SmartEdge: An End-to-End Encryption Framework for an Edge-enabled Smart City Application

Mian Ahmad Jan[a,*], Wenjing Zhang[b,2,*], Muhammad Usman[c,*], Zhiyuan Tan[d,*], Fazlullah Khan[a], Entao Luo[e]

[a]*Department of Computer Science, Abdul Wali Khan University Mardan, Pakistan*
[b]*School of Information Science and Technology, Hebei Agricultural University, China*
[c]*Department of Computer Science and Software Engineering, Swinburne University of Technology, Australia*
[d]*School of Computing, Edinburgh Napier University, United Kingdom*
[e]*School of Electronics and Information Engineering, Hunan University of Science and Engineering, China*

Abstract

The Internet of Things (IoT) has the potential to transform communities around the globe into smart cities. The massive deployment of sensor-embedded devices in the smart cities generates voluminous amounts of data that need to be stored and processed in an efficient manner. Long-haul data transmission to the remote cloud data centers leads to higher delay and bandwidth consumption. In smart cities, the delay-sensitive applications have stringent requirements in term of response time. To reduce latency and bandwidth consumption, edge computing plays a pivotal role. The resource-constrained smart devices at the network core need to offload computationally complex tasks to the edge devices located in their vicinity and have relatively higher resources. In this paper, we propose an end-to-end encryption framework, SmartEdge, for a smart city application by executing computationally complex tasks at the network edge and cloud data centers. Using a lightweight symmetric encryption technique, we establish a secure connection among the smart core devices for multimedia streaming towards the registered and verified edge devices. Upon receiving the data, the edge devices encrypts the multimedia streams, encodes them, and broadcast to the cloud data

*Corresponding Authors
[1]mianjan@awkum.edu.pk
[2]zwjndjs@163.com
[3]musman@swin.edu.au
[4]Z.Tan@napier.ac.uk

centers. Prior to the broadcasting, each edge device establishes a secured connection with a data center that relies on the combination of symmetric and asymmetric encryption techniques. In SmartEdge, the execution of a lightweight encryption technique at the resource-constrained smart devices, and relatively complex encryption techniques at the network edge and cloud data centers reduce the resource utilization of the entire network. The proposed framework reduces the response time, security overhead, computational and communication costs, and has a lower end-to-end encryption delay for participating entities. Moreover, the proposed scheme is highly resilient against various adversarial attacks.
Keywords: Internet of Things, Smart City, End-to-End Encryption, Smart Devices, Network Edge, Cloud Data Centers.

1. Introduction

The Internet of Things (IoT) bridges the gap between the real-world ubiquitous devices and the virtual world of Internet [1]. This technological growth has the potential to transform communities around the globe into smart cities by creating a new era of urban life [2]. The vast deployment of heterogeneous sensors in smart cities generates huge volumes of real-time uninterrupted data streams [3]. These streams need to be efficiently analyzed to provide seamless delivery of various services to the inhabitants of smart cities [4]. To extend their lifetime, the resource-constrained sensors, also known as smart devices, need to offload computationally complex operations to the cloud data centers [5].

The delay-sensitive smart cities' applications have strict requirements in term of response time [6, 7]. These applications expect the responses to be received within the pre-defined deadlines. The conventional cloud computing is unable to meet such demands due to the presence of unreliable public networks between the smart devices and cloud data centers. The massive amount of data generated by smart devices supersede the storage and bandwidth capabilities of the cloud data centers. Processing huge volumes of data at the cloud results in a much higher latency and response time for smart devices, which is unacceptable for delay-sensitive applications [8]. In these

applications, edge computing plays a crucial role by bridging the gap among the smart devices and cloud data centers [9, 10]. The resource-constrained nature of these devices means that computationally complex and resource-consuming operations need to be offloaded to the edge devices and cloud data centers. The migration of computationally complex tasks ensures that applications experience lower latency and are able to provide high-quality services to the inhabitants of smart cities. The edge devices at the network edge are in proximity to smart devices and perform resource-intensive operations by prioritizing the processing of delay-sensitive services. Moreover, they offload the delay-tolerant services to the cloud data centers [11].

The massive deployment of smart devices and the presence of network edge in smart cities increases the reliability and scalability of services. However, as the number of connected devices increases, management of security credentials becomes a challenging issue [12]. It becomes impractical for each device to connect to a cloud data center to update its security credentials. In an end-to-end smart cities' application, all the participating entities need to be secured against various malicious threats. For example, the smart surveillance cameras ensure safety and security of inhabitants by reducing the overall criminal activities [3]. An adversary may eavesdrop on the images, audio and video clips, captured by these cameras to infer trajectories of smart cities' residents and inherently endanger their privacy. In smart cities, the network edge perform various tasks such as malware scanning and software updates, on behalf of resource-constrained smart devices. Unlike the smart devices, the network edge has abundant of resources and are often the target of various adversarial attacks [13]. A spyware-infected server in a cloud data center may steal security credentials that may allow an adversary to steal mission-critical data. The download of maliciously injected data from the cloud may further expose the residents to various vulnerabilities [14].

Although off-the-shelf encryption, authentication and anonymity algorithms might be applied directly to address these security and privacy challenges, the adversaries may still infiltrate a smart city environment. Most of these algorithms involve computationally complex cipher suites that require abundant of storage and computational power [15]. However, these smart devices are either battery-powered or rely on energy harvesting techniques [9]. To alleviate the escalation in resource congestion, computa-

tionally complex and resource-intensive operations need to be offloaded to the network edge [8]. Each edge device, enacting the role of a gateway, ensures an efficient utilization of the cloud data centers' bandwidth.

To protect the communication among smart core devices, edge devices, and cloud data centers, we propose an end-to-end encryption framework, SmartEdge, for seamless and reliable transmission of multimedia streams. The smart devices forward multimedia streams to edge devices for resource-intensive operations, i.e., encoding and encryption. The encoded and encrypted data are uploaded to the cloud for partial decryption that can be downloaded and fully decrypted by the end users, upon establishing a secured session. In view of limited resources of smart devices, most of the resource-intensive operations are performed at the edge and cloud data centers. The major contributions of SmartEdge are as follow.

1. A lightweight authentication framework is proposed to secure the transmission of multimedia streams among the smart devices, located at the network core. This framework relies on symmetric encryption, i.e., advanced encryption standard (AES) [16] with a key length of 128 bits, for establishing a secured session among the smart devices.
2. To verify the authenticity of edge devices, we propose a simple registration procedure. Prior to data collection from smart devices, each edge device registers itself with a base station that maintains a database of legitimate smart devices in a smart city application.
3. A relatively complex authentication framework is proposed to allow the edge devices to exchange data with cloud data centers. Using high efficiency video coding (HEVC) [17], each edge device encodes the data, and uploads to the cloud upon encryption, that can be downloaded by the end users. This framework executes an AES-256 bit at the network edge and an asymmetric algorithm, i.e., RSA 2048-bit, at the cloud end.

