Flexible IoT Security Middleware for End-to-End Cloud-Fog Communication

Bidyut Mukherjee[a], Songjie Wang[b], Wenyi Lu[a], Roshan Lal Neupane[a], Daniel Dunn[a], Yijie Ren[a], Qi Su[a], Prasad Calyam[b]

[a]*{bm346, wldh6, rlnzq8, dpdbp7, yry7d, qsn4c}@mail.missouri.edu*
[b]*{wangso, calyamp}@missouri.edu*

Abstract

IoT (Internet of Things) based smart devices such as sensors have been actively used in edge clouds i.e., 'fogs' along with public clouds. They provide critical data during scenarios ranging from e.g., disaster response to in-home healthcare. However, for these devices to work effectively, end-to-end security schemes for the device communication protocols have to be *flexible* and should depend upon the application requirements as well as the resource constraints at the network-edge. In this paper, we present the design and implementation of a *flexible* IoT security middleware for end-to-end cloud-fog communications involving smart devices and cloud-hosted applications. The novel features of our middleware are in its ability to cope with intermittent network connectivity as well as device constraints in terms of computational power, memory, energy, and network bandwidth. To provide security during intermittent network conditions, we use a 'Session Resumption' algorithm in order for our middleware to reuse encrypted sessions from the recent past, if a recently disconnected device wants to resume a prior connection that was interrupted. In addition, we describe an 'Optimal Scheme Decider' algorithm that enables our middleware to select the best possible end-to-end security scheme option that matches with a given set of device constraints. Experiment results show how our middleware implementation also provides fast and resource-aware security by leveraging static properties i.e., static pre-shared keys (PSKs) for a variety of IoT-based application requirements that have trade-offs in higher security or faster data transfer rates.

Keywords: IoT Security Middleware, Mobile Edge Cloud, Cloud-Fog Communication, Secure IoT Applications

1. Introduction

Internet of Things (IoT) systems typically comprise of a network of connected devices or "things". 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 to an edge or core cloud computing platform can be a part of an IoT system. As mentioned in [1], emerging IoT system trends are set to completely change the way businesses, governments, and consumers interact with each other, and transact in a data-driven economy.

As a specific example, an IoT system of Geo Sensors could involve collection of various kinds of geographical information related to soil, forest terrains, and weather. The collected data is continually sent to nearby edge clouds or 'fog computing platforms' for aggregation and drill-down analysis/visualization. Similarly, smart devices in IoT systems can also provide critical data during e.g., disaster response scenarios or in-home healthcare. An exemplar application of an IoT system used for disaster response is Panacea's Cloud [2, 3, 4]. Such systems aid in providing medical triaging and quick response during emergency disaster situations. The Panacea's Cloud system involves many distributed IoT devices transmitting data from various locations in a disaster region to a responding personnel's handheld device, which in turn utilizes a mobile cloud-fog setup that provides intelligent dashboard visualizations. Considering the in-home healthcare application scenario, IoT systems such as [5, 6] are aimed at providing emergency services to the elderly, if the need arises. These involve smart sensors and cameras using cloud-fog storage for keeping track of movements made by the elderly. The IoT system here could use cloud-fog computation for algorithms that help notify primary care contacts if an elder's gait signature appears to contain certain types of anomaly events that suggest high fall risk.

We can see that the above IoT-based applications can have broad use cases involving diverse infrastructure configurations and data/resource security requirements. Consequently, ad-hoc communication protocol implementations with undesirable security overheads are not suitable. Consider the aforementioned case of disaster response systems [3]; the edge network here comprises of IoT devices that typically operate in resource-constrained (compute, memory, storage, network, energy) environments. Such systems could become highly unstable and unreliable due to physical infrastructure damage or lossy edge links when there are extreme events such as hurricanes, tornadoes or earthquakes. The IoT system security in this case needs to be configured such that the security overhead of the communication protocols does not impede the already slow data transfer speeds due to constrained edge resources and infrastructure.

Alternately, the above IoT systems for providing in-home healthcare, as in [5, 6] may have access to fully functional edge infrastructure including Gigabit fiber connections (e.g., Google Fiber) to user sites and a set of fog resources at a nearby hospital data center. In such cases, the available resources can be readily used to handle the big data generated from patient homes (typically up to 23GB/Person/Week). The IoT system security for this application case involving elderly patient data needs to be configured for maximum data confidentiality and integrity, even if data transfer speeds are affected due to security overhead of the communication protocols.

In this paper, we address the wide-ranging IoT-based application security needs and diverse network-edge resource constraints by proposing a novel design of a *flexible* IoT security middleware for end-to-end cloud-fog communication. Our goal is to primarily secure the network located at the user fogs, i.e., where the IoT devices are located. However, we also seek to maintain security compatibility with an existing core cloud network using *System Level* or *Application Level* deployment considerations of our middleware.

The salient features of our IoT security middleware, which form the main paper contributions of our work are in its ability to provide: (i) 'Intermittent Security', and (ii) 'Resource-aware Security' for smart devices and cloud-hosted applications. Pertaining to Intermittent Security, our middleware uses a *Session Resumption* algorithm 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 connection. With regards to Flexible Security, our middleware uses a novel *Optimal Scheme Decider* algorithm that allows users to configure the best possible end-to-end security scheme option that matches with a given set of device resource constraints. Through application resource-awareness obtained via supervised machine learning (versus blindly following a rigid/adhoc security configuration), our middleware enables users to configure either higher security or prioritize faster data transfer rates, via RESTful APIs for data collection at a cloud or a fog site.

We validate our middleware implementation's ability to provide robust, fast and resource-aware security through experiment results in an actual cloud-fog testbed, as well as through simulation results. Results also show that our work lays a foundation for promoting increased adoption of static properties such as Static PSKs that can handle the trade-offs in high security or faster data transfer rate requirements within IoT-based applications.

The remainder of this paper is organized as follows: In Section 2, we discuss related work. In Section 3, we provide an overview of our middleware and provide a detailed description of our approach with a corresponding reference architecture. Section 4 elaborates on our Intermittent Security solution and details our Session Resumption algorithm. Next, Section 5 discusses our Resource-aware Security solution and details our Optimal Scheme Decider algorithm. Section 6 evaluates the effectiveness of our middleware and compares the performance of an IoT-based application with-and-without our middleware. Section 7 concludes this paper and suggests future work.

2. Related Work

The core concepts of IoT related sensing and communication have been outlined in prior works such as [7] and [8]. Authors in [7] identify that IoT systems need to implement a shared understanding of the situation of users and their devices with context-awareness. Thus, a requirement for IoT-based sensing systems design including security needs to address adaptation requirements to cope with dynamic contexts and varied application platforms. Similarly, authors in [8] identify the high-level abstractions and interoperability needs to ensure proper security in the form of confidentiality, integrity and availability, as well as to bridge the gap between IoT (i.e., constrained edge) and enterprise (i.e., unconstrained core cloud) communications.

From the perspective of frameworks that address IoT related sensing and communication issues, prior work such as [9] and [10] have focused on smart city applications. In [9], data service models to deal with real-time data analysis are presented along with a tiered security service to handle data transmissions. The proposed framework supposes that communication between IoT sensors and the application back-end (i.e., in an agricultural data analysis use case) needs to be ad-hoc and fast, while security design needs to be flexible to handle time-sensitivity or content-sensitivity. In contrast, a case for scalability and plug-in components in IoT-based applications is presented in [10], and a flexible framework design similar to our security middleware is presented. The authors list challenges in handling differences in communication protocols between the IoT devices and the edge gateway.

