# ORIOT: A Source Location Privacy System for Resource Constrained IoT Devices

Clemens Lachner, Thomas Rausch, Schahram Dustdar
Distributed Systems Group
TU Wien, Vienna, Austria
{c.lachner, t.rausch, dustdar}@dsg.tuwien.ac.at

***Abstract*—Privacy and Security are one of the major research topics regarding the Internet of Things (IoT). Due to the vast amount of devices collecting and processing sensitive data, anonymity and privacy mechanism are needed. Source Location Privacy (SLP) plays a key role in prohibiting adversaries from tracing back this kind of data to its origin. In this paper we propose a SLP preserving system that leverages techniques from the well established Onion Routing paradigm. The system is specifically designed for resource constrained IoT devices, i.e., devices lacking computing power. It features combined encryption schemes and symmetric key exchanges via Elliptic-Curve Diffie-Hellman (ECDH). Our performance measurements, conducted on typical resource constrained IoT devices, show the feasibility of ORIOT and facilitate the integration into existing or planned IoT systems, depending on SLP features.**

***Index Terms*—onion routing, privacy, SLP, IoT, system**

## I. Introduction

The Internet of Things (IoT) describes a heterogeneous network comprising a variety of different connected devices with minimal to average computing power. These devices continue to permeate deeper in our personal environment as well as in commercial and industrial areas, by sensing, processing, and storing all kind of data [1]. For many applications, like in healthcare, home automation or infrastructure monitoring, these circumstances call for privacy and security protection [2]. Integrity, confidentiality, availability, undetectability, and unobserveability are the key elements of such protection mechanisms. Though features of these elements overlap, according to [3], we place integrity, confidentiality, and availability into the security domain; undetectability and unobserveability into the privacy domain. Regarding privacy, we further distinguish between data-anonymity and source location privacy (SLP). Broadly speaking, (personal) data provided by IoT devices can be used by adversaries to obtain or derive sensitive information that could compromise users. Data-anonymization techniques offer a solution to mitigate such privacy breaches. SLP, as the name implies, aims to keep the location private, where data was originally collected. Referring to IoT, being an interconnected network, this would in most cases result in efforts to keep the IP-address of a device private. A well established approach to achieve this goal is Single Path Routing (SPR). Data packets are routed to their final destination following a specific path inside a network to make it harder for an adversary to trace back their origin. Mix-cascades and onion routing are prominent concepts for SPR, where DC-nets and Tor are its most popular implementations. In this paper we leverage principles of onion routing, where data is encapsulated in multiple layers of encryption, hence the analogy to an onion, and routed along a predefined path to its destination. Such a path consists of various nodes called *onion routers* or *relay nodes*. Each intermediary node removes one encryption layer and thereby only reveals the address of the next node in the route. Therefore, each node only knows the location of its predecessor and successor node. This mechanism facilitates sender anonymity. However, several weaknesses have been found to break this anonymity, like Timing or Traffic Analysis. The design of ORIOT is based on typical IoT systems, e.g., Amazon AWS IoT, where data is generated at various nodes inside a network and sent to the cloud for further processing. Many IoT devices are constrained by their available resources, i.e., in most cases computing power, like microcontrollers, that are not running any operating system. Therefore, well established implementations of Onion Routing, e.g., Tor, are infeasible for such devices. The main contribution of this paper is a SLP system, specifically designed for resource constrained IoT devices, to address this issue. It is implemented in C, and therefore highly compatible and portable to most IoT devices. Furthermore, we provide performance results on different cryptographic mechanisms that are integral parts of our system. Experiments have been conducted on typical resource constrained IoT devices, therefore our results facilitate the design and development of IoT systems that rely on SLP features, e.g., by implementing ORIOT.

The rest of the paper is organized as follows. Section II provides an overview about related work on IoT security and privacy. In Section III, we introduce our proposed implementation. We evaluate our approach and discuss the results in Section IV. Finally, in Section V we conclude the paper and give an outlook on future research.

## II. Related Work

With the increasing spread and usage of IoT devices, security and privacy aspects have become a major research topic in the area of IoT data protection. Suo et al. [17] state that there are four abstraction layers in IoT. Bottom-up these are: Perception Layer, Network Layer, Middleware Layer and Application Layer. In their summarizing paper, Farooq et al. [14] give an overview about possible threats and scenarios on these different layers, where the majority of

threats are located in the Network Layer. Energy consumption and management, as well as efficient computing algorithms (e.g., share of workload among multiple devices) play a key role for resource constrained IoT devices. Therefore, for the majority of use cases concerning privacy and security aspects, trade-offs have to be made either at design time or runtime. An example of such a trade-off could be a stronger encryption scheme resulting in lower data throughput. In this paper we want to tackle those issues and minimize such trade-offs. In the domain of Wireless Sensor Networks (WSN) various solutions to problems dealing with SLP or anonymization have been proposed. Most of the devices in WSN reside at the lowest end regarding computing power and represent a subset of IoT devices. Commonly those devices adhere to the IEEE 802.15.4 protocol, using communication frameworks like ZigBee. Besides being part of a WSN, they are also integrated in smart objects such as smart phones, tablets, smart watches, and many others gadgets [18]. Security and privacy mechanism often require considerable computing power that cannot be provided by such devices. A typical pragmatic solution is the usage of *IoT gateways* that are placed between (sensor) networks and the Internet, powerful enough to facilitate more compute-intensive security and privacy mechanisms [3], [13]. However, there may be situations where such gateways are not desired or possible. A well-known example in literature is the Panda-Hunter Game, where a WSN is deployed in a forest to monitor pandas. Hunters take the role of an adversary, trying to capture the panda. The goal is to prevent the hunter from locating the source, i.e., the sensor attached to a specific panda [21].

Generally speaking, privacy can be either achieved by leveraging data-anonymization techniques, SLP-mechanisms, or a combination of both. Researchers have investigated several anonymization techniques, such as simple pseudonymization, attribute suppression, or more sophisticated approaches like the k-Anonymization model [5]–[7], [19], [20]. However, most of this techniques do not incorporate SLP features, especially not for resource constrained devices. Jebri et al. [8] propose a generic security and privacy model for IoT and WSN that includes SLP. Their work is based on a lightweight key agreement protocol, Identity Based Encryption (IBE), and Pseudonym Based Encryption (PBC). To make use of an IBE system the authors had to incorporate a Private Key Generator (PKG) that acts as a trusted central key authority. PBC is a technique based on IBE, and is generally used to protect the identity of an entity. The architecture comprises a base station, a sink node, and a set of nodes. The PKG is integrated into the base station which stores the identities of the nodes. Before any data is sent over the network, a setup phase takes place in which several encryption and privacy mechanisms, