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

Bidyut Mukherjee, Roshan Lal Neupane, Prasad Calyam
*University of Missouri-Columbia, USA*
{*bm346, rlnzq8*}*@mail.missouri.edu; calyamp@missouri.edu*

***Abstract*— IoT (Internet of Things) devices such as sensors have been actively used in 'fogs' to provide critical data during e.g., disaster response scenarios or in-home healthcare. Since IoT devices typically operate in resource-constrained computing environments at the network-edge, data transfer performance to the cloud as well as end-to-end security have to be robust and customizable. In this paper, we present the design and implementation of a middleware featuring "intermittent" and "flexible" end-to-end security for cloud-fog communications. Intermittent security copes with unreliable network connections, and flexibility is achieved through security configurations that are tailored to application needs. Our experiment results show how our middleware that leverages static pre-shared keys forms a promising solution for delivering light-weight, fast and resource-aware security for a variety of IoT-based applications.**

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

## I. INTRODUCTION

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.

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. As mentioned in [1], emerging IoT trends are set to completely change the way businesses, governments, and consumers interact with each other, and transact in a data-driven economy.

Since IoT devices typically operate in resource-constrained (computing, memory, storage) environments at the network-edge, data transfer performance to the cloud as well as end-to-end security have to be robust and customizable. Consider the use-case of disaster response systems such as [2]; the edge network here comprising of IoT devices and network gateways to the cloud (i.e., location of abundant resources) could be highly unstable due to physical infrastructure damage or lossy edge links. On the other end of the IoT use-case spectrum, the fog nodes might be used for providing ElderCare-as-a-Service, as in [3] that requires significant amount of resources to handle the big data generated from patient homes. In this case, security needs to be configured smartly for data confidentiality and integrity, even if data transfer speeds are affected due to security overhead. Thus, IoT-based applications can have extremely broad use-cases, and ad-hoc implementations may not be suitable to address the wide-ranging IoT-based application security needs.

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. 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 at a given network edge location. The core features of our end-to-end IoT security middleware, and the main paper contributions are:

- *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 connection.
- *Flexible Security*: Our middleware allows users to flexibly configure required security based on the application resource-awareness versus blindly following a rigid security configuration. This enables the user to thus configure higher security or prioritize faster data transfer.

The remainder of the paper is as follows: Section II discusses related works. Section III describes our end-to-end IoT security middleware scheme. Section IV presents our performance evaluation and findings from testbed experiments. Section V concludes the paper.

## II. RELATED WORK

IoT-based application deployment is a relatively new trend, however methods to secure networked IoT devices have been explored in the past. Work in [4] 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 [5]. Similarly, in [6], a light-weight authentication scheme is used in the context

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.

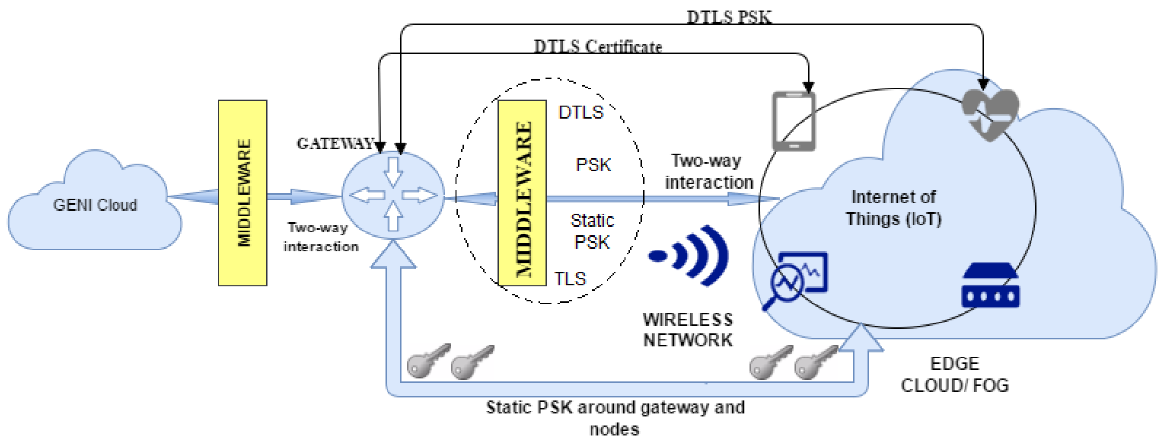


Fig. 1. Edge Cloud/Fog system architecture - GENI-Gateway forms an unconstrained network, whereas Gateway-IoT forms a constrained network segment

of IoT systems. 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 earlier works. We phase down to just one iteration for authentication and thus our approach reduces security overhead and is less time consuming.

Furthermore, [7] 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 key created, however, is ephemeral. They propose a selection criteria based on trust level to select the assisting nodes. The protocol is compatible with other end-to-end security protocols. Our work builds on top of this work, 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.

A comprehensive session resumption mechanism is discussed in [8]. 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.

Authors in [9] 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 a step further towards a more practical layer for use within IoT-based applications.

## III. IoT End-to-End Security Scheme

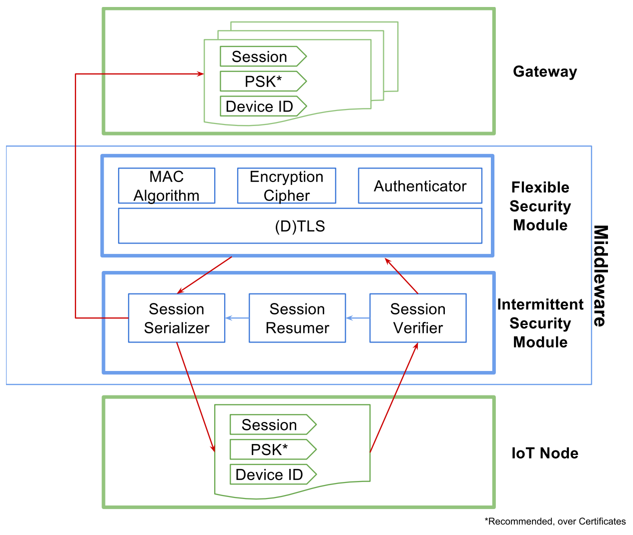


Fig. 2. End-to-End IoT Security Middleware Module Diagram

### A. Overview

As mentioned earlier, our work closely follows the idea of flexible security framework described in [9]. We build upon the previous framework by adapting their implementation for our core features suited for a variety of application use cases.

Figure 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, 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.

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 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.

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.

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.

## B. *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 [8], 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 [10]. 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.

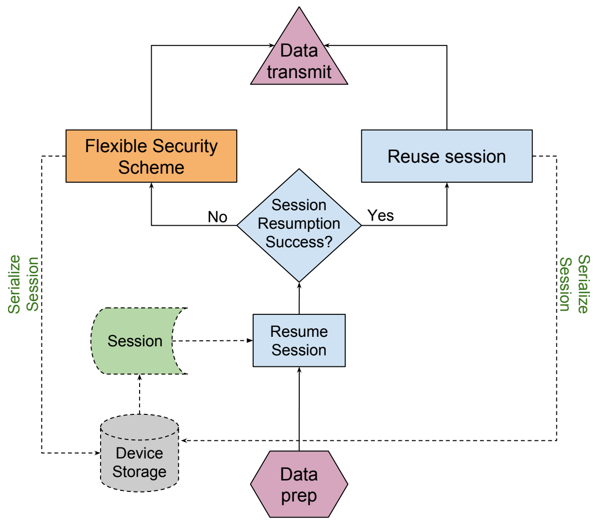


Fig. 3. Illustration of Intermittent Security handling with Session Resumption

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, [11]. 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.

TABLE I
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 |

Hence, it is a preferred scheme for most use cases, allowing a tradeoff balance between speed and security requirements of an IoT-based application [12].

