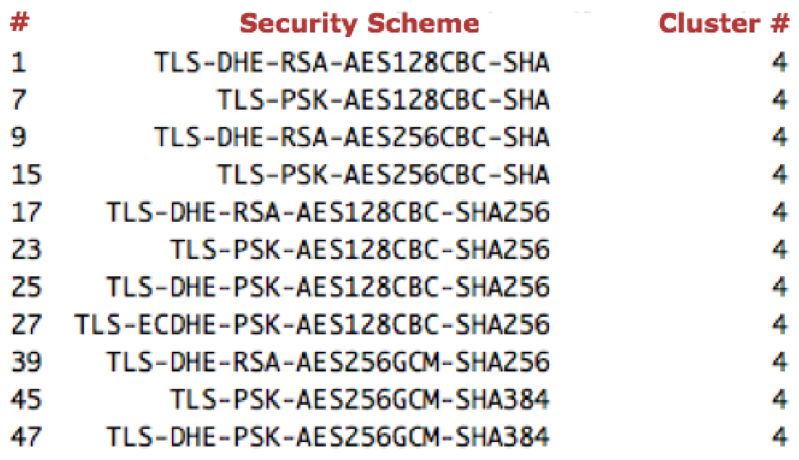
```
Clustering vector:
67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
 2  2  2  4  2  2  2  4  2  2  2  4  4  3  3  3  1  1  1  2  2  2  3  3  1  1  1  4  4  4  2  2  2

Within cluster sum of squares by cluster:
[1] 20.94528 46.51529 44.52632 21.46218
 (between_SS / total_SS =  62.1 %)
```

Figure 4.8: Within-cluster sum of squares for Group-C



#	Security Scheme	Cluster #
2	TLS-DHE-RSA-AES128CBC-SHA	3
8	TLS-PSK-AES128CBC-SHA	3
10	TLS-DHE-RSA-AES256CBC-SHA	3
16	TLS-PSK-AES256CBC-SHA	3
18	TLS-DHE-RSA-AES128CBC-SHA256	3
24	TLS-PSK-AES128CBC-SHA256	3
26	TLS-DHE-PSK-AES128CBC-SHA256	3
40	TLS-DHE-RSA-AES256GCM-SHA256	3
46	TLS-PSK-AES256GCM-SHA384	3
48	TLS-DHE-PSK-AES256GCM-SHA384	3

Figure 4.9: Group-A Cluster

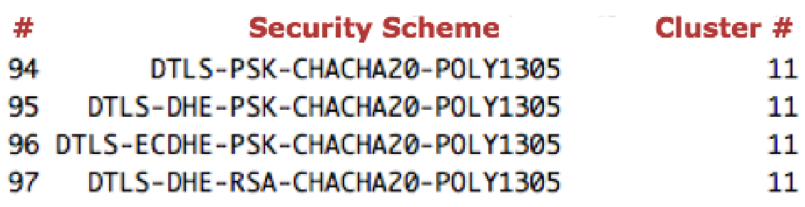


#	Security Scheme	Cluster #
1	TLS-DHE-RSA-AES128CBC-SHA	4
7	TLS-PSK-AES128CBC-SHA	4
9	TLS-DHE-RSA-AES256CBC-SHA	4
15	TLS-PSK-AES256CBC-SHA	4
17	TLS-DHE-RSA-AES128CBC-SHA256	4
23	TLS-PSK-AES128CBC-SHA256	4
25	TLS-DHE-PSK-AES128CBC-SHA256	4
27	TLS-ECDHE-PSK-AES128CBC-SHA256	4
39	TLS-DHE-RSA-AES256GCM-SHA256	4
45	TLS-PSK-AES256GCM-SHA384	4
47	TLS-DHE-PSK-AES256GCM-SHA384	4

Figure 4.10: Group-B Cluster

To verify that clustering and choosing a narrow scope is still able to provide us an optimal scheme, a visual representation through Dendrogram can be used. The dendrogram portrays the relationship between various schemes in a group.

As can be observed in Figures 4.15 and 4.16, multiple schemes in each group have siblings at the same level. This implies extreme similarities in the sibling schemes, and not considerable performance benefit. This redundancy can be easily handled by randomly picking one of the leaves and discarding the other scheme. Dendrogram for Group-C has not been shown since due to small number of available schemes in the cluster.



#	Security Scheme	Cluster #
94	DTLS-PSK-CHACHA20-POLY1305	11
95	DTLS-DHE-PSK-CHACHA20-POLY1305	11
96	DTLS-ECDHE-PSK-CHACHA20-POLY1305	11
97	DTLS-DHE-RSA-CHACHA20-POLY1305	11