IoT-based application deployment is a relatively new trend, however methods to secure networked IoT devices have been explored in the past. Work in [11] discusses security procedures for constrained IoT devices. 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 [12]. In [13] and [14], the authors propose lightweight authentication schemes that can be used in the context of IoT systems and constrained wireless sensor networks. In comparison, our work investigates a security middleware which makes use of static Pre-Shared Keys (PSKs) that is different from the multi-phase encryption and decryption used in earlier works. We phase down to just one iteration for lightweight authentication. Consequently, our approach reduces security overhead in the cloud-fog communication protocols and is less time consuming in an IoT-based application deployment.

Work in [15] addresses issues in session key management for IoT-based applications, particularly concerning health-care sensors. The authors devised a secure end-to-end protocol for resource-constrained devices by adapting security functionality used in unconstrained devices, but without computationally intensive operations. They offloaded heavy computation at constrained devices to neighboring trusted nodes/devices. Their session key creation, however, was ephemeral. Similarly, authors in [16] provide methods to offload security computation from devices to the edge cloud. Our work builds on top of these works, and uses an easier, yet effective key management scheme. Specifically, we create static keys which are not short-lived, reducing the key exchange cost and time significantly.

Our work is closely related to prior work in [17, 18], where the same authors developed security schemes for mobility-enabled healthcare IoT systems. They outline architectures that are based on certificate-based DTLS handshake. In addition, their scheme utilizes the session resumption concept for communication, and proposes system mobility through interconnected smart gateways. Our work extends this highly relevant work to incorporate flexibility and resource-awareness in the security scheme configuration for IoT-based applications that function in both austere and smart network environments.

A comprehensive session resumption mechanism is discussed in [19]. The work uses HIP DEX i.e., Host Identity Protocol Diet EXchange which provides secured end-to-end connections in IoT systems. Perfect forward secrecy and non-repudiation properties of HIP result in significantly decreased protocol handshake overhead and reduced handshake run-time. 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. 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.

We encountered use of different encryption techniques in earlier works. As an exemplar, work in [20] makes use of Physical Unclonable Functions (PUF) over public key cryptography, which takes advantage of existing physical properties of a device. They utilize Physical Key Generation (PKG) that use physical properties of the communication channel, and is lighter than common public-key cryptography. Along the same lines, work in [21] aims for lightweight user authentication and key agreement protocol implementation, but relies heavily on use of smart cards. Our work extends upon such works to use physical properties of the devices, and is able to forgo re-authentication by using session resumption.

Authors in [22] give a standard security compliant framework to secure the IEEE 802.15.4 networks in low power lossy network (LLNs). The framework supports five 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. 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 security framework, specifically the concept of Resource-aware Security, a step further towards practical use within IoT-based applications.

3. IoT Security Middleware Overview

In this section, we first describe our vision of the physical infrastructure necessary to deploy our middleware that builds upon our previous work [23]. Following this, we list the various modules in our middleware and explain the interactions of our middleware with a cloud gateway and a smart device. Lastly, we discuss practical deployment considerations for the deployment of our middleware in an IoT system.

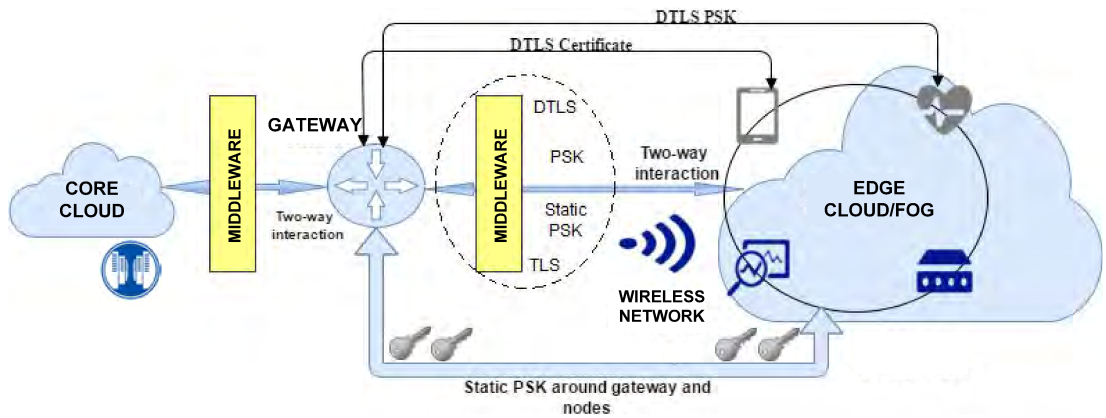


Figure 1: IoT system architecture - Core Cloud along with the Gateway forms an unconstrained network, whereas Gateway along with the IoT devices forms a constrained network segment [23]

.

3.1. Physical Infrastructure

Figure 1 illustrates a physical infrastructure view to integrate our middleware in an IoT system that involves cloud-fog communication. The infrastructure primarily consists of a core cloud infrastructure, a gateway, and several edge IoT devices. The core cloud has a communication channel with gateways at the network-edge. The edge gateways form an interface to the network for the IoT devices. Parts of the primary middleware need to be installed on both the gateway and the fog IoT devices. The fog network can constitute any number or type of IoT devices, such as heart monitor, beacon, geo sensor, etc. The primary middleware allows secure and fast data transfer between the sensor devices and the gateway by using the "Intermittent Security" feature to ensure robustness against frequent disconnections, and by using security schemes chosen through our "Resource-aware Security" module. To realize these benefits, our middleware stores and tracks sessions, certificates, or keys, 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 device protocols.

Optionally, secondary middleware can be integrated between the gateway and the core cloud to provide additional flexibility and robustness, if needed. There is a key benefit of having this integration 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 ideal integration consists of a middleware piece in the core cloud network side, and another middleware piece at the fog network side. Each middleware piece consists of a server-client pair interacting with just each other. At the gateway, the client of the cloud interacts with the server of edge, allowing end-to-end secured communication. Our middleware additionally supports device resource-aware security by accommodating different protocols for individual nodes in the fog network, in addition to being accessible to use static PSKs for quick encryption setup, and support for intermittent security through session resumption.

We suppose in Figure 1 that smart applications that are based on IoT data from e.g., sensors collect contextual data at large scales from several geographically distributed fog/edge locations. Our middleware helps in secure integration and analysis of such data using cloud platforms and wireless communication networks for actionable insights. Our middleware can be useful in cases where custom application dashboards in e.g., public safety, transportation or rural healthcare require secure data import and export in a cloud and fog communications infrastructure to deal with IoT devices with varying trust, resource constraints and wireless network reliability. The middleware can be customized in rural areas or in the middle of areas with sparse/intermittent wireless connectivity, where strict security requirements might lead to channel bandwidth consumption overheads or fast drainage of constrained device resources. This in turn might block data access from IoT devices, or cause data integrity issues that lead to discarding of important data in some cases. Thus, our middleware addresses the lack of flexible trust management in today's smart applications in a manner that can enable data collection with minimal viable security. Consequently, it helps with real-time IoT device data ingest into a cloud from both trusted and un/low-trusted devices to facilitate follow-up actions by decision makers at the fog sites.

3.2. Logical Modules

A modular diagram of our proposed middleware is shown in Figure 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 (see Section 4 for details on the background, algorithm and implementation) try and resume a past connection, by first verifying session existence and validity. If the resumption fails, Resource-aware Security

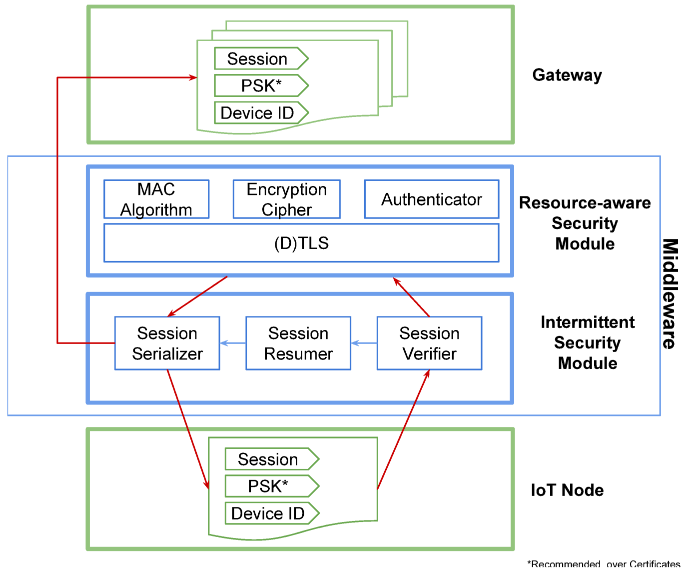


Figure 2: End-to-End IoT Security Middleware Module Diagram [23]

module (see Section 5 for details on the background, algorithm and implementation) acts as an interface to allow configuration of required security schemes. These two modules in conjunction form the middleware.

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.

3.3. Deployment

In practical IoT system deployments, the middleware can be installed at both the *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.

4. Intermittent Security

4.1. Solution Approach

Intermittent security utilizes session resumption to quickly bring a disconnected device back in the network when needed again. This concept closely follows the ideas proposed in [19], and but we consider a few modified factors. Our middleware implements intermittent security using a "Device ID", instead of using the 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 module allows fewer security handshake steps that result in time savings, and the data to be transmitted can still successfully be transferred without compromising security.

A possible major concern in a session resumption implementation is the possibility of Replay Attacks [24]. 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.

Data Encryption is commonly done using keys established through Public Key Cryptography (PKC). Instead of using PKC, our middleware chooses to leverage 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, [20]. Despite

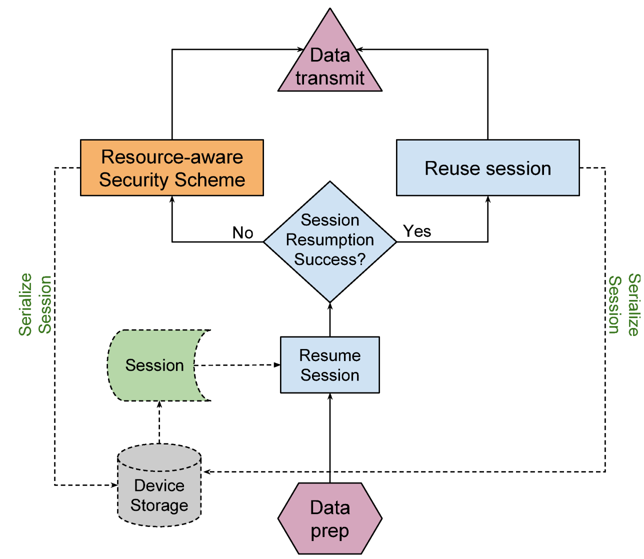


Figure 3: Illustration of Intermittent Security handling with Session Resumption [23]

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 preferable scheme for most use cases, allowing a tradeoff balance between speed and security requirements of an IoT-based application [25].

4.2. Logic Implementation

Figure 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 with device resource-awareness 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.

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 ← flex_security_vector(dID, {p, auth, en, mac})
end
function resume ()
    /* Pulls the stored session and uses it as new session */
    session ← stored_session
end
function serializeSession (x)
    /* Store the session in storage of member devices */
    while true do
        sleep (x)
        stored_session ← 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
```

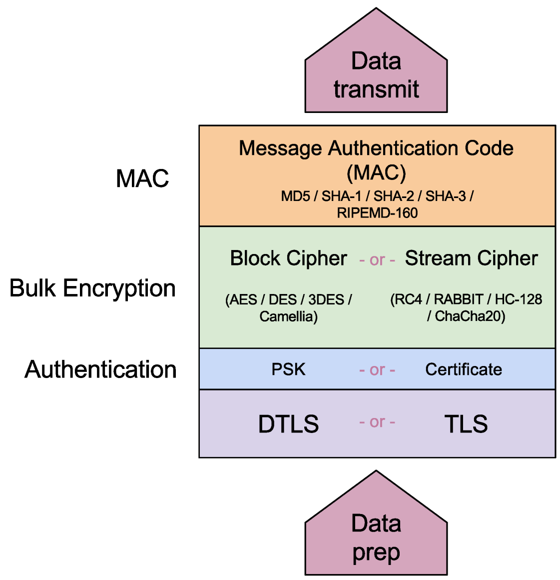


Figure 4: Illustration of Resource-aware Security handling with Protocol Selection [23]

5. Resource-aware 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 category. The categories include: (i) Protocol, (ii) Authentication Scheme, (iii) Bulk Encryption Scheme, and (iv) Message Authentication Code (MAC) algorithm as shown in Figure 4.

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.

Authentication Scheme Selection: Authentication Scheme can be either {PSK, Certificate}. Furthermore, each of these entities can be 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 significant amount 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, and 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, or DH (Diffie-Hellman).

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 significant amount 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.

Message Authentication Code Selection: Lastly, the chosen Message Authentication Code algorithm is used to generate a 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 to be vulnerable to various checksum attacks [26, 27] and collision attacks [28]. Through permutation and combination, the possible choices for the security scheme can be a large collection. Tables 1 shows a subset of candidate security schemes, and Table 2 provides a brief description for each of these schemes.

Table 1: A subset of candidate security schemes for *resource-aware security*

Security Scheme	**Protocol**	**Authentication**	**Encryption**	**MAC**
DTLS_PSK_WITH_CHACHA20_SHA256	DTLS	Static PSK	ChaCha20	SHA2(256)
DTLS_DHE_WITH_NULL_SHA384	DTLS	Certificate	-	SHA2(384)
DTLS_DHE_PSK_WITH_3DES_EDE_SHA	DTLS	PSK	3DES (EDE)	SHA1
TLS_PSK_WITH_AES_128_CBC_SHA	TLS	Static PSK	AES128(CBC)	SHA1
TLS_PSK_WITH_CHACHA20_POLY1305	TLS	Static PSK	ChaCha20	POLY1305
TLS_ECDHE_WITH_AES_256_GCM_SHA384	TLS	Certificate	AES256(GCM)	SHA2(384)

Table 2: Key use-cases and description of schemes in Table 1 for *resource-aware security*

Security Scheme	**Description**
DTLS_PSK_WITH_CHACHA20_SHA256	Very fast, secure. Excellent for secure video streaming
DTLS_DHE_WITH_NULL_SHA384	Fast scheme, high security
DTLS_DHE_PSK_WITH_3DES_EDE_SHA	Fast, but risk of integrity loss due to SHA1
TLS_PSK_WITH_AES_128_CBC_SHA	Fast, highly secure, suitable for moderately heavy data
TLS_PSK_WITH_CHACHA20_POLY1305	Fast, highly secure, suitable for quick bulk data transfer
TLS_ECDHE_WITH_AES_256_GCM_SHA384	Very high security, suitable for confidential data on a reliable network

5.1. Solution Approach

As discussed in the previous subsection, our middleware allows picking and choosing of security protocol components (e.g., bulk encryption, message authentication) as per the user's requirements. However, when it comes to actually choosing the best scheme option for an application context, the decision process is quite difficult. It can be even more onerous given 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 various schemes, their advantages or disadvantages. Consequently, users need to be presented with relevant information of suitable options to choose the best scheme option. To assist in this difficult decision process, the Optimal Scheme Decider solution of our middleware becomes relevant. We found ∼200 possible security scheme choices (that result from combinations of various protocol components) that the Optimal Scheme Decider needs to be analyze to find the optimal choice of security scheme for a given IoT-based application context. As part of the solution approach, we use supervised machine learning that is done in two phases: (i) *Offline Phase*, and (ii) *Online Phase*.

5.1.1. Offline Phase

The Offline Phase is a step used to narrow down the searchable space of security schemes by filtering our ∼200 scheme set database. Although the schemes can be quite varied, there exist many schemes that have close similarities in 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 permutation or combination of protocol components is not a necessary option for IoT-based application developers. In fact, as shown in Figure 5, we are able to group security schemes based on their protocol choices.

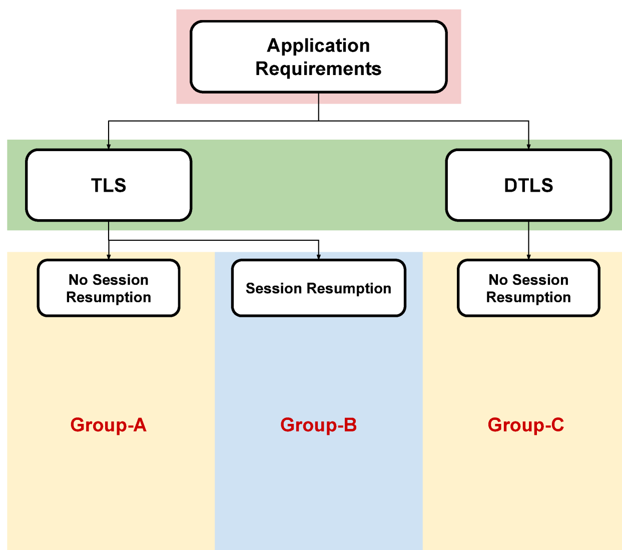


Figure 5: Scheme Groups formed based on various protocol component configurations; the groups can be used to categorize future clusters of security schemes to choose the optimal scheme

As a proof-of-concept, we implemented three groups {Group-A, Group-B, Group-C} that are significant. 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 comprises of all DTLS schemes with no session resumption. Session Resumption can be an optional configuration because there are trade-offs in having this feature. The resumption feature itself requires some periodic disk I/O to serialize active sessions. In addition, the serialized files are stored on the device, utilizing notable disk space. This process also ends up consuming notable energy. Thus, in cases of highly constrained IoT devices, not having session resumption enabled within the middleware may be beneficial.

Since our server is considered to be Full Function Device (i.e., a device that is enterprise data center grade and is generally not limited in terms of resources and function capabilities), 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 Optimal Scheme Decider.

Table 3: Parameters used for client benchmark analysis and their descriptions. These parameters are utilized in Offline and Online phases

Parameter Label	**Client Parameter**	**Description**	**Desired Level for RFD**
P1	CPU Usage	The security scheme's CPU usage	Smaller value
P2	Bandwidth Usage	The bandwidth of the security schemes network usage	Smaller value
P3	Peak Bytes	The peak memory consumption on device	Smaller value
P4	Connection Time	The elapsed time to successfully connect with server	Smaller value
P5	TX	The speed of transmitting data	Higher value
P6	RX	The speed of receiving data	Higher value

Table 3 shows the parameters used for client benchmark analysis in our database. The desired level represents whether high value is optimal, or lower, when operating on a Reduced Function Device (i.e., a sensor device that is highly constrained in terms of available resources and functioning capabilities). These 6 metrics under the 'Client Parameter' column in Table 3 can next be used to form the cluster of devices. By calculating the mean values of each variable for each cluster, we can classify them into intermediate categories. For instance, if the mean value of the bandwidth for a specific cluster is larger than other clusters, we classify this cluster as "security configuration with high bandwidth need". The formation of the cluster of devices is thus fuzzy in nature, but aligns well with our approach for the security scheme selection. 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*
   Note: The parameter labels $\{P1, P2, P3, P4, P5, P6\}$ can be collectively referred to as members of set *"P"*. 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 our raw data using the scaling formula:

$$x' = \frac{x_{ij} - x_{mj}}{\sigma_j} \tag{1}$$

where

x' : scaled value
x_{ij} : observations or row vectors of j^{th} parameter, where $j \in P$
x_{mj} : mean value of j^{th} parameter, where $j \in P$
σ_j : standard variance for j^{th} parameter, where $j \in P$

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 needed number of clusters is selected by a commonly-used statistic testing index viz., *cumulative proportion of variance explained*. Usually, when this index is around 80%, that number of clusters is the appropriate one. After deciding how many clusters we need to separate at a broad level, we apply the k-means clustering to see what security combinations should be distributed to which cluster. After receiving the k-means outcome, we pick the security combination which best represents the property of a specific cluster based on: (a) our knowledge of the properties of each cluster, and (b) the mean values from the PCA result and the security combinations that are contained in each cluster. The formula for each potential cluster can be as below:

$$z_{i1} = \phi_{11} \cdot x_{i1} + \phi_{12} \cdot x_{i2} + \cdots + \phi_{1p} \cdot x_{ip} \tag{2}$$
$$z_{i2} = \phi_{21} \cdot x_{i1} + \phi_{22} \cdot x_{i2} + \cdots + \phi_{2p} \cdot x_{ip} \tag{3}$$
$$\cdots$$
$$z_{ip} = \phi_{p1} \cdot x_{i1} + \phi_{p2} \cdot x_{i2} + \cdots + \phi_{pp} \cdot x_{ip} \tag{4}$$

where

z_{ip} : final adjusted data in i^{th} row and p^{th} parameter ($\in P$) that make eigen vectors perpendicular to each other
ϕ : eigen vectors that make covariance matrix have length 1
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:

$$\begin{aligned} &\underset{\phi_{11},\cdots,\phi_{p1}}{\text{maximize}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1} \cdot x_{ij} \right)^2 \right\} \\ &\text{subject to} \qquad \sum_{j=1}^{p} \phi_{j1}^2 = 1. \end{aligned} \tag{5}$$

Figure 6 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 bandwidth, and so on. Similarly, principal component analysis is generated for Group-B and Group-C.

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 6: Group-A principal component raw 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:

$$\sum_{j=1}^{p} \mathrm{Var}\left(X_j\right) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2 \tag{6}$$

The PVE result for each group is shown in Figures 7, 8, and 9. Analyzing Group-A PVE in Figure 7 indicates that 89.1% (∼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 8 suggests taking **four** clusters to encompass 90.4% (∼90%) of the schemes. Similarly, Group-C can benefit by housing **four** clusters, i.e., 90.6% (∼90%) of the schemes. Hence, we accept 3 clusters in Group-A, and 4 clusters each in Group-B and Group-C, as shown in Figure 10. We remark that Figure 10 illustrates a proof-of-concept method to create various categories of security schemes, and for an optimal scheme to flexibly customize as per an application's needs. In the specific case of Figure 10, the cluster formation is based on our proof-of-concept implementation with *wolfSSL* [29] for crypto and authentication functions with ∼200 security schemes. We choose *wolfSSL* owing to its lightweight SSL/TLS libraries that are small, portable and standardized for development with embedded system devices. However, our security framework can work with other categories of security schemes based on any other choice of implementation stacks (e.g., *OpenSSL*) suitable to cater to any application needs.

[A] **0.493** **0.766** **0.891** 0.966 0.989 0.996 0.998 0.999 1.000 1.000 1.000

Figure 7: Group-A proportion of variance explained results. Forming 3 clusters is sufficient to include ∼90% of the schemes

[B] **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 8: Group-B proportion of variance explained results. Forming 4 clusters is sufficient to include ∼90% of the schemes

[C]	**0.495**	**0.698**	**0.825**	**0.906**	0.961	0.981	0.991	0.995	0.998	1.000	1.000

Figure 9: Group-C proportion of variance explained results. Forming 3 clusters is sufficient to include ∼90% of the schemes

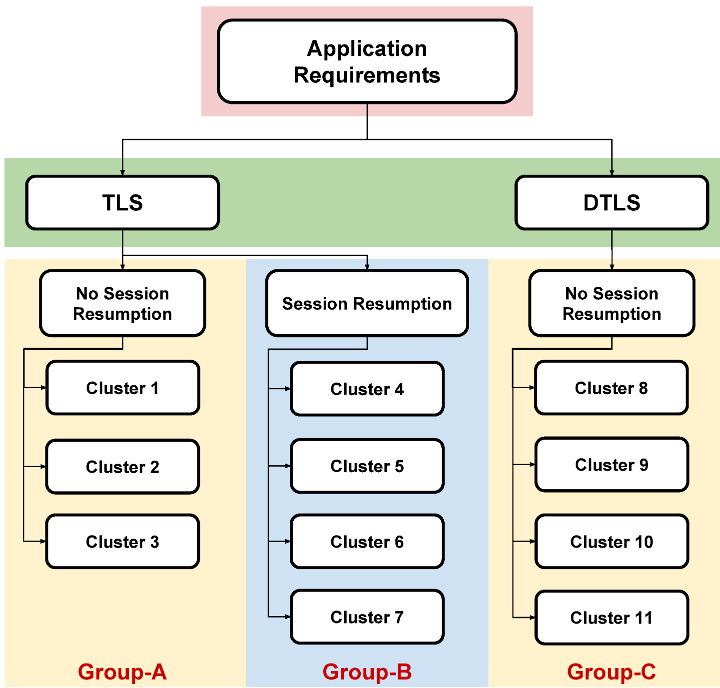


Figure 10: Number of clusters per group, calculated using proportion of variance explained in Step 4

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:

$$W\left(C_k\right) = \frac{1}{|C_k|} \sum_{i,i' \epsilon C_k} \sum_{j=1}^{p} \left(x_{ij} - x_{i'j}\right)^2 . \qquad (7)$$

where

$W(C_k)$: within-cluster variation, square Euclidean distance
C_k : the amount by which observations within a cluster differ from each other
$x_{ij} - x_{i'j}$: the distance between different observations, Euclidean distance

We want to cluster together the observations with minimum distance between them. Hence, we find the minimum distance in Equation 7:

$$\underset{C_1,\cdots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W\left(C_k\right) \right\}. \tag{8}$$

where

C_k : set of observations in cluster k

In essence, clustering allows us to reduce the overhead of relying on use of hundreds of redundant security schemes in our database, and instead finds the best alternative from a smaller scope containing equivalent schemes. We are able to choose the most optimal cluster in each group, hence allowing a small subset of schemes to be used by our middleware in an Online Phase (i.e., in real-time application use context).

5.1.2. Online Phase

For the online phase, the non-redundant schemes shortlisted in the offline phase can be accessed using a set of RESTful APIs by our middleware integration on an IoT device, gateway, or a core cloud. Figure 11 shows a tabular listing of available APIs for use in our middleware. The Optimal Scheme Decider acts as our decision making engine which finds the optimal security scheme for a given application use case and resource awareness. The metrics for resource awareness that we consider are: Type of Data, Estimated size of data transfer, Energy available to the device, Computation power available, Memory available, and 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 12.

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 11: A subset of the RESTful APIs available during Online Phase; Connected devices in the IoT system may invoke the API methods to collect required target device or parameter information

5.2. Logic Implementation

Once the internal map has been generated as detailed above, the Optimal Scheme Decider algorithm shown in Algorithm 2 is invoked to filter good candidate schemes from the cluster chosen in the Offline Phase. We can see the various filters applied in the algorithm to narrow down to a single security scheme from the input metrics. We decided to choose Stream Cipher as the preferred option 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.

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 12: RESTful API Parameter Values. The acceptable JSON values are mapped internally as noted in the Description column

In the algorithm, once the data type and size have been used to filter according to protocol and cipher type, subsequent decisions are based on the device specifications. 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, a low energy availability and extremely-high security scheme will not be combined together. Three major threshold variables have been set up to make certain decisions, mapped on a scale of {*1-10*}. The threshold value to decide whether the device has enough energy available to support complex schemes can be set using the variable *iotEnergyThreshold*. E.g., a threshold of *Level-4* implies any device with energy value below *4* is not to be handed a heavy security scheme. Similarly, *lowCompMemThreshold* & *medCompMemThreshold* are used to filter low or medium security requirements. The best scheme is picked based on the limiting value between available CPU and 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 is the choice of scheme that gets used by the middleware to initiate a secure session.

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
Data: On a scale of 1-10, threshold used to decide low energy availability,
      iotEnergyThreshold
Data: On a scale of 1-10, threshold used to decide low memory/computation
      availability, lowCompMemThreshold
Data: On a scale of 1-10, threshold used to decide medium memory/computation
      availability, medCompMemThreshold

Result: The best security scheme is chosen

/* data.PACKET_SIZE in bits */
if data.TYPE = MULTIMEDIA or data.PACKET_SIZE < 128 then
    protocol ← DTLS
    cipher.TYPE ← STREAM
else
    protocol ← TLS
    cipher.TYPE ← BLOCK
end

if energyLevel < iotEnergyThreshold then
    Eliminate heavy encryption schemes
end

if min (cpuLevel, memLevel) < lowCompMemThreshold then
    Choose low-level security scheme
else if min (cpuLevel, memLevel) < medCompMemThreshold then
    Choose medium-level security scheme
else
    Choose high-level security scheme
end
```