Figure 3 shows a flowchart illustration of how our middleware handles 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:** 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
```

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.

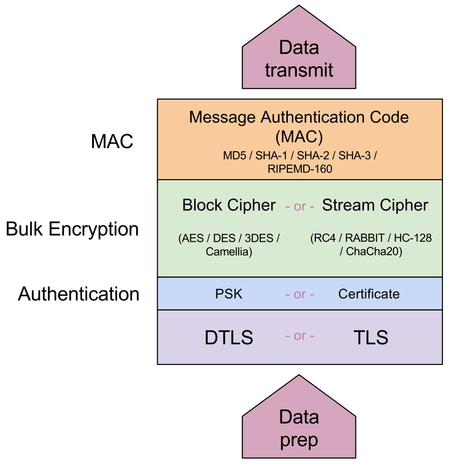

Data transmit
MAC
Message Authentication Code (MAC)
MD5 / SHA-1 / SHA-2 / SHA-3 / RIPEMD-160
Bulk Encryption
Block Cipher - or - Stream Cipher
(AES / DES / 3DES / Camellia)
(RC4 / RABBIT / HC-128 / ChaCha20)
Authentication
PSK - or - Certificate
DTLS - or - TLS
Data prep

Fig. 4. Illustration of Flexible Security handling with Protocol Selection

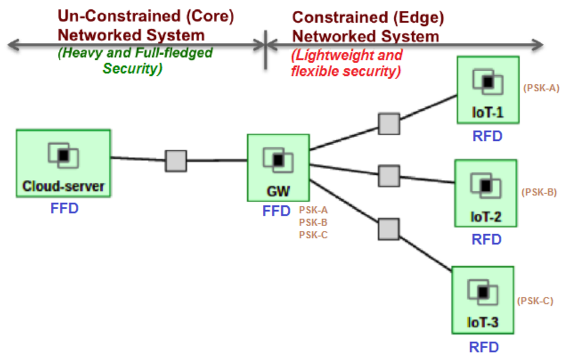
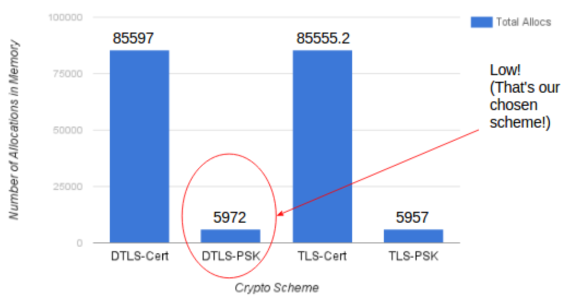
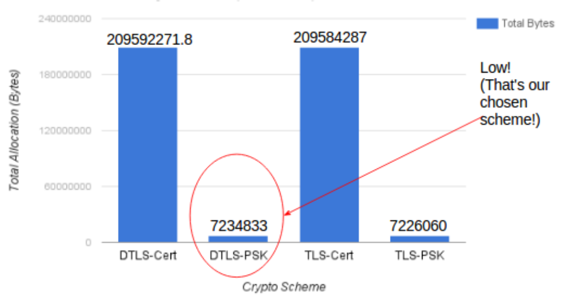

Un-Constrained (Core) Networked System
(Heavy and Full-fledged Security)
Constrained (Edge) Networked System
(Lightweight and flexible security)
Cloud-server
FFD
GW
FFD
PSK-A
PSK-B
PSK-C
IoT-1
RFD
(PSK-A)
IoT-2
RFD
(PSK-B)
IoT-3
RFD
(PSK-C)

Fig. 5. Testbed setup over the GENI Cloud Infrastructure



(a) Number of memory allocations



(b) Memory allocated (in bytes)

Fig. 6. Memory footprint for different encryption schemes; UCN has a server-client with full security, while the CN has a flexible security server-client setup

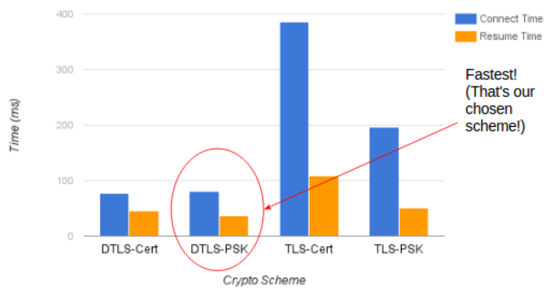


Fig. 7. Time for connection vs. resumption for different encryption schemes

### C. *Flexible Security: Protocol Selection*

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 as shown in Figure 4.

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 works over UDP (User Datagram Protocol). For use with TCP based applications, TLS is the preferred choice. Next, Authentication Scheme can be {PSK, Certificate}.

If PSK is chosen, 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.

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. 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.

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 ([13], [14]) and collision attacks [15]. Through permutation and combination, the possible choices for the security scheme can be many. Table I shows a few of the possible schemes.

## IV. Performance Evaluation

In this section, we compare the performance of various schemes accessible on our middleware to illustrate the need for flexibility. A minimum viable implementation of the middleware has been used on a GENI [16] Cloud testbed.

Figure 5 shows the network setup using the GENI Cloud infrastructure. To assess the lightweight nature of schemes, we use the following qualitative and quantitative metrics.

(a) Memory footprint (accounts for both Number of memory allocations and the Size of memory allocations)
(b) Time taken for Security Association

Our implementation of crypto and authentication uses WolfSSL [17], an embedded SSL implementation library. Live video stream is supported using OpenCV [18]. The application itself is built completely using C/C++, using GCC compiler.

Figure 6(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 6(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 7 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 the need for Intermittent security in IoT systems, without compromising the security, by allowing flexibility in configuration.

## V. Conclusion and Future Work

In this paper, 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.

Future work can extend 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 deprecate the need to maintain trust state in the network amongst the IoT devices.

## References

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

[2] 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 Mobile Cloud*, 2015.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

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

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

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

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

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