Figure 4.11: Group-C Cluster

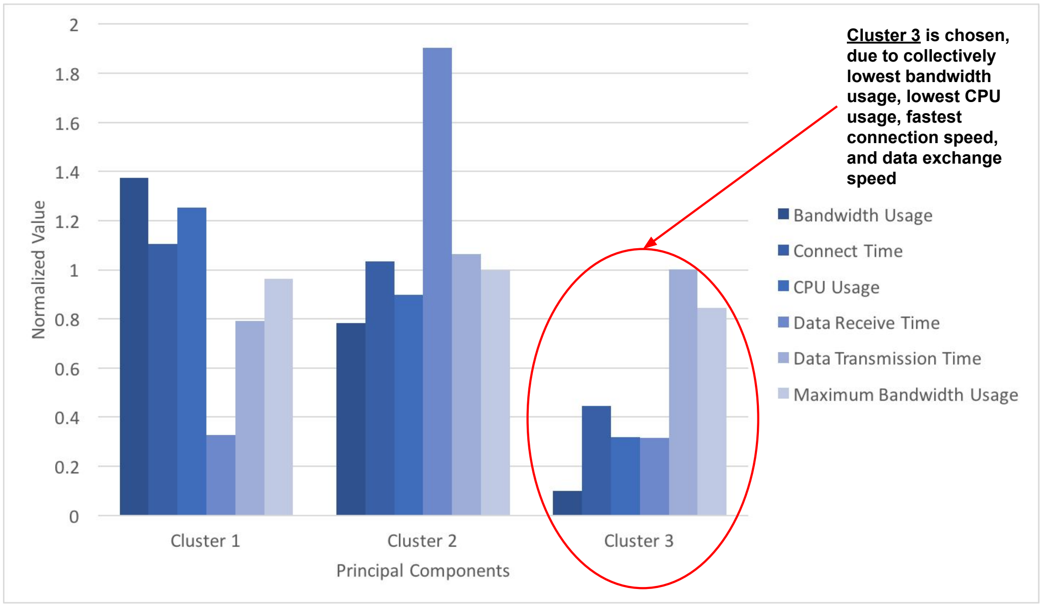


Figure 4.12: Chosen cluster in Group-A

And finally, the online phase requires collecting software requirements for the application to be utilized. The current requirement collection can be done by making an API call to

`/api/gateway/:device_id` endpoint. The application user may send the required information using the parameter list to the gateway. The response from these queries help form the metrics that are later used by the *decider*.

In our test case, we tested the decider by using random values as input requirements to check the scheme chosen by the middleware. The results are shown in Figure 4.17. As expected, for a low-security low-resource multimedia application, PSK-CHACHA20 combination was found to be the optimal choice. In addition, we analyze the trade-off charac-

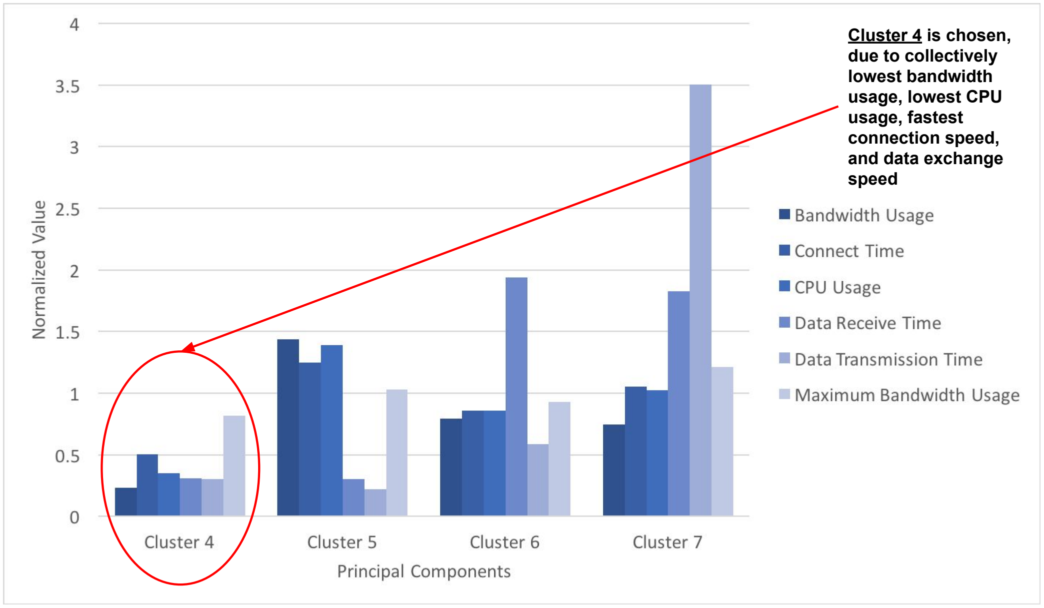


Figure 4.13: Chosen cluster in Group-B

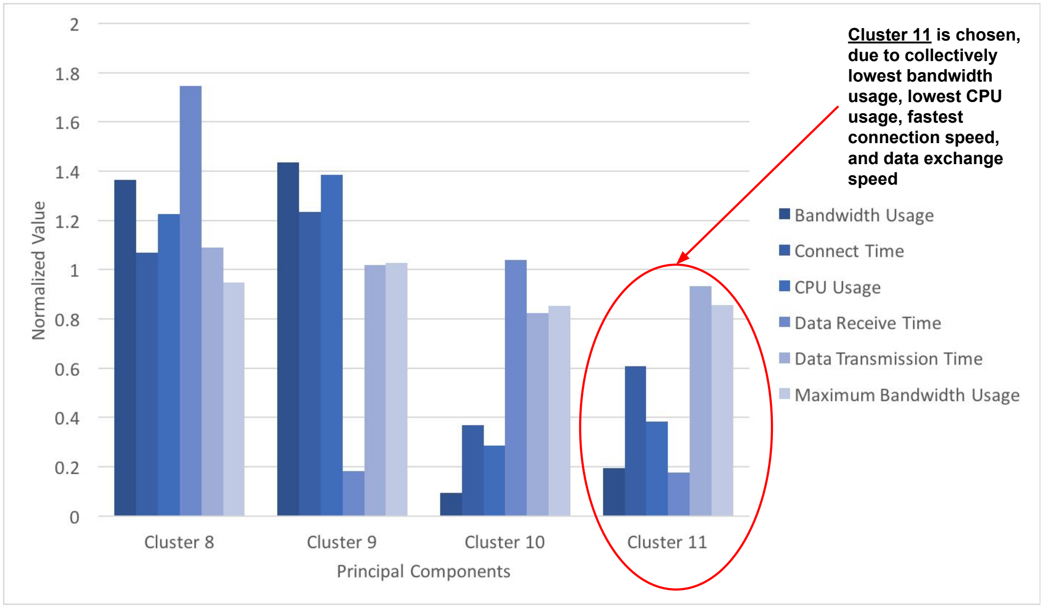


Figure 4.14: Chosen cluster in Group-C

teristics for various use-cases of IoT. A radar diagram in Figure 4.18 of the same portrays the trade-offs made. As can be seen, this scheme was found to have plenty of memory requirements and good re/connection speed, and not quite advance level of security.

Similarly, results from a second test are shown in Figure 4.19. The new requirement

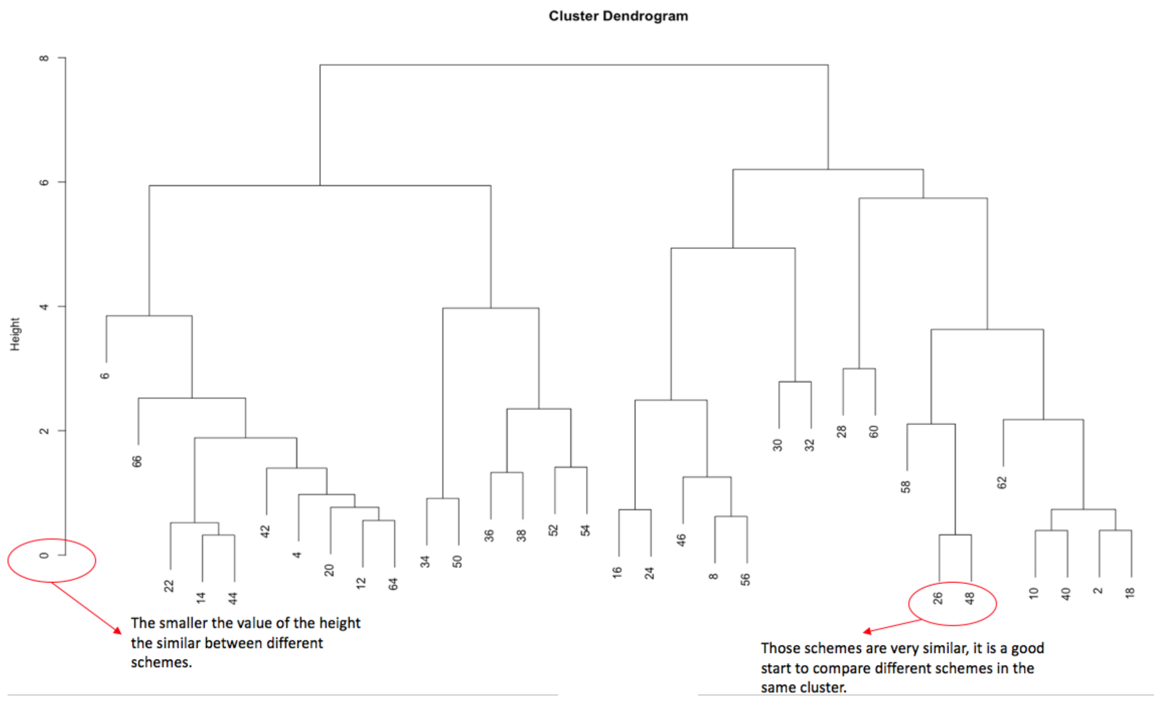


Figure 4.15: Group-A Dendrogram

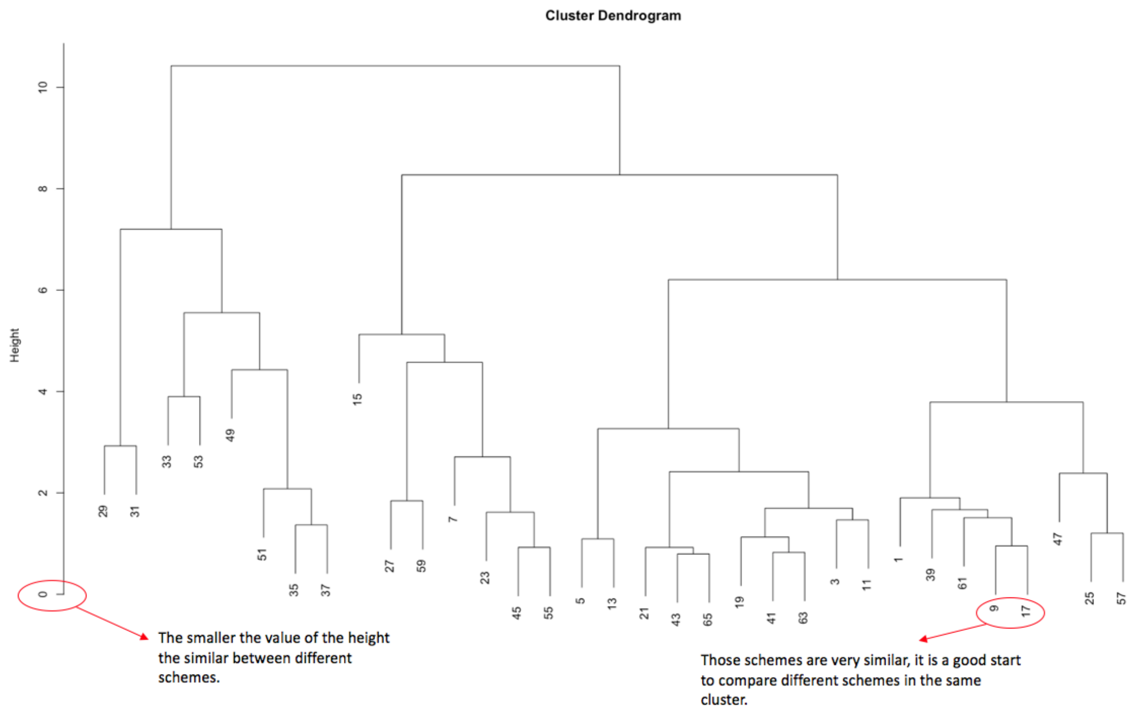
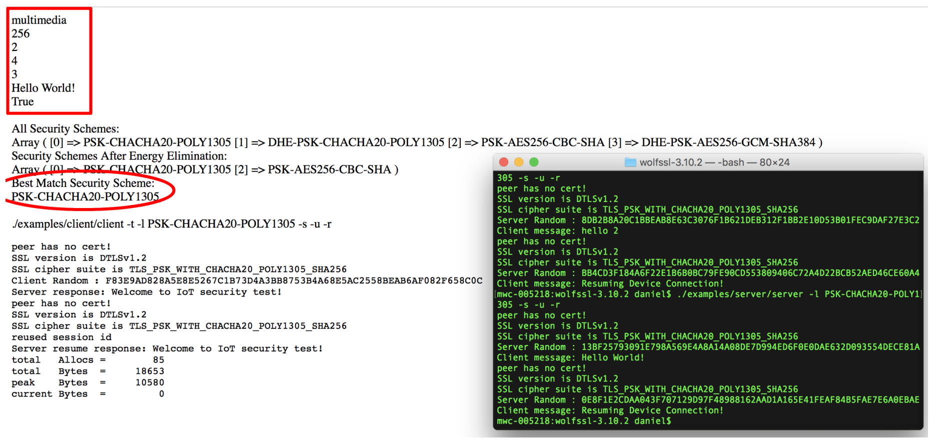


Figure 4.16: Group-B Dendrogram



```
multimedia
256
2
4
3
Hello World!
True

All Security Schemes:
Array ( [0] => PSK-CHACHA20-POLY1305 [1] => DHE-PSK-CHACHA20-POLY1305 [2] => PSK-AES256-CBC-SHA [3] => DHE-PSK-AES256-GCM-SHA384 )
Security Schemes After Energy Elimination:
Array ( [0] => PSK-CHACHA20-POLY1305 [2] => PSK-AES256-CBC-SHA )
Best Match Security Scheme:
PSK-CHACHA20-POLY1305

./examples/client/client -t -l PSK-CHACHA20-POLY1305 -s -u -r

peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_PSK_WITH_CHACHA20_POLY1305_SHA256
Client Random : F83E9AD828A5E8E5267C1B73D4A3BB8753B4A68E5AC2558BEAB6AF082F658C0C
Server response: Welcome to IoT security test!
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_PSK_WITH_CHACHA20_POLY1305_SHA256
reused session id
Server resume response: Welcome to IoT security test!
total   Allocs =        85
total   Bytes  =     18653
peak    Bytes  =     10580
current Bytes  =         0
```

```
wolfssl-3.10.2 — -bash — 80×24
305 -s -u -r
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_PSK_WITH_CHACHA20_POLY1305_SHA256
Server Random : 8DB2B8A20C1BBEAB8E63C3076F1B621DEB312F1BB2E10D53B01FEC9DAF27E3C2
Client message: hello 2
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_PSK_WITH_CHACHA20_POLY1305_SHA256
Server Random : BB4CD3F184A6F22E1B6B0BC79FE90CD553809406C72A4D22BCB52AED46CE60A4
Client message: Resuming Device Connection!
mwc-005218:wolfssl-3.10.2 daniel$ ./examples/server/server -l PSK-CHACHA20-POLY1
305 -s -u -r
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_PSK_WITH_CHACHA20_POLY1305_SHA256
Server Random : 13BF25793091E798A569E4A8A14A08DE7D994ED6F0E0DAE632D093554DECE81A
Client message: Hello World!
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_PSK_WITH_CHACHA20_POLY1305_SHA256
Server Random : 0E8F1E2CDAA043F707129D97F48988162AAD1A165E41FEAF84B5FAE7E6A0EBAE
Client message: Resuming Device Connection!
mwc-005218:wolfssl-3.10.2 daniel$
```

Figure 4.17: Chosen scheme for a low resource device for frequent disconnections

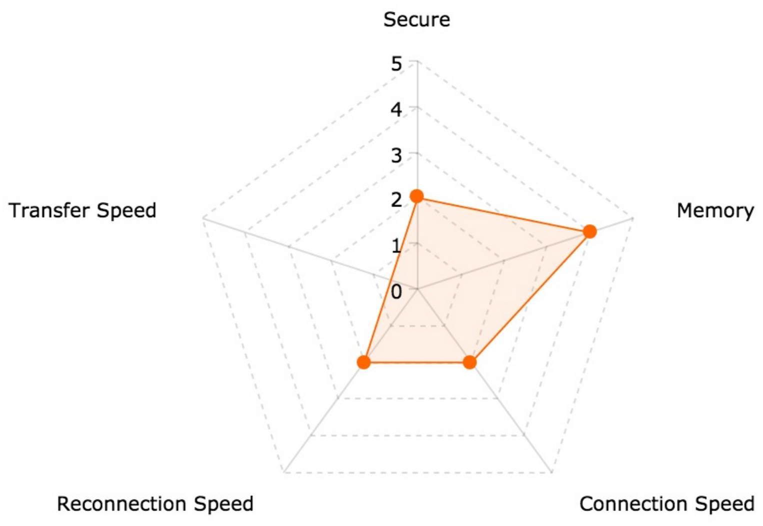
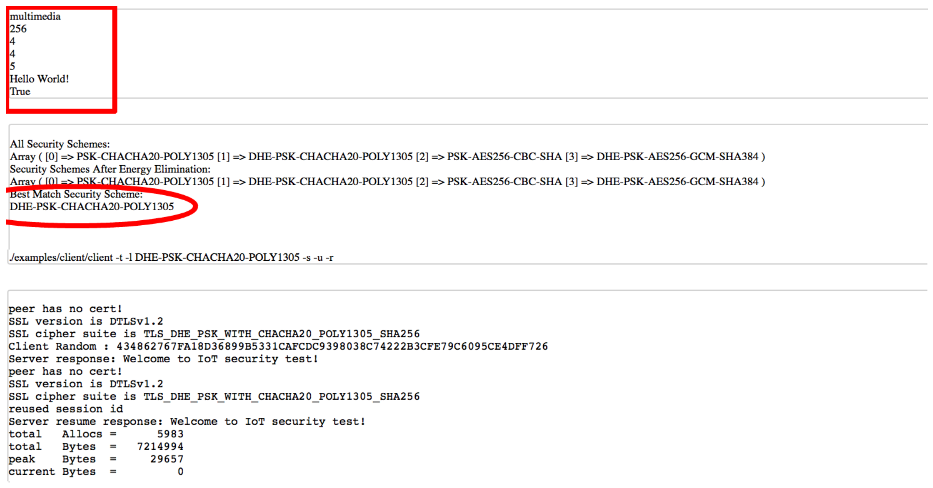


Figure 4.18: Radar representation of scheme *DTLS-PSK-CHACHA20-POLY1305*

priority being high security on a low-resource multimedia application, Epehemeral (DHE) PSK-CHACHA20 combination was found to be the optimal choice. A radar diagram in Figure 4.20 shows the security mapping of the same. In this case, security was found to be the most important criteria, even if at the expense of re/connection speed. Hence, public key cryptography scheme was chosen for key exchange, along with a highly reliable integrity check MAC algorithm (POLY1305).

In many other test cases, we have observed the radar diagram to visualize the trade-offs. This has also been used to analyze when session resumption is found useful, and when



```
multimedia
256
4
4
5
Hello World!
True
```

```
All Security Schemes:
Array ( [0] => PSK-CHACHA20-POLY1305 [1] => DHE-PSK-CHACHA20-POLY1305 [2] => PSK-AES256-CBC-SHA [3] => DHE-PSK-AES256-GCM-SHA384 )
Security Schemes After Energy Elimination:
Array ( [0] => PSK-CHACHA20-POLY1305 [1] => DHE-PSK-CHACHA20-POLY1305 [2] => PSK-AES256-CBC-SHA [3] => DHE-PSK-AES256-GCM-SHA384 )
Best Match Security Scheme:
DHE-PSK-CHACHA20-POLY1305


./examples/client/client -t -l DHE-PSK-CHACHA20-POLY1305 -s -u -r
```

```
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256
Client Random : 434862767FA18D36899B5331CAFCDC9398038C74222B3CFE79C6095CE4DFF726
Server response: Welcome to IoT security test!
peer has no cert!
SSL version is DTLSv1.2
SSL cipher suite is TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256
reused session id
Server resume response: Welcome to IoT security test!
total   Allocs =      5983
total   Bytes  =   7214994
peak    Bytes  =     29657
current Bytes  =         0
```

Figure 4.19: Chosen scheme for a low resource device for high security

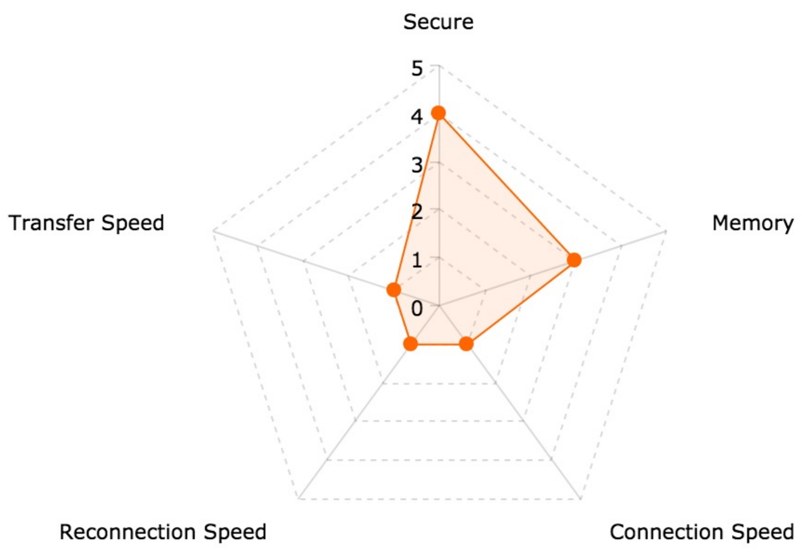


Figure 4.20: Radar representation of scheme *DTLS-DHE-PSK-CHACHA20-POLY1305*

it is not. For instance, Figure 4.21 shows the specifications for a chosen scheme for an application requiring high reconnection speed through session resumption, having lower memory and security requirements. This application appears to be a good candidate for ‘lost-edge‘ IoT use-cases. Contrasting this application to a different application’s chosen scheme trade-off specifications, as shown in Figure 4.22, we can observe that a very high level of security can be obtained at expense of faster reconnection. Such application is a good candidate for ‘smart-edge‘ IoT use-cases.

Similarly, for another lost-edge IoT application, chosen scheme's specifications are visualized in Figure 4.23. Very high reconnection speed appears to be a requirement, hence speed over security. And finally, for a similar application without session resumption support, the chosen scheme specifications can be visualized in Figure 4.24. This application must be another one of 'smart-edge' IoT application, as an all-rounded security scheme is needed, with a higher priority for security and initial connection speed.

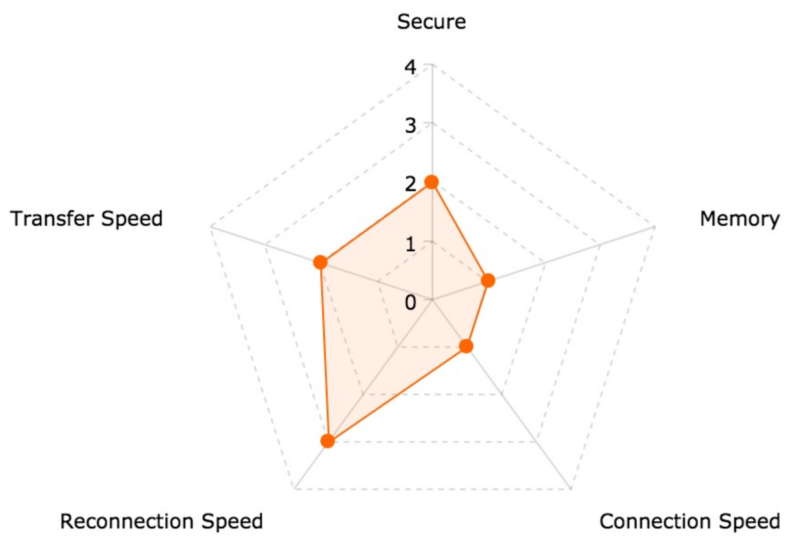


Figure 4.21: Radar representation of scheme *TLS-DHE-PSK-AES256-GCM-SHA384 w/ Session Resumption*

Hence, we can see that using the *optimal scheme decider* we are able to determine the optimal scheme for a given device in an IoT-based application, and support many different kinds of IoT applicationsl

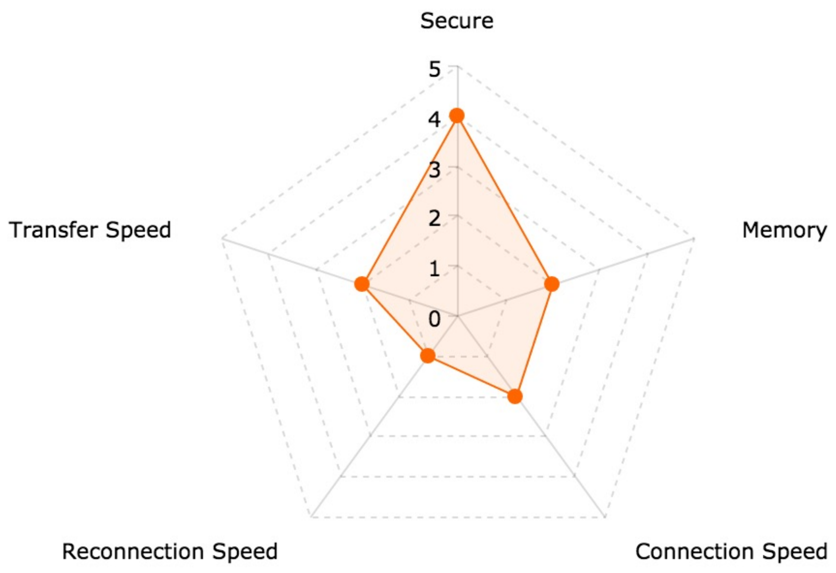


Figure 4.22: Radar representation of scheme *TLS-DHE-PSK-AES256-GCM-SHA384 w/o Session Resumption*