6. Performance 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 sets of experiments: (i) Validation of Intermittent Security, and (ii) Validation of Resource-aware Security.

6.1. Intermittent Security Validation Results

Our first set of experiments aim 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. For this, we use a prototype implementation of our middleware based on wolfSSL [29] within a GENI [30] Cloud testbed as shown in Figure 13. Live video stream is supported using OpenCV [31]. The application itself is built completely using C/C++, using GCC compiler. Our implementation consists of a client and server prototype. 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.

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 category. The categories include Protocol, Authentication Scheme, Bulk Encryption Scheme, and Message Authentication Code (MAC) algorithm. Our experiments account 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.

Figure 14(a) gives the graph generated by comparing different cipher schemes. The schemes we compared are Datagram TLS (DTLS), and TLS. The schemes were evaluated using Pre-shared Keys (PSKs) and certificates. Likewise, Figure 14(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,

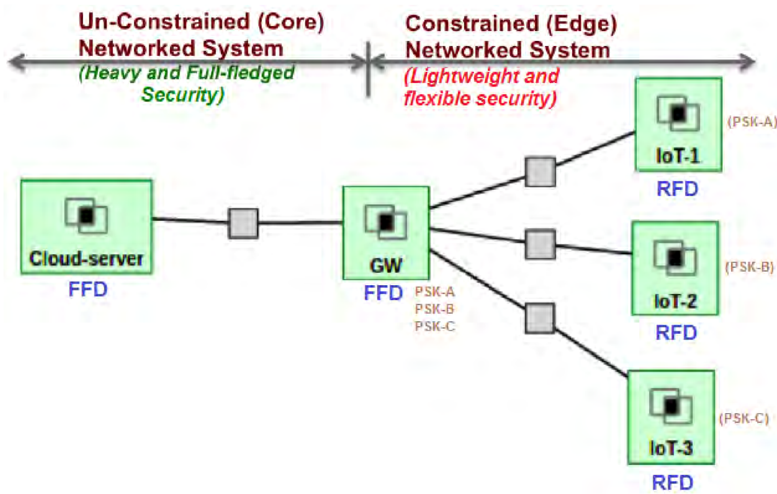


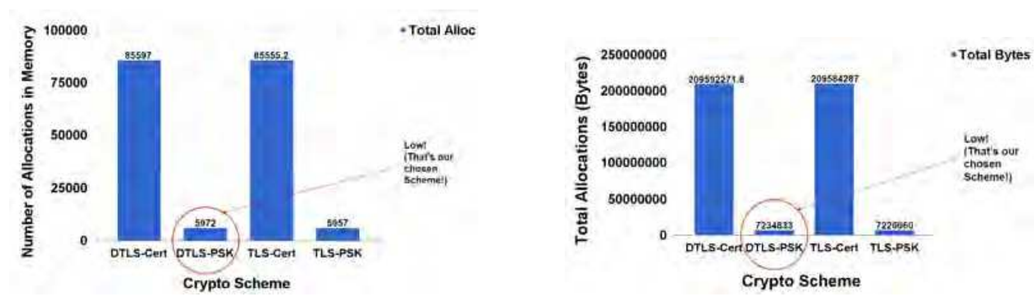
Figure 13: Testbed setup over the GENI Cloud Infrastructure

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

6.2. Resource-aware Security Validation Results

In our first set of experiments describe above, we established the usefulness of utilizing Session Resumption, as well as having the option to choose from a few different security schemes. In our second set of experiments, we aim to expand on our results and investigate our middleware's ability to form clusters from almost 200 security schemes. To test the offline phase detailed in Section 5.1.1, we check validity of: (i) 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.



(a) Number of memory allocations (b) Memory allocated (in bytes)

Figure 14: Memory footprint for different encryption schemes. Our chosen scheme utilizing DTLS-PSK shows the most promising result in terms of memory consumption

Figure 15: Time for connection vs. resumption for different encryption schemes. Our chosen scheme DTLS-PSK has very fast connection, and the fastest reconnection, due to Session Resumption support

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 Section 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 16, 17, and 18 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 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 for Group-A are shown in Figure 19. Similarly, Group-B and Group-C have their own schemes for their corresponding chosen cluster.

```
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:
[A] 71.84645 17.85738 47.12641
 (between_SS / total_SS = 61.1 %)
```

Figure 16: Within-cluster sum of squares for Group-A. Value of 61.1%(>60%) shows successful clustering

```
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:
[B] 42.004220 56.013587 4.281206 54.327632
 (between_SS / total_SS = 69.4 %)
```

Figure 17: Within-cluster sum of squares for Group-B. Value of 69.4%(>60%) shows successful clustering

```
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:
[C] 20.94528 46.51529 44.52632 21.46218
 (between_SS / total_SS = 62.1 %)
```

Figure 18: Within-cluster sum of squares for Group-C. Value of 62.1%(>60%) shows successful clustering

Figures 20, 21, and 22 show the selection of best cluster in each category, as chosen by the Optimal Scheme Decider's offline phase. We can see the clusters in each group that are formed after normalization of the computation index. The computation index is a compilation of the benchmark metrics used for clustering and decision making. Each of the chosen clusters have their advantages and disadvantages, which can be filtered in the online phase by the Optimal Scheme Decider.

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 Figure 23, multiple schemes in

#	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 19: Chosen cluster's security schemes categorized in Group-A

each group have siblings at the same level. This implies extreme similarities in the sibling schemes, and an absence of considerable performance benefit. This redundancy can be easily handled by randomly picking one of the leaves in the dendrogram and discarding the other scheme.

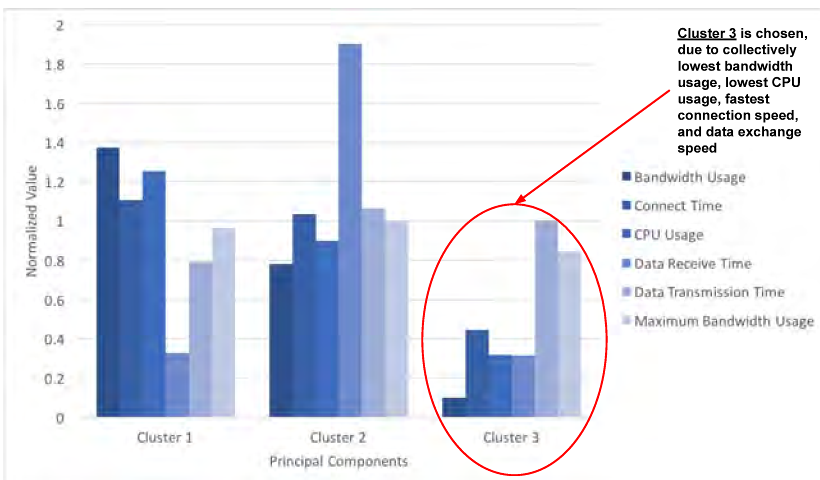


Figure 20: Chosen cluster in Group-A, by the Optimal Scheme Decider

Finally, the online phase requires collecting software requirements for the IoT-based application to be utilized. The current requirement collection can be done by making an API call to `/api/gateway/:device_id` service endpoint. The application user may send the required information using a parameter list to the gateway. The response from these queries help form the metrics that are later used by the Optimal Scheme Decider. In our related set

Figure 21: Chosen cluster in Group-B, by the Optimal Scheme Decider

of experiments, we tested the Optimal Scheme Decider by using random values as input requirements to check the scheme chosen by our middleware. As expected, for a low-security and low-resource multimedia application, PSK-CHACHA20 combination was found to be the optimal choice. To analyze the trade-off characteristics for various use-cases of IoT-based applications, we utilize radar diagrams for visual representation of the results. For instance, as shown in Figure 24(a), PSK-CHACHA20 scheme demands substantial memory resources, and exhibits fast re/connection speed but at the expense of a lower level of security.