The rest of this paper is organized as follows. In Section 2, related works from the literature are provided. In Section 3, we provide the network model of our proposed SmartEdge framework. In Section 4, we present a lightweight authentication approach

adopted by SmartEdge at the network core. In Section 5, the registration and verification of SmartEdge at the network edge is presented. In Section 6, a secured data transmission and storage approach adopted by SmartEdge is presented. In Section 7, we provide the experimental results of our framework. Finally, the paper is concluded in Section 8.

2. Related Work

The use of IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) makes it possible to interconnect real-world physical devices with the Internet hosts, e.g. smartphones, laptops, desktops and computing clouds, to form the Internet of Things [18]. The Constrained Application Protocol (CoAP) is used as the de-facto application layer protocol for sensor-embedded smart devices of an IoT network [15]. These resource-constrained smart devices are connected to the cloud data centers to avail their computational and storage capabilities. To secure the exchange of data among the smart devices and data centers, CoAP mandates the use of Datagram Transport Layer Security (DTLS) protocol at the transport layer of each smart device [19]. However, the handshake and record layers of DTLS incur 25 bytes of overhead for each datagram header. The IEEE 802.15.4, on the other hand, specifies a physical layer Maximum Transmission Unit (MTU) of only 127 bytes. As a result, only 60-75 bytes are left for the payload after the addition of DTLS, Medium Access Control (MAC), and upper layers headers. The cloud data centers allow the smart devices to broadcast and store their measurements in a centralized location accessible by multiple hand-held devices. With an increase in urbanization, it is expected that a massive number of such devices will be connected to the Internet. The presence of these devices makes a city smarter by generating a huge amount of data that can be leveraged for various services such as, safety and privacy, infotainment, better health facilities, greener environment and better waste management [3, 20]. To realize these services, the voluminous data generated by smart devices need to be managed in an efficient manner at the edge of

the network. The resource-starving smart devices at the network core[5] need to offload the computational tasks to the network edge [8, 11, 9]. Once the data is processed at the edge, it is forwarded to the cloud for storage that can be used to provide better facilities to the residents of smart cities.

In literature, there exist various smart applications, such as CloudAware [21], Femtoclouds [22], EdgeIoT [23], ParaDrop [24] and HomeCloud [25] that offload computationally complex tasks to nearby edge devices. CloudAware and FemtoClouds monitor the usage of edge devices for offloading resource-intensive operations by relying on some specific application programming interfaces (APIs) and scheduling algorithms. EdgeIoT configures the neighboring mobile base stations by deploying virtual machines (VMs) for migrating various tasks from smart devices to the network edge. ParaDrop configures the neighboring wireless access points to offload various tasks to the edge. HomeCloud relies on Software Defined Networking (SDN) for scheduling its Network Function Virtualization (NFV) capabilities.

The aforementioned solutions [21, 22, 23, 24, 25, 8, 11, 9] for offloading computational tasks to the network edge focus mainly on scalar data. They lack the support for processing real-time multimedia streams such as video analysis. The existing works on video analytics support the offloading of streams directly to the cloud data centers that incur excessive burden on smart devices. Besides, most of the existing schemes for offloading computational tasks to the edge lack any support for security and privacy. The inclusion of edge computing in cloud-enabled IoT platforms raises new and unforeseen challenges such as, the interaction among heterogeneous edge devices, their interoperability with smart devices and the migration of tasks across global and local scales in smart cities. These challenges enable an adversary to launch various threats on the network's core, edge and cloud data centers. There exist very few studies in this context to combat such threats. In [26, 13, 14, 27], the authors conclude that data streams stored at the network edge are more vulnerable to threats in comparison to their storage at the cloud. To prevent such threats, differential privacy need to be used to protect users' privacy at the edge.

[5] due to the presence of sensor at the core of each smart device

3. Network Model

In a typical smart city, heterogeneous sensors, wired/wireless and cellular devices, edge devices, and cloud data centers are the essential components. The resource-constrained multimedia sensor nodes (MSNs) sense and collect the critical data streams pertaining to various applications. Using the IEEE $802.15.4$ standard, the MSNs transmit these data streams to the nearby cluster heads. One or more authorized edge devices collect these streams from cluster heads using the IEEE 802.11 standard. Finally, the data is uploaded to the cloud data centers that enable various organizations to provide reliable and timely services to the inhabitants of a smart city.

In our proposed SmartEdge framework, each participating device needs to be authorized for a seamless and reliable transmission of gathered data. In each application at the network core, the MSNs and their respective cluster head communicate with each other for mutual authentication. If the authentication is successful, a session key is shared with the MSN to enable it for secured transmission of its data to the cluster head. The mutual authentication at the network core prevents the intruders from accessing the session keys generated by the cluster heads. As a result, the intruders are no longer capable to inject their malicious data into the network. In our proposed framework, the base stations located at the network edge maintain a shared database of all the authorized cluster heads. Each base station is a conventional computing platform that has abundant of computational and storage resources. To communicate with the cluster heads, each edge device needs to be authorized by the base station. Each edge device broadcasts a registration request to a base station to retrieve the database of nominated cluster heads. If the registration request is authorized, the database of nominated cluster heads is provided to the edge device. In SmartEdge, only the authorized edge devices can communicate with the cluster heads for secured transmission of their data streams toward the cloud data centers. Each edge device establishes a secured connection with the data centers for the upload of data gathered from cluster heads. The end users at different enterprises and organizations can download the data using an asymmetric encryption approach. In Fig. 1, the network model of our proposed framework is shown.

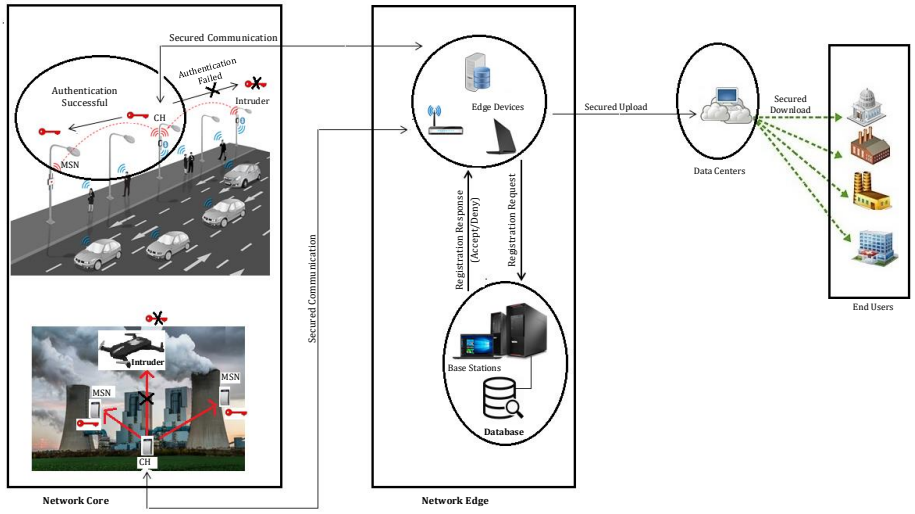


Figure 1: Network Model of SmartEdge

In SmartEdge, we focus on the multimedia data collection. Unlike the conventional computing platforms, the MSNs and cluster heads are relatively low-energy nodes that are unable to execute computationally complex cryptographic algorithms. In view of their resource limitations, we propose an extremely lightweight symmetric encryption for establishing secured connections among them. The edge devices and cloud data centers, on the other hand, have ample resources and are capable to perform resource-intensive operations. As a result, we migrate computationally complex cryptographic tasks to these entities by using an asymmetric encryption approach.

4. SmartEdge: A Lightweight Authentication at the Network Core

At the network core, we use a lightweight mutual authentication approach to establish a secured connection between each MSN and its respective cluster head of Fig. 1. The authentication is performed using four handshake messages. Each message corresponds to an authentication phase. These messages are encrypted using AES-128 bit. The four phases are as follow.

1. MSN Connection Request
2. Cluster Head Challenge

3. MSN Response and Challenge
4. Cluster Head Response

The MSN connection request is preceded by a provisioning phase, a prerequisite offline phase during which the MSNs share a 128-bit secret key (λ_i) with their respective cluster heads. For each application of a smart city, this prerequisite phase is initiated at the time of nodes deployment. Here, the λ_i is known only to the cluster head and the MSN to whom it belongs. Each cluster head maintains a table of all the λ_i for a given application. In this table, the unique identity (α_i) of an MSN is associated with λ_i. This association enables a cluster head to perform the identity verification. Upon successful verification, both the MSN and cluster head communicate with each other for the exchange of a session key. The session key allows an MSN to transmit the gathered data to its respective cluster head. It is assumed that the MSNs are tamper-safe to avoid the compromise of security primitives in accordance with the Internet Threat Model [28]. This model assumes that λ_i are hardcoded at the time of node deployment. In case, if an attacker attempts to tamper with the hardware of an MSN, an alarm is raised to notify about any security breach. The provisioning phase prevents the intruders from registering their fabricated λ_i inside the table of any cluster head.

During the connection request phase, each MSN broadcasts a join-request message (J_{req}) to its respective cluster head. The payload of J_{req} contains α_i, where $i \in \{1, 2, 3, ..., N\}$, and its header contains the identity of a potential cluster head (ID_{CH_n}), where $n \ll N$. Each α_i and ID_{CH_n} are 16 bits long. An intruder may eavesdrop on J_{req} and retrieve λ_i from it. However, it is unable to generate a legitimate session key for the MSN as it was barred from communication with the cluster head during the provisioning phase.

Upon receiving J_{req}, each cluster head retrieves α_i and ID_{CH_n} from it. If ID_{CH_n} matches with the identity of a receiving cluster head and α_i matches with an entry within the cluster head's table, a search for a matching λ_i is made. If a match is found, it means that the J_{req} was received from a legitimate MSN and the cluster head responds with an encrypted challenge ($CH_{challenge}$). This challenge is created by appending a 128-bit session key (μ) with a 128-bit nonce (η_{CH}) and XOR ($\oplus$) with

λ_i. The resultant is encrypted with λ_i and transmitted to α_i as a 256-bit challenge, as shown in Eq. 1.

$$CH_{challenge} = AES\{\lambda_i, (\lambda_i \oplus \mu \mid \eta_{CH})\}. \tag{1}$$

Here, η_{CH} is a temporary random number that is used only once by a given cluster head, and $\oplus$ is an extremely lightweight operation found as a component in various complex cipher suites because it does not leak any information about the original plaintext. The use of η_{CH} in $CH_{challenge}$ makes the latter highly unpredictable.

During the third phase, α_i needs to successfully decrypt $CH_{challenge}$ in order to retrieve μ and η_{CH}. Only a legitimate α_i that possesses the required λ_i can decrypt $CH_{challenge}$. Upon decryption, α_i will have access to μ, that is required for data exchange with its respective cluster head. Upon successful decryption, α_i has authenticated itself. However, for data exchange, the cluster head also needs to authenticate itself. In response to $CH_{challenge}$, α_i creates its own challenge ($\alpha_{challenge}$) by appending λ_i with its own generated nonce η_i and $\oplus$ with η_{CH}. The resultant cipher is encrypted with μ to generate a 256-bit $\alpha_{challenge}$, as shown in Eq. 2.

$$\alpha_{challenge} = AES\{\mu, (\eta_{CH} \oplus \lambda_i \mid \eta_i)\}. \tag{2}$$

In our authentication approach, the η_{client} and η_{server} are generated using a pseudo-random number (R_i). The R_i is appended to a timer (T_i) to make sure that the intruders find it extremely difficult to replay their malicious messages. The T_i assures that η_{client} and η_{server} are non-reproducible, whereas, the R_i assures that η_{client} and η_{server} are non-predictable. The non-predictable nature of R_i and the non-reproducible nature of T_i make it extremely difficult for the intruders to replay maliciously injected data streams.

During the final phase, a cluster head receives $\alpha_{challenge}$ and deciphers it to observe the presence of η_{CH}. If η_{CH} is present, the cluster head retrieves η_i and creates a 256-bit encrypted response of its own, using Eq. 3. This encrypted response is transmitted to α_i and at this stage, the status of α_i at the given cluster head changes to *Authenticated*.

$$CH_{response} = AES\{\lambda_i, (\eta_i \mid \mu)\}. \tag{3}$$

Upon reception of $CH_{response}$, α_i verifies the presence of η_i in $CH_{response}$. If present, it means that the cluster head had authenticated itself as well, and its status at the MSN changes to *Authenticated*. Upon successful verification, the mutual authentication process is completed, and the MSN is now permitted to forward its data to the cluster head. The detailed operational mechanism of our payload-based mutual authentication approach is highlighted in Algorithm 1.

Algorithm 1 Authentication at the Network Core

Initialization:
- $\alpha \longleftarrow [i, \lambda_i]$, where $i \in \{1, 2, 3, ..., N\}$
- *CH* $\longleftarrow [n, \lambda_i]$, where $n \ll N$
- Input [$\lambda_i = \eta_i = \eta_{CH}$= 2^{128}]

$\alpha_i \rightarrow CH_n$: {J_{req}, containing ID_{CH_n} and α_i}
if ID_{CH_n} and α_i matches **then**
    $CH_n \rightarrow \alpha_i$: {$CH_{challenge}$, containing μ and η_r}
else
    Discard J_{req} ▷ MSN is unauthorized.
end if
if λ_i matches **then**
    $\alpha_i \rightarrow CH_n$: {$\alpha_{challenge}$, containing η_i and η_{CH}}
else
    Discard $CH_{challenge}$.
end if
if η_r exists **then**
    α_i is authenticated
    $CH_n \rightarrow \alpha_i$: {$CH_{response}$, containing η_i}
else
    Discard $\alpha_{challenge}$. ▷ MSN is unauthentic.
end if
if η_i exists **then**
    CH_n is authenticated
    $\alpha_i \rightarrow CH_n$: {$D_{\alpha_i \rightarrow CH}$, data exchanges}
else
    Discard $CH_{response}$. ▷ Cluster Head is unauthentic
end if

5. Smart Edge: Registration and Verification at the Network Edge

Upon successful authentication at the network core, each MSN transmits its gathered data streams towards its respective cluster head. For example in Fig. 1, the MSNs at the roadside of smart traffic monitoring application gather the happening events, i.e., data streams, and transmit to their cluster head. The devices at the network edge collect these streams and broadcast to the cloud data centers. Prior to the broadcast, the edge devices need to be registered with and verified by the base stations, co-located at the network edge. Once registered, an edge device attains access to the database of verified cluster heads and communicate with them for secured data transmission. The registration process is completed in four simple phases.

1. Registration Request
2. Encrypted Challenge
3. Encrypted Response
4. Registration Response

During the first phase, each edge device (β_k) transmits a registration request message (R_k) to the base station. The β_k generates a 256-bit session key (μ_k) and XOR with a 256-bit η_k. The 256-bit resultant is encrypted with k to generate a 256-bit R_k, as shown in Eq. 4.

$$R_k = AES\{k, (\eta_k \oplus \mu_k)\}. \tag{4}$$

Here, k is a 256-bit identity of a requesting β_k, η_k is a 256-bit nonce, and μ_k is a potential session key of 256-bit, all generated by β_k. In our scheme, there are only three edge devices, i.e., $k \in \{k_1, k_2, k_3\}$, as we have a limited number of smart city applications. However, it can be extended up to K edge devices for a very large-scale multi-application smart city deployment. We have used AES-256 bit at the network edge because the edge devices and the base stations have ample resources and are capable to process computationally complex cipher suites. Similar to the network core, the AES used at the network edge relies on block cipher mode of operation to provide authenticity and confidentiality of messages.

Once R_k is received by the base station, it decrypts and extracts all the embedded values, i.e., k, η_k and μ_k. At this point, the base station uses the extracted values to generate an encrypted challenge ($BS_{challenge}$). This challenge is composed of a registration certificate (Ω_k), a time-stamp (δ_k) and an authentication value (Ψ_k). To generate Ω_k, the base station performs XOR operation on η_k and μ_k, both received from β_k. Next, an OR operation ($||$) is performed using the resultant ($\eta_k \oplus \mu_k$) and the identity of β_k, i.e., k. Finally, the end product is operated by an AND (.) operator using η_{b_1} to generate a 256-bit Ω_k as shown in Eq. 5.

$$\Omega_k = \eta_{b_1}.(k \mid\mid (\eta_k \oplus \mu_k)). \tag{5}$$

Here, η_{b_1} is a 256-bit nonce generated by the base station. If R_k was received from a legitimate edge device, the latter will be able to decrypt Ω_k using its identity k. The base station generates two copies of Ω_k, one copy is kept for the future communication and the second copy is used in $BS_{challenge}$.

Next, the base station generates a time-stamp (δ_k) using k and a 256-bit η_{b_2}. Both these primitives are operated by an OR operation to generate a 256-bit δ_k as shown in Eq. 6. δ_k represents the total amount of time required by β_k to collect the data from one or more cluster heads.

$$\delta_k = (k \mid\mid \eta_{b_2}). \tag{6}$$

Finally, the base station generates an authentication value (Ψ_k) for the β_k. The Ψ_k is a 256-bit security primitive that is generated by applying XOR operation on k, η_k and η_{b_2}, as shown in Eq. 7. The Ψ_k enables the β_k to generate a response for $BS_{challenge}$.

$$\Psi_k = (k \mid\mid \eta_k \mid\mid \eta_{b_2}). \tag{7}$$

Once Ω_k, δ_k and Ψ_k are generated by the base station, they are transmitted to β_k as a challenge using Eq. 8. At this point, the base station uses the identity k of β_k to encrypt the three security primitives. The encrypted security primitives are embedded in a single message, i.e., $BS_{challenge}$, that has an accumulative size of 768 bits and is

transmitted to β_k.

$$BS_{challenge} = AES\{k, (\Omega_k, \delta_k, \Psi_k)\}. \tag{8}$$

Upon reception of $BS_{challenge}$, β_k checks for the presence of k and μ_k in it. As k and μ_k were generated by β_k, it means that the challenge was received from a legitimate base station. For the verification of β_k and its registration with the base station, β_k needs to successfully decrypt the challenge using its identity (k) to extract Ω_k, δ_k, and Ψ_k. β_k checks for the presence of μ_k and k in the challenge. If these primitives are present, it means that the challenge was received from a legitimate base station and the latter has access to the session key (μ_k) of β_k. The β_k stores Ω_k and δ_k, and uses Ψ_k to generate an encrypted response ($\beta_{response}$) for the base station, as shown in Eq. 9. In this equation, the resultant of 256-bit security primitives, i.e., Ψ_k, η_k, and η_{b_2}, is encrypted with μ_k. The R_k generated by β_k in Eq. 5 enables the base station to have access to μ_k that can be used to decrypt $\beta_{response}$.

$$\beta_{response} = AES\{\mu_k, (\Psi_k \mid\mid (\eta_k \oplus \eta_{b_2})\}. \tag{9}$$

Upon reception of $\beta_{response}$, the base station uses μ_k to decrypt it. The presence of Ψ_k means that the response was received from a legitimate β_k and the latter is registered by the base station. After the decryption of $\beta_{response}$, the base station performs two operations, a) shares Ω_k and δ_k of the verified β_k with all the cluster heads, and b) forwards the database of cluster heads to the verified β_k. At this stage, the base station generates a registration response ($R_{response}$) for β_k that contains the database having the identities of all the legitimate cluster heads, as shown in Eq. 10.

$$R_{response} = AES\{ID_{CH_n}\}. \tag{10}$$

Here, AES is used only to encrypt the identities (ID_{CH_n}) of legitimate cluster heads by appending them in the $R_{response}$ message. The AES neither compress nor affect the size of $R_{response}$. In Algorithm 2, the registration and verification procedure for an edge device is shown.

Algorithm 2 Registration and Verification at the Network Edge

1: **Initialization:**

- $\beta \longleftarrow k$, where $k \in \{1, 2, 3, ..., K\}$
- $\beta \longleftarrow [\mu_k, \eta_k]$
- $BS \longleftarrow [\eta_{b_1}, \eta_{b_2}, ID_{CH_n}]$

▷ BS represents the base station

- Input [$k = 2^{256}, \mu_k = 2^{256}, \eta_k = \eta_{b_1} = \eta_{b_2}$= 2^{256}]

2: $\beta_k \rightarrow BS$: {R_k, Registration Request}
3: $BS \rightarrow \beta_k$: {$BS_{challenge}$, Encrypted challenge of BS}
4: $\beta_k \rightarrow BS$: {$\beta_{response}$, Encrypted response of β_k}
5: **if** Ψ_k exists **then**
6: β_k is authentic ▷ Edge device verified
7: $BS \rightarrow CH_n$: {Ω_k and δ_k, share them with CH_n}
8: $BS \rightarrow \beta_k$: {ID$_{CH_n}$, share database with β_k}
9: **else**
10: β_k is unauthentic
11: **end if**

6. SmartEdge: Secured Data Storage and Sharing

Upon data collection from the cluster heads, each β_k encrypts and encodes the multimedia streams to generate HEVC-Encoded Video Streams (HEVS). For this purpose, a High Efficiency Video Coding (HEVC) standard is used at the network edge [17]. For encryption of HEVS, β_k executes an AES-256 bit algorithm, while at the cloud data centers, RSA-2048 bit algorithm is used for decryption. Using RSA, a pair of private key (k_{pri}) and public key (k_{pub}) is generated at the cloud data center. The latter is transmitted to β_k for video encryption. The AES generates a 256-bit secret key (S_k) for data encryption. The secured data sharing and storage between the network edge and the cloud data center is accomplished in three steps, as shown in Fig. 2. In this section, we discuss these three steps.

1. Encoding and Encryption
2. Partial Decryption
3. Full Decryption

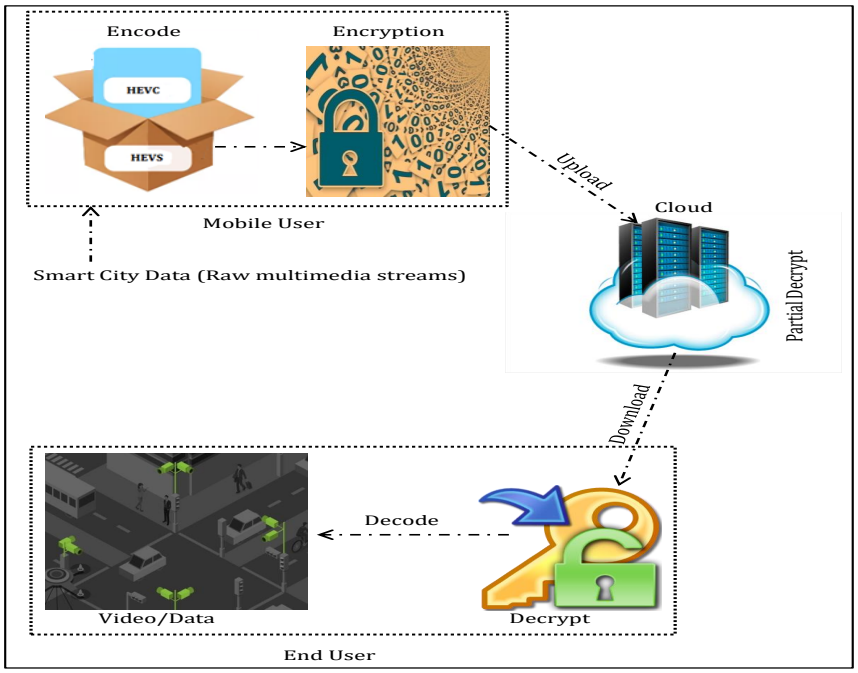


Figure 2: Secured Data Storage and Sharing

6.1. Encoding and Encryption

In this step, each β_k receives the multimedia data streams (D_{stream}) from the cluster heads and encrypts them using its S_k as shown in Eq. 11. The encrypted streams ($D_{encrypt}$) are then embedded into the encoded videos (f(1)....f(N)). Finally, these videos are encrypted using k_{pub} to generate HEVS.

$$D_{encrypt} = AES\{S_k, D_{stream}\}. \tag{11}$$

The data streams generated by MSNs contain N frames. It is impractical to process and encode an entire video sequence at a time [17]. Furthermore, abundant of hardware and computing resources are required for executing HEVC standard and high definition (HD) videos at the network edge. These limitations restrict the encoding of an entire video sequence of N frames at the network edge. A simple solution to this problem is to reduce the number of frames to n for encoding, where $n < N$. Moreover, it is not mandatory to encrypt an entire HEVS because that incurs extra computational burden

and time complexity. The encryption of an entire HEVS affects its format as well. To reduce the computational cost, time complexity, and to preserve the video format, a part of HEVS is nominated for encryption to enhance the security level. To intact the contents of a video, i.e., its format, spatial information (SI), motion vectors (MVs) and intra prediction mode information (IPMI) are used [29]. Once the multimedia streams are encoded and encrypted, β_k uploads these videos, i.e., HEVS, to the cloud data centers.

6.2. Partial Decryption

At the cloud end, the application system provides three major services, i.e., a key generator (KG), a secured storage space (SSS), and a partial decryption function (PDF). During the second step, the KG enables β_k to upload the encrypted HEVS which are stored in SSS. However, prior to this storage, HEVS are decrypted by PDF. To run such an application system over the cloud, we make certain assumptions.

1. Public clouds allow the users to upload/download their data freely with a trust factor. The cloud owners can only view the uploaded contents but are unable to alter them. A robust encryption scheme needs to ensure the security of data in these environments.
2. Each β_k possesses the same S_k.
3. A secured protocol, i.e., Secure Shell (SSH) [30], is used to protect data transmission over wireless channels between β_k and the cloud data centers.

These assumptions are crucial and need to be strictly followed to prevent an adversary from uploading malicious videos/data to the clouds. If the encryption scheme is not robust, an adversary may capture the in-transit video streams, maliciously manipulate them, and replay to the cloud data centers. A weak encryption scheme may allow an adversary to upload an infected or fabricated video in order to control the application system that runs over the public clouds. The presence of a single S_k synchronizes β_k with each other. In case of multiple S_k, the encryption and decryption schemes become complicated. Wireless channels are vulnerable to various threats and as such, secured protocols need to be in place to validate the authenticity of exchanged data.

Using RSA, the KG generates a pair of k_{pri} and k_{pub}. However, it is unable to determine S_k which is associated with a given β_k. Using k_{pub} and S_k, a β_k encrypts HEVS videos and upload to the cloud data centers. Using the same k_{pub} and its k_{pri}, each data center partially decrypts these videos. In our proposed scheme, partial decryption is the responsibility of PDF. For proper functioning of the application system, β_k needs to register itself with KG by transmitting its identity k. Upon reception of k, KG responds back with k_{pub}, that is used by β_k to encrypt its identity and password. The encrypted cipher is forwarded to the application system for secured login session request. In response, the application system provides session information (S_i) for encrypting a video with k_{pub}.

6.3. Full Decryption

Finally, if β_k wants to download specific data/videos from the cloud, it first needs to send a secured session request (S_{req}). KG has a list of authentic β_k that are authorized to download/upload the data from/to the cloud. Recall that KG had provided k_{pub} to each β_k interested in uploading/downloading a video to/from the cloud. KG compares the list of authentic users' identities (K) against the identity (k) of the requesting β_k, where $k \in \{1, 2, 3, ..., K\}$. If a match is found, KG creates a login session for β_k and forwards S_{req} to SSS for checking the requested data on cloud data centers. If the requested data is found, SSS forwards the decoded data to β_k. Upon reception, it decrypts the data using its S_k. If in case, β_k has stolen k from a legitimate user, it can only decrypt the data, but is unable to extract hidden information. This is because, it does not possesses a valid S_k. The detailed operational mechanism of secured data storage and sharing over the cloud is highlighted in Algorithm 3.