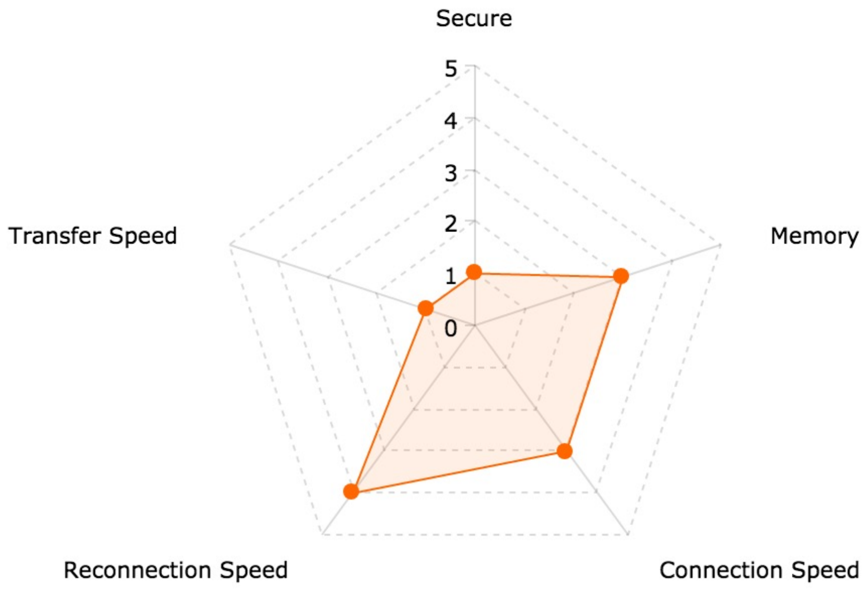


Figure 4.23: Radar representation of scheme *TLS-PSK-AES256-CBC-SHA w/ Session Resumption*

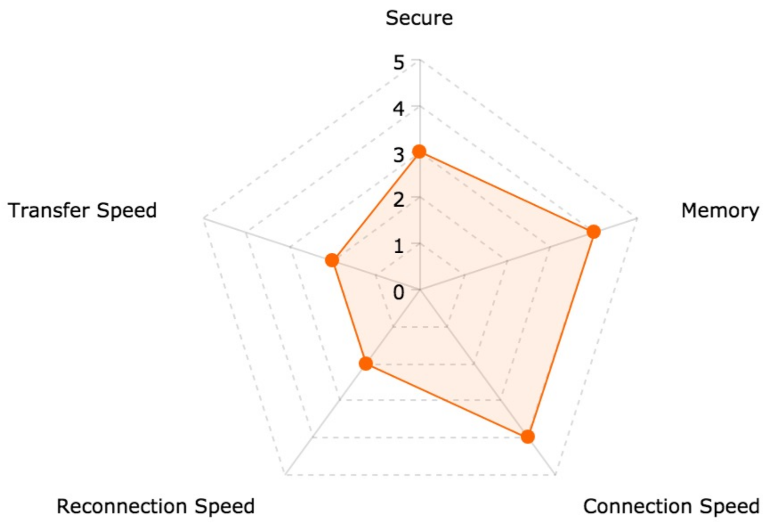


Figure 4.24: Radar representation of scheme *TLS-PSK-AES256-CBC-SHA w/o Session Resumption*

Chapter 5

Future Work

The future work scope for this middleware is massive, and extension of this research can span across multiple facets. Some of the natural extensions for this research can be in the form of automating the process of collecting device parameter and software requirements, and adding many other protocols, ciphers, and MAC to the offline process.

Another nifty feature that could be implemented in close future is dynamic switching of security scheme (protocols in use), based on various factors, such as resources available, data context, importance or confidential requirements of data, and so on.

In addition, there is an opportunity to explore ‘*deferred security*’. Deferred security could allow: (a) an ex-member node transmit encrypted data to be decrypted at a later time, using a buffer, (b) an ex-member node to back-off until a later point, until it has been authenticated which in turn could allow the system to be more reliable, even in relatively unreliable network.

Other future work could focus on trust management through device reputation. Using the history of devices’ past security associations, a trust management framework could be integrated with our middleware. Reputation for a device can be measured using the record of how many times the devices has been disconnected and/or compromised by adversaries.

With high reputation, devices can reconnect to the network with minimal security checks whereas, if a device has a spotty reputation, the device will have to undergo comprehensive security handshake process.

Another branch of work could be development of transient reputation scheme, that can collect and use short-term knowledge about the connecting devices to build a short-lived reputation. This would deprecate the need to maintain trust state in the network amongst the IoT devices.

Among many other possible extensions, an excellent scope could be explored by having the middleware adopt the concept of '*Privilege De-escalation*' followed by re-establishment of trust. What this involves is the gateway delegating trivial tasks to a new node to establish trust. If the node is able to build up trust by accomplishing multiple trivial tasks, its privilege can get escalated and the gateway could decide to trust the node with more sensitive and important data transfer. All of this is achieved without relying on long-term trust framework.

Chapter 6

Summary and concluding remarks

In this thesis, we developed an end-to-end IoT security middleware between devices at the network edge and the core cloud side of an application system. Our middleware is based on a novel security scheme, which provides flexibility for securing IoT-based application data, along with offering quick re-connections to aid in situations of unreliable network conditions within cloud-fog communication platforms.