Similarly, results were analyzed to highlight a contrasting case. The new requirement 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 24(b) 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 experiment results, we observed the radar diagram to visualize the trade-offs and analyze when session resumption is useful, and when it is not. For instance, Figure 25(a) shows the specifications for a chosen scheme for an application requiring high reconnection speed through

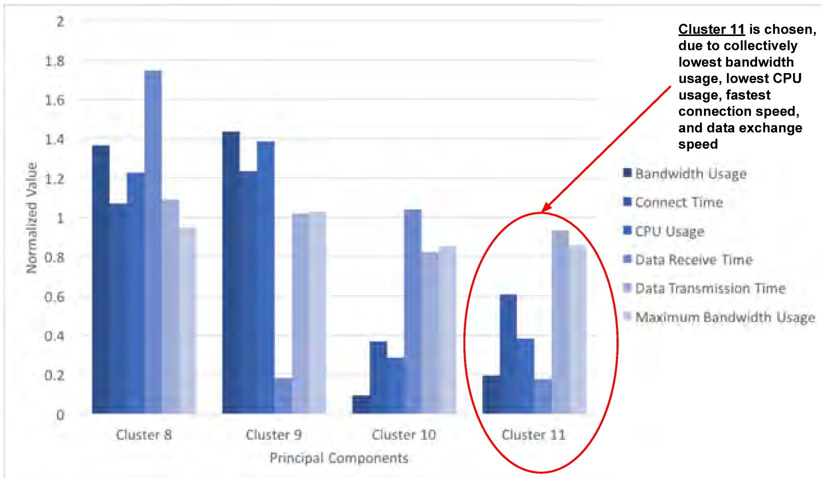


Figure 22: Chosen cluster in Group-C, by the Optimal Scheme Decider

session resumption, having lower memory and security requirements. This application appears to be a good candidate for 'austere-edge' (constrained network-edge with limited resource availability) IoT-based application use-cases. Contrasting this application to a 'smart edge' (unconstrained network-edge with enterprise data center resource availability), Figure 25(b) shows the corresponding application's chosen scheme trade-off specifications. We can observe that a very high level of security can be obtained at expense of faster reconnection.

Similarly, for another austere-edge IoT-based application, chosen scheme's specifications are visualized in Figure 26(a). Very high reconnection speed appears to be a requirement, hence speed is preferred over security. And finally, for a similar application without session resumption support, the chosen scheme specifications can be visualized in Figure 26(b). This application represents another 'smart-edge' IoT-based application, as an all-rounded security scheme is needed, with a higher priority for security and initial connection speed.

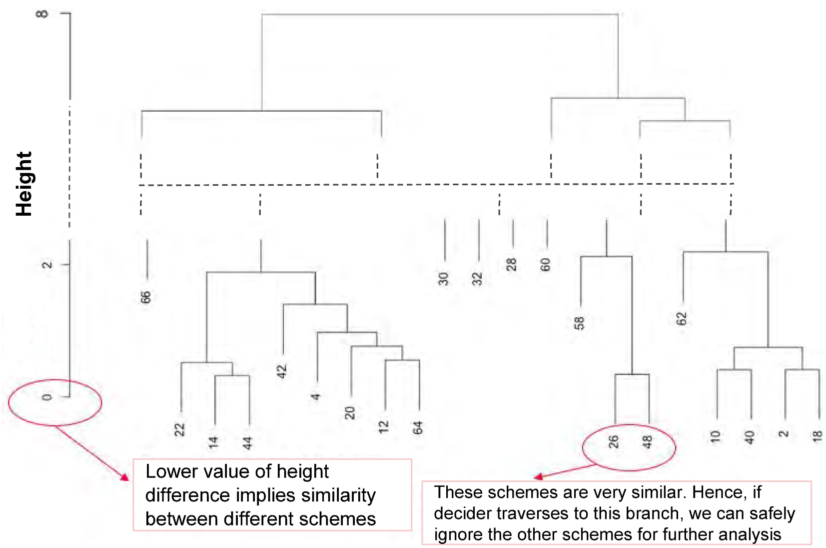
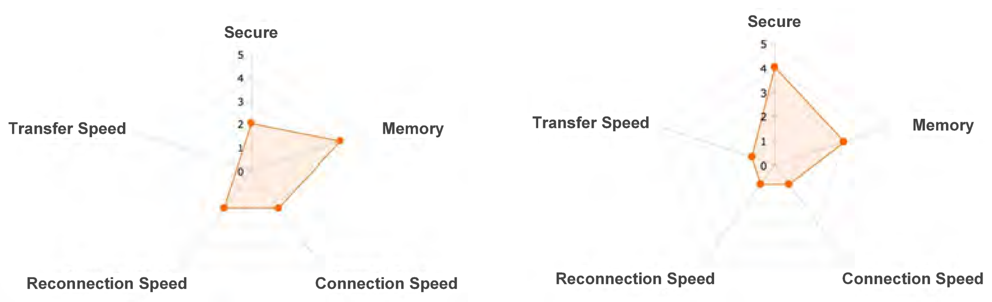


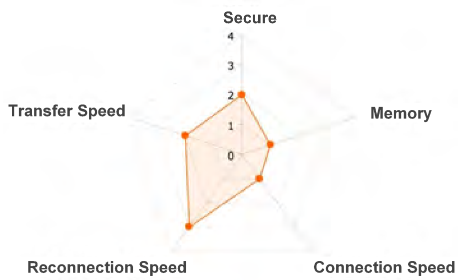
Figure 23: Group-A Dendrogram depicting reasoning for safely narrowing scope of search of optimal scheme in Offline Phase



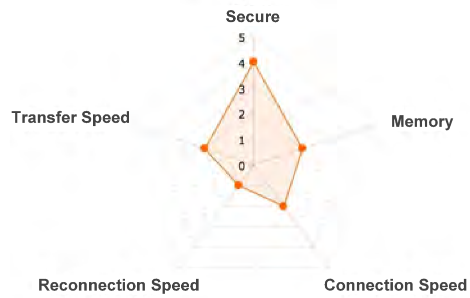
(a) Scheme *DTLS-PSK-CHACHA20-POLY1305*

(b) Scheme *DTLS-DHE-PSK-CHACHA20-POLY1305*

Figure 24: Radar representation of chosen schemes in two contrasting IoT applications

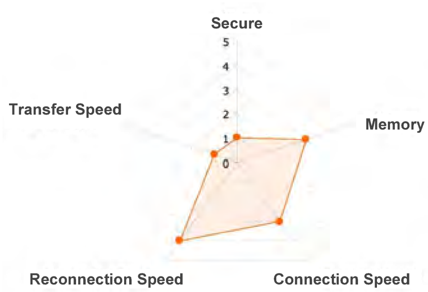


(a) Scheme *TLS-DHE-PSK-AES256-GCM-SHA384 w/ Session Resumption*

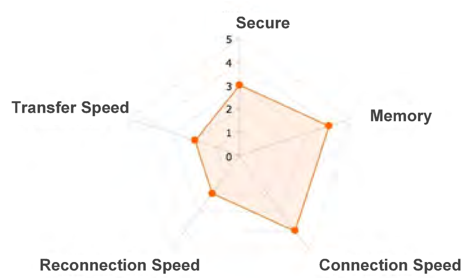


(b) Scheme *TLS-DHE-PSK-AES256-GCM-SHA384 w/o Session Resumption*

Figure 25: Radar representation of two very similar schemes, chosen for 'Austere-Edge' and 'Smart-Edge' respectively. Presence/Absence of resumption support allows different use-cases



(a) Scheme *TLS-PSK-AES256-CBC-SHA w/ Session Resumption*



(b) Scheme *TLS-PSK-AES256-CBC-SHA w/o Session Resumption*

Figure 26: Radar representation of two more schemes, chosen for 'Austere-Edge' and 'Smart-Edge' respectively. In certain cases, lack of resumption might prove more useful, as shown in (b)

7. Conclusion

In this paper, we developed a flexible IoT security middleware that can be used in end-to-end cloud-fog communications involving smart devices at the network-edge and a cloud-hosted application. The end-to-end feature is possible due to the middleware's task of translating protocols between two different network segments (i.e., cloud-to-gateway and gateway-to-edge), hence resulting in full security without being limited by protocol compatibility between the networks. Our middleware features two major modules, the first of which is the 'Intermittent Security' module that offers quick re-connections to aid in situations of unreliable network conditions within cloud-fog communication platforms. The second module handles 'Flexibile Security', which provides flexibility in choosing an optimal security scheme that is suitable for securing an IoT-based application depending upon user's requirements and edge-resource constraints.