7. Performance Evaluation

For our experiments, we use Raspberry *Pi Zero W* boards[6] with raspbian jessie OS and 512 MB RAM. Our setup is based on five neighboring homes, each one having 5 *Pi* boards. Among them, 4 Pi works as MSN and are deployed inside the home. At the

[6]https://www.raspberrypi.org/products/raspberry-pi-zero-w/

Algorithm 3 Secured Data Storage and Sharing

Initialization:

- $\beta_k \longleftarrow S_k$, where $k \in \{1, 2, 3, ..., k\}$
- $KG \longleftarrow [k_{pub}, k_{pri}]$
- Input [$S_k = 2^{256}, k_{pub} = k_{pri}$= 2^{2048}]

$KG \rightarrow \beta_k : \{k_{pub}$, for upload/download of data$\}$
β_k: $f(1....N) \rightarrow (f(1)....f(N)) : \{f()$, encoded video$\}$
▷ β_k encodes N sequences of a video
β_k: $S_k \rightarrow D_{stream} : \{C_{encrypted}$, encrypted data$\}$
▷ Encrypting the data D to create C
β_k stores C in $f()$
▷ Embedding the encrypted data in encoded video
β_k: $k_{pub} \rightarrow f() : \{$HEVS, a binary file$\}$
$\beta_k \rightarrow KG : \{S_{Req}$, secured session request$\}$
▷ Transmitting an encrypted login request to Cloud
$KG \rightarrow \beta_k : \{S_i$, session information$\}$
$\beta_k \rightarrow KG : \{$HEVS$\}$
$KG : [k_{pub}, k_{pri}] \rightarrow f() : \{HEVS$, Partial deecryption$\}$
$SSS \longleftarrow KG :\{$HEVS$\}$
▷ KG forwards the encoded video to cloud's SSS
$\beta_k \rightarrow KG : \{S_{Req}$, secured session request for HEVS$\}$
KG retrieves k from S_{Req} ▷ k is the identity of β_k
if k matches **then**
  $KG \rightarrow \beta_k : \{S_i$, session information$\}$
  SSS checks for the requested data, i.e., HEVS.
  if S_{Req} == True **then**
    $KG \rightarrow \beta_k : \{$HEVS, transmit to β_k $\}$
    β_k retrieves HEVS using S_k
    ▷ Video data successfully downloaded
  else
    $KG \rightarrow \beta_k : \{$HEVS Unavailable $\}$
  end if
else
  β_k unauthorized
end if

main gate in each home, a single Pi operates as a cluster head. Inside the home, each *Pi* is connected to a Spy camera that is capable to support 2592×1944 pixel static images, 720p60, 1080p30, and 640×480p60/90 videos. We use motionEyeOS, a Linux distribution to monitor the video signals from these cameras. In the SmartEdge framework, we used cipher block chaining (CBC) for message authentication (integrity) and SHA-1 for HMAC operation [31]. An edge device, i.e., a core i5 laptop with 64-bit OS, a 2.5GHz processor and 4GB RAM collects data from cluster heads and uploads to Azure Virtual Machines (VMs) located in a public cloud. We analyze the performance of our scheme by comparing it against Lithe [18] and MicroCoAP [15]. We evaluate its efficiency in term of resilience against attacks, security overhead, computation and communication costs, end-to-end encryption delay, and response time.

In Table 1, we analyze the resilience of SmartEdge against various malicious threats. In SmartEdge, η_i and η_{CH} were generated by a random number ($rand_i$) and appended to a timer (T_i). This combination of T_i and $rand_i$ assures that an adversary finds it extremely difficult to launch a replay attack. An adversary may eavesdrop on ciphers, i.e., $\lambda_i \oplus \mu \mid \eta_{CH}$, $\eta_{CH} \oplus \lambda_i \mid \eta_i$, and $\eta_i \mid \mu$, but is unable to decrypt them in absence of λ_i, η_{CH}, and η_i, respectively. The absence of these secret primitives prevents an adversary from launching masquerade attacks. For an adversary to launch a Sybil attack, it needs to steal λ_i from multiple MSNs. However, the MSNs are tamper-safe in view of Internet Threat model [28]. Moreover, the base station maintains α_i of MSNs and ID_{CH_n} of cluster heads that restricts an adversary from fabricating multiple identities. In SmartEdge, each cluster head allows up to four J_{req} attempts. This limitation restricts a node, be it a legitimate one or a malicious one, from launching DoS attacks. The security mechanism deployed by edge devices and VMs is even more resilient to various attacks in presence of RSA algorithm and AES-256. The presence of k_{pub} and k_{pri} at VMs requires 2^{2048} attempts from an adversary to decrypt these keys. The same adversary would require 2^{256} attempts to decrypt S_k. In comparison, the absence of a centralized base station exposes Lithe [18] and MicroCoAP [15] to Sybil attacks. Besides, both these approaches have no defined policy for the prevention of masquerade attacks.

In SmartEdge, we implemented the authentication and key establishment phase

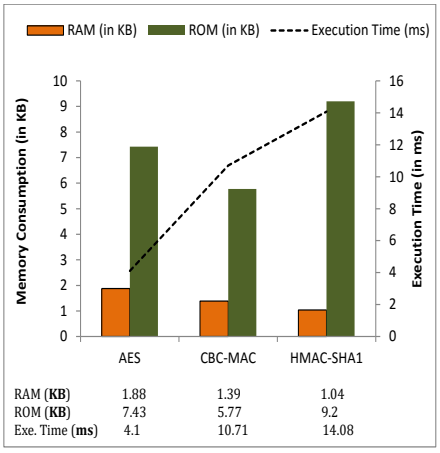
Table 1: Resilience against various Attacks

Threats	MicroCoAP	Lithe	Proposed
Replay	✓	✓	✓
DoS	✓	✓	✓
Masquerade	✗	✗	✓
Sybil	✗	✗	✓

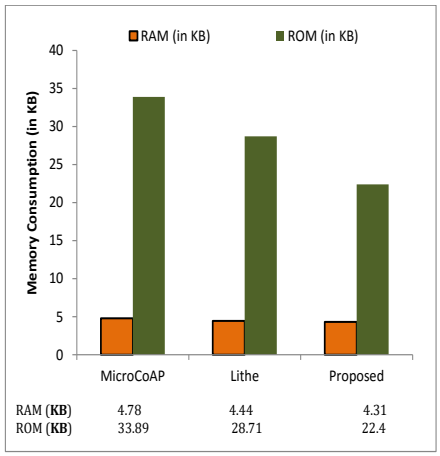
while considering the messages of the following sizes, IDs as 2 bytes, MAC as 4 bytes, nonces as 16 bytes, key size as 16 bytes, and HMAC as 16 bytes, respectively. The MSNs and cluster heads are battery-powered smart devices. For this reason, we determine the price of security overhead, i.e., memory consumption and execution time, for these deployed nodes. In Fig. 3(a), the memory consumption, and the execution time required by each operation, i.e., AES, CBC, and HMAC, on a smart device, are shown. These operations occupy a significantly lower RAM and ROM, allowing these smart devices to perform vital smart home services. Moreover, the symmetric encryption, lightweight MAC and SHA-1 of our scheme have a significantly lower execution time (T_E) on a given smart device. In Fig. 3(b), we compare our scheme against Lithe and MicroCoAP in term of memory consumption. Both these schemes use complex cipher suites and DTLS handshake messages that consume a considerable amount of RAM and ROM.

To determine the lightweightness of our scheme, we evaluate the energy consumption of the cryptographic operations performed by MSNs and cluster heads. The computation and communication costs incurred by these operations in term of energy are shown in Table 2. The computational energy consumed by these cryptographic operations can be calculated using the formula $V \times I \times T_E$ [32]. The value of the voltage (V) is $5V$ and current (I) is $160\ mA$, derived from *Pi Zero W* datasheet[7] in active mode. Using Fig. 3(a), the value of T_E is $28.89\ m$s. Upon evaluation, $V \times I \times T_E$ yields a computational cost of $23.11\ m$J. The communication cost is the amounts of energy spent by an MSN or cluster head to exchange, i.e., transmit or receive, a cipher

[7]https://www.electronicsdatasheets.com/manufacturers/raspberry-pi/parts/raspberry-pi-zero-w



(a) Memory Consumption and Execution Time



(b) Comparison with Existing Schemes

Figure 3: Security Overhead

message. During the authentication operation, four cryptographic messages, i.e., J_{req}, $CH_{challenge}$, $\alpha_{challenge}$ and $CH_{response}$, are exchanged between an MSN and a cluster head. Among them, J_{req} is 32 bits and $CH_{challenge}$, $\alpha_{challenge}$, and $CH_{response}$ are 256 bits each, respectively. According to [33], the transmission and reception of a single bit require 0.72 μJ and 0.81 μJ, respectively. Hence, the overall communication cost incurred by each MSN of our scheme for exchanging these cipher messages is 0.622 mJ, only. In comparison, Lithe and MicroCoAP use large-sized record headers and client certificate requests. These primitives incur an excessive burden in term of computation and communication on a given node. In case of Lithe and MicroCoAP, a node requires 14.3% and 41.8%, more computational energy, respectively. These schemes require 75.2% and 121% more energy in term of communication, in comparison to our scheme.

Table 2: Computation and Communication Cost

Energy (in mJ)	MicroCoAP	Lithe	Proposed
Computation	32.76	26.41	23.11
Communication	1.38	1.09	0.622

In Table 2, we derive the communication cost incurred by an MSN, only. In case of a cluster head, the communication cost is even lower. Based on the mode of operation (transmit/receive) and node type, we derive the communication cost in Table 3. Similar to Table 2, Table 3 is derived based on the operational mode and size of each cipher message.

Table 3: Communication Cost: MSN (α)$\longleftrightarrow$ CH

Node Type	J_{req}	$CH_{challenge}$	$\alpha_{challenge}$	$CH_{response}$	Cost (mJ)
MSN	Trans	Recv	Trans	Recv	0.622
CH	Recv	Trans	Recv	Trans	0.601

To establish secured sessions among various entities of a smart home, we calculated the end-to-end encryption delay in Fig. 4. In SmartEdge, encryption is achieved at three levels, i.e., MSN to cluster head, cluster head to the edge, and edge to VMs. The average session establishment time between the smart devices, i.e., MSN and its respective cluster head, range from 24 to 33 ms. The average session time for cluster head to edge ranges from 40 to 65 ms, and for edge to VMs ranges from 81 to 145 ms, respectively. Based on these calculations, the end-to-end encryption delay for a complete session ranges from 243 to 699 ms. In Fig. 4, cluster heads to clouds/VMs mean that there is no edge device involved, which is not the case with SmartEdge. In absence of an edge device, it requires 554 to 775 ms for establishing a session between a cluster head (a smart device) and VMs. In absence of network edge, a smart device communicates directly with VMs and experiences a significantly higher delay for establishing a secured session.

To analyze the round trip time (RTT), we configure the general-purpose input/output (GPIO) pins of MSNs and cluster heads to operate with enabled radio duty cycling (RDC) and disabled RDC. The radio remains off most of the time and is turned on at certain intervals to check the medium for any incoming or outgoing packet. In Fig. 5(a), we evaluate SmartEdge with enabled RDC. In this figure, the data packets have varying payload sizes (in bytes). SmartEdge has a smaller RTT in comparison to the existing schemes. RTT is measured as the length of time it takes for a request to be

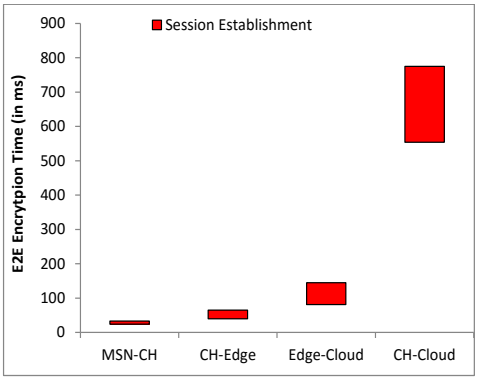
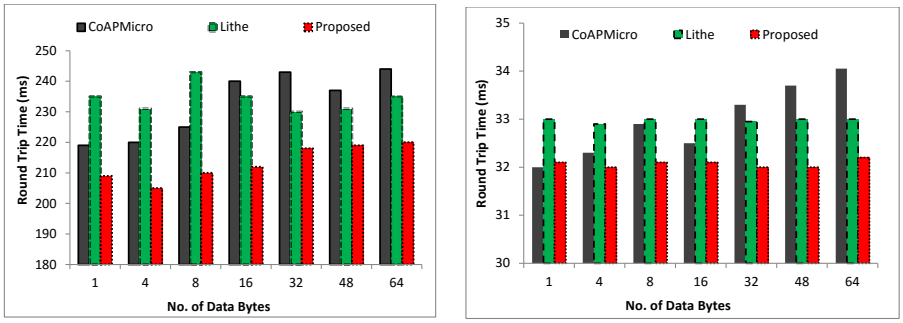


Figure 4: End-to-End Encryption

sent and its response be received [15]. In Fig. 5(b), the RTT is calculated for a disabled RDC. In comparison to Fig. 5(a), Fig. 5(b) has a relatively lower RTT for all the three schemes. In RDC, energy consumption of a node is reduced by keeping the radio in sleep mode for most of the time. However, RDC achieves energy conservation at the expense of higher latency because a transmitter node needs to wait until a receiver node wakes up. It may happen that the waiting interval of a transmitter overlaps with the sleeping interval of a receiver. Therefore, the overall RTT is much higher for data packets with enabled RDC. Irrespective of RDC-enabled or RDC-disabled, our scheme has significantly lower RTT values in comparison to Lithe and MicroCoAP. In Fig. 5(a) and Fig. 5(b), the response time is calculated as $\frac{1}{2}\times$ RTT.



(a) With RDC (b) Without RDC

Figure 5: Round Trip Time

8. Conclusion

In this paper, we propose a SmartEdge framework to secure the transmission of multimedia data in smart homes. SmartEdge offloads computationally complex and resource-intensive operations to the network edge and cloud data centers. Using a lightweight symmetric encryption, each multimedia sensor node (MSN) establishes a secured connection with its respective cluster head for the transmission of its raw data streams. At the network edge, one or more verified edge devices collect these streams from cluster heads, manipulate them, and transmit to the cloud. Prior to the data transmission, each edge device encrypts the streams, encodes them and establishes a secured connection with the cloud data centers. Each edge device executes a relatively complex symmetric encryption while the cloud data centers execute resource-intensive asymmetric encryption. Upon establishing a secured session, the multimedia data is uploaded to the cloud that can be downloaded and decrypted by the end users for provisioning of various services in a smart home environment. In comparison with the existing schemes, security overhead is reduced by up to 10% in case of RAM consumption and 51% in case of ROM consumption. In term of energy consumption, the computational cost is reduced by up to 42% and communication cost is reduced by up to 21%. In term of end-to-end encryption, our scheme has a significantly lower session establishment delay. It is highly resilient against various adversarial attacks and has a lower response time for a payload of various sizes. In future, we aim to configure our scheme by incorporating mobility of smart devices and scalability of the network edge.