Our results demonstrate the need for flexibility in choice of an IoT security scheme based on resource constraints in computation, bandwidth, memory, network reliability, as well as the application for which the IoT system is being designed. We show that whenever feasible and acceptable, the use of static properties such as Static PSK can notably speed-up secure communications. Static PSKs in prior literature have not received much attention, however they could be a useful tool for low-resource, moderate-security within IoT systems.

Additionally, we have successfully shown an all-encompassing solution to IoT-based application security requirements. A standardized version of this could result in seamless end-to-end, robust, reliable, and secure connectivity between multiple networks, IoT or otherwise. The middleware is able to provide security compatibility with existing core cloud network.

This middleware can have a number of use-cases. One of the biggest impact can be considered in disaster response systems which utilize Cloud and Edge Fog architecture involving edge IoT nodes and beacons, for instance in [2]. In general, any system that requires the node data security to be flexible and dynamic, based on security association, or data context, should be able to utilize this middleware. Key impact would be the speed of secure communication, since the protocol does not enforce full security all the time, but instead allows flexibility to decide. Time, and other critical resources can be saved in this way. The benefit of being able to handle trade-off can prove extremely useful for a many different kinds of IoT-based applications. Hence, many complementing use-cases can be managed using our middleware without getting overwhelmed by protocols.

Bibliography

[1] IoT trends 2016- tech insider. `http://www.businessinsider.com/top-internet-of-things-trends-2016-1`.

[2] J. Gillis, P. Calyam, O. Apperson, S. Ahmad. “Panacea’s Cloud: Augmented reality for mass casualty disaster incident triage and co-ordination”. *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2016.

[3] J. Gillis, P. Calyam, A. Bartels, M. Popescu, S. Barnes, J. Doty, D. Higbee, S. Ahmad. “Panacea’s Glass: Mobile Cloud Framework for Communication in Mass Casualty Disaster Triage”. *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, 2015.

[4] M. Rantz, M. Skubic, C. Abbott, C. Galambos, M. Popescu, J. Keller, E. Stone, J. Back, S. J. Miller, and G. F. Petroski. “Automated in-home fall risk assessment and detection sensor system for elders”. *The Gerontologist*, 55(Suppl 1):S78–S87, 2015.

[5] J. Burchard, D. Chemodanov, J. Gillis, P. Calyam. “Wireless Mesh networking Protocol for sustained throughput in Edge Computing”. *2017 International Conference on Computing, Networking and Communications (ICNC)*, 2017.

[6] R. Bonetto, N. Bui, V. Lakkundi, A. Olivereau, A. Serbanati, M. Rossi. “Secure Communication for Smart IoT Objects: Protocol Stacks, Use Cases and Practical

Examples". *IEEE World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2012.

[7] X. S. Huber Flores, V. Kostakos, A. Y. Ding, P. Nurmi, S. Tarkoma, P. Hui, and Y. Li. "Large-scale Offloading in the Internet of Things". In *International Conference on Pervasive Computing and Communications Workshops*, PerCom WS, 2017.

[8] H. Khemissa and D. Tandjaoui. "A novel lightweight authentication scheme for heterogeneous wireless sensor networks in the context of Internet of Things". In *2016 Wireless Telecommunications Symposium, WTS 2016, London, United Kingdom, April 18-20, 2016*, pages 1–6, 2016.

[9] H. Khemissa, D. Tandjaoui. "A Lightweight Authentication Scheme for E-health applications in the context of Internet of Things". *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, 2015.

[10] C. Huth, J. Zibuschka, P. Duplys, T. Guneysu. "Securing Systems on the Internet of Things via Physical Properties of Devices and Communications". *Systems Conference (SysCon), 2015 9th Annual IEEE International*, 2015.

[11] Md. A. Iqbal, M. Bayoumi. "Secure End-to-End Key Establishment Protocol for Resource-Constrained Healthcare Sensors in the Context of IoT". *International Conference on High Performance Computing and Simulation (HPCS)*, 2016.

[12] R. Hummen, H. Wirtz, J. H. Ziegeldorf, J. Hiller, K. Wehrle. "Tailoring End-to-End IP Security Protocols to the Internet of Things". *21st IEEE International Conference on Network Protocols (ICNP)*, 2013.

[13] G. Piro, G. Boggia, L. A. Grieco. "A standard compliant security framework for IEEE 802.15.4 networks". *IEEE World Forum on Internet of Things (WF-IoT)*, 2014.

[14] P. Syverson. “A Taxonomy of Replay Attacks [cryptographic protocols]”. *Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings*, 1994.

[15] F. C. Kuo, H. Tschofenig, F. Meyer, X. Fu. “Comparison Studies between Pre-Shared and Public Key Exchange Mechanisms for Transport Layer Security”. *25th IEEE International Conference on Computer Communications. Proceedings (INFOCOM)*, 2006.

[16] J. A. Dev. “Usage of Botnets for High Speed MD5 Hash Cracking”. *3rd International Conference on Innovative Computing Technology, INTECH*, 2013.

[17] D. Lee. “Hash Function Vulnerability Index and Hash Chain Attacks”. *3rd IEEE Workshop on Secure Network Protocols, NPSec*, 2007.

[18] The first collision for full SHA-1. `http://shattered.io/static/shattered.pdf/`.

[19] NSF-supported GENI Cloud Infrastructure. `https://www.geni.net/`.

[20] WolfSSL. `https://www.wolfssl.com/wolfSSL/Home.html`.

[21] OpenCV. `http://opencv.org/`.