Our results from testbed experiments with a prototype implementation of our middleware validate its benefits. We showed 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, our work demonstrated that they could be a valuable tool for low-resource, moderate-security applications within IoT systems in both 'austere' and 'smart' network-edges, even for unreliable networks with frequent disconnections. Our investigations have also shown that it is possible to find equivalence amongst various kinds of security schemes available, and there is potential to have real-time advice to select the optimal scheme for a given set of constraints in an IoT application. Towards this end, we have developed an optimal scheme decider which is capable of using a security scheme database to find and use the optimal security scheme for a given set of constraints. This has been done by using a new set of RESTful APIs that we created, which allows our middleware to collect information about data and devices involved in the network. We also used supervised machine learning to narrow down on subsets of appropriate security schemes by considering different trade-offs involved in IoT-applications.

Future work can involve extending our middleware with a transient reputation scheme that can collect and use short-term knowledge about the connecting devices to build a short-lived reputation. This would alleviate the need to maintain trust states in the network-edge amongst the IoT devices.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Award Number: CNS-1647182. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] IoT Trends - Tech Insider, 2016. 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 coordination", 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", IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (2015).

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

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

[6] P. Calyam, A. Mishra, R. Bazan Antequera, D. Chemodanov, A. Berryman, K. Zhu, C. Abbott, M. Skubic, "Synchronous Big Data Analytics for Personalized and Remote Physical Therapy", Elsevier Pervasive and Mobile Computing (2015).

[7] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions", Future generation computer systems 29 (2013) 1645–1660.

[8] M. Ali, N. Ono, M. Kaysar, Z. Shamszaman, T. Pham, F. Gao, K. Griffin, A. Mileo, “Real-time data analytics and event detection for IoT-enabled communication systems”, Web Semantics: Science, Services and Agents on the World Wide Web 42 (2017) 19–37.

[9] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos, “Sensing as a service model for smart cities supported by internet of things”, Transactions on Emerging Telecommunications Technologies 25 (2014) 81–93.

[10] S. Sicari, A. Rizzardi, L. Grieco, A. Coen-Porisini, “Security, privacy and trust in Internet of Things: The road ahead”, Computer Networks 76 (2015) 146–164.

[11] 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).

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

[13] H. Khemissa, D. Tandjaoui, “A novel lightweight authentication scheme for heterogeneous wireless sensor networks in the context of Internet of Things”, in: Wireless Telecommunications Symposium, WTS 2016, London, United Kingdom, April 18-20, 2016, pp. 1–6.

[14] M. Turkanovi, B. Brumen, M. Hlbl, “A novel user authentication and key agreement scheme for heterogeneous ad hoc wireless sensor networks, based on the Internet of Things notion”, Ad Hoc Networks 20 (2014) 96 – 112.

[15] 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).

[16] C. Stergiou, K. E. Psannis, B. G. Kim, B. Gupta, “Secure integration of IoT and Cloud Computing”, Future Generation Computer Systems (2016).

[17] S. R. Moosavi, T. N. Gia, E. Nigussie, A. M. Rahmani, S. Virtanen, H. Tenhunen, J. Isoaho, “Session Resumption-Based End-to-End Security for Healthcare Internet-of-Things”, IEEE International Conference on Computer and Information Technology (2015) 581–588.

[18] S. R. Moosavi, T. N. Gia, E. Nigussie, A. M. Rahmani, S. Virtanen, H. Tenhunen, J. Isoaho, “End-to-end security scheme for mobility enabled healthcare Internet of Things”, Future Generation Computer Systems 64 (2016) 108 – 124.

[19] 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).

[20] 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), 9th Annual IEEE International (2015).

[21] R. Amin, G. P. Biswas, “A secure light weight scheme for user authentication and key agreement in multi-gateway based wireless sensor networks”, Ad Hoc Networks 36, Part 1 (2016) 58 – 80.

[22] 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).

[23] B. Mukherjee, R. Neupane, P. Calyam, “End-to-End IoT Security Middleware for Cloud-Fog Communication”, IEEE Cyber Security and Cloud Computing (CSCloud) (2017).

[24] P. Syverson, “A Taxonomy of Replay Attacks [cryptographic protocols]”, Computer Security Foundations Workshop VII. (1994).

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

[26] J. A. Dev, "Usage of Botnets for High Speed MD5 Hash Cracking", International Conference on Innovative Computing Technology (INTECH) (2013).

[27] D. Lee, "Hash Function Vulnerability Index and Hash Chain Attacks", IEEE Workshop on Secure Network Protocols, NPSec (2007).

[28] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, Y. Markov, "The first collision for full SHA-1", Accessed June 2017. https://shattered.io/static/shattered.pdf.

[29] wolfSSL SSL library, Accessed June 2017. https://www.wolfssl.com/wolfSSL/Home.html.

[30] NSF-supported GENI Cloud Infrastructure, Accessed June 2017. https://www.geni.net.

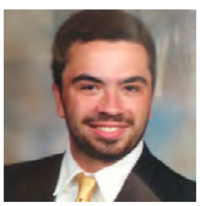
[31] OpenCV library, Accessed June 2017. http://opencv.org.

Bidyut Mukherjee received his MS degree in Computer Science from University of Missouri-Columbia in 2017, and BE degree in Computer Technology from RTM Nagpur University, India in 2015. His current research interests include: Cloud Computing, Computer Networking, Internet of Things, and Cyber Security. He is a student member of IEEE.

Songjie Wang is currently a graduate student pursuing MS degree in the Department of Computer Science at the University of Missouri-Columbia. His current research interests include: Cloud Computing, Internet of Things and Data Science.

Wenyi Lu received her MS degree in Statistics from University of Missouri-Columbia in 2015. Currently, she is a PhD student in the Department of Computer Science at MU. Her current research interests include: Serious-game Design and Development, Deep-Learning analytics and Data Mining technologies.

Roshan Lal Neupane received his BE degree in Computer Science and Engineering from Visvesvaraya Technological University, Karnataka, India. He is currently a graduate student at University of Missouri-Columbia, pursuing MS in Computer Science. Cloud Computing, Computer Networking, Internet of Things, and Cyber Security. He is a student member of IEEE.



Daniel Dunn received his BS degree in Computer Science from the University of Missouri-Columbia in 2017. He remains a student there with the expectation to receive his MS degree in Computer Science in 2018 as a student in the 5-year BS/MS Computer Science Fast Track Program. He is a member and the Vice President of Upsilon Pi Epsilon Computer Science Honors Society. His current research focus is in Cyber Security.

Yijie Ren is now earning her MS degree in Computer Science at University of Missouri-Columbia. She also received her BS Degree in Computer Science at MU, and BE Degree in Software Engineering at Taiyuan University in China. Her research interests include: Big Data Analytics and Cloud Computing.

Qi Su received her bachelor's degree in Communications from University of Missouri-Columbia in 2014. She is currently a MS student in the Department of Computer Science at MU. Her current research interests include: Cloud Computing and Bioinformatics.

Prasad Calyam received his MS and PhD degrees from the Department of Electrical and Computer Engineering at The Ohio State University in 2002 and 2007, respectively. He is currently an Assistant Professor in the Department of Computer Science at University of Missouri-Columbia. Previously, he was a Research Director at the Ohio Supercomputer Center. His research interests include: Distributed and Cloud computing, Computer Networking, and Cyber Security. He is a Senior Member of IEEE.