References

[1] Y. Qin, Q. Z. Sheng, N. J. Falkner, S. Dustdar, H. Wang, A. V. Vasilakos, When things matter: A survey on data-centric internet of things, Journal of Network and Computer Applications 64 (2016) 137–153.

[2] J. Jin, J. Gubbi, S. Marusic, M. Palaniswami, An information framework for creating a smart city through internet of things, IEEE Internet of Things Journal 1 (2014) 112–121.

[3] A. Gharaibeh, M. A. Salahuddin, S. J. Hussini, A. Khreishah, I. Khalil, M. Guizani, A. Al-Fuqaha, Smart cities: A survey on data management, security, and enabling technologies, IEEE Communications Surveys & Tutorials 19 (2017) 2456–2501.

[4] H. Cai, B. Xu, L. Jiang, A. V. Vasilakos, Iot-based big data storage systems in cloud computing: perspectives and challenges, IEEE Internet of Things Journal 4 (2017) 75–87.

[5] M. Díaz, C. Martín, B. Rubio, State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing, Journal of Network and Computer Applications 67 (2016) 99–117.

[6] N. Mohamed, J. Al-Jaroodi, I. Jawhar, S. Lazarova-Molnar, S. Mahmoud, Smartcityware: A service-oriented middleware for cloud and fog enabled smart city services, Ieee Access 5 (2017) 17576–17588.

[7] M. Dalla Cia, F. Mason, D. Peron, F. Chiariotti, M. Polese, T. Mahmoodi, M. Zorzi, A. Zanella, Using smart city data in 5g self-organizing networks, IEEE Internet of Things Journal 5 (2018) 645–654.

[8] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, X. Yang, A survey on the edge computing for the internet of things, IEEE Access 6 (2018) 6900–6919.

[9] S. Li, N. Zhang, S. Lin, L. Kong, A. Katangur, M. K. Khan, M. Ni, G. Zhu, Joint admission control and resource allocation in edge computing for internet of things, IEEE Network 32 (2018) 72–79.

[10] T. Wang, G. Zhang, A. Liu, M. Z. A. Bhuiyan, Q. Jin, A secure iot service architecture with an efficient balance dynamics based on cloud and edge computing, IEEE Internet of Things Journal (2018).

[11] X. Lyu, H. Tian, L. Jiang, A. Vinel, S. Maharjan, S. Gjessing, Y. Zhang, Selective offloading in mobile edge computing for the green internet of things, IEEE Network 32 (2018) 54–60.

[12] P. Hu, S. Dhelim, H. Ning, T. Qiu, Survey on fog computing: architecture, key technologies, applications and open issues, Journal of Network and Computer Applications (2017).

[13] X. Huang, N. Ansari, Secure multi-party data communications in cloud augmented iot environment, in: Communications (ICC), 2017 IEEE International Conference on, IEEE, 2017, pp. 1–6.

[14] J. Shen, D. Liu, J. Shen, Q. Liu, X. Sun, A secure cloud-assisted urban data sharing framework for ubiquitous-cities, Pervasive and mobile Computing 41 (2017) 219–230.

[15] M. A. Jan, F. Khan, M. Alam, M. Usman, A payload-based mutual authentication scheme for internet of things, Future Generation Computer Systems (2017).

[16] J. Daemen, V. Rijmen, The design of Rijndael: AES-the advanced encryption standard, Springer Science & Business Media, 2013.

[17] M. Usman, M. A. Jan, X. He, Cryptography-based secure data storage and sharing using hevc and public clouds, Information Sciences 387 (2017) 90–102.

[18] S. Raza, H. Shafagh, K. Hewage, R. Hummen, T. Voigt, Lithe: Lightweight secure coap for the internet of things, IEEE Sensors Journal 13 (2013) 3711–3720.

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

[20] H. Tao, M. Z. A. Bhuiyan, A. N. Abdalla, M. M. Hassan, J. M. Zain, T. Hayajneh, Secured data collection with hardware-based ciphers for iot-based healthcare, IEEE Internet of Things Journal (2018).

[21] G. Orsini, D. Bade, W. Lamersdorf, Cloudaware: A context-adaptive middleware for mobile edge and cloud computing applications, in: Foundations and

Applications of Self* Systems, IEEE International Workshops on, IEEE, 2016, pp. 216–221.

[22] K. Habak, M. Ammar, K. A. Harras, E. Zegura, Femto clouds: Leveraging mobile devices to provide cloud service at the edge, in: Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on, IEEE, 2015, pp. 9–16.

[23] X. Sun, N. Ansari, Edgeiot: Mobile edge computing for the internet of things, IEEE Communications Magazine 54 (2016) 22–29.

[24] P. Liu, D. Willis, S. Banerjee, Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge, in: Edge Computing (SEC), IEEE/ACM Symposium on, IEEE, 2016, pp. 1–13.

[25] J. Pan, L. Ma, R. Ravindran, P. TalebiFard, Homecloud: An edge cloud framework and testbed for new application delivery, in: Telecommunications (ICT), 2016 23rd International Conference on, IEEE, 2016, pp. 1–6.

[26] J. Son, D. Kim, M. Z. A. Bhuiyan, R. Hussain, H. Oh, A new outsourcing conditional proxy re-encryption suitable for mobile cloud environment, Concurrency and Computation: Practice and Experience 29 (2017) e3946.

[27] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, K. Ren, Heavy hitter estimation over set-valued data with local differential privacy, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 192–203.

[28] E. Rescorla, B. Korver, Guidelines for writing RFC text on security considerations, Technical Report, 2003.

[29] G. J. Sullivan, J. Ohm, W.-J. Han, T. Wiegand, Overview of the high efficiency video coding (hevc) standard, IEEE Transactions on circuits and systems for video technology 22 (2012) 1649–1668.

[30] T. Ylonen, C. Lonvick, The secure shell (ssh) protocol architecture (2006).

[31] D. Eastlake 3rd, P. Jones, US secure hash algorithm 1 (SHA1), Technical Report, 2001.

[32] D. He, S. Chan, S. Tang, M. Guizani, Secure data discovery and dissemination based on hash tree for wireless sensor networks, IEEE transactions on wireless communications 12 (2013) 4638–4646.

[33] G. De Meulenaer, F. Gosset, F.-X. Standaert, O. Pereira, On the energy cost of communication and cryptography in wireless sensor networks, in: Networking and Communications, 2008. WIMOB'08. IEEE International Conference on Wireless and Mobile Computing,, IEEE, 2008, pp. 580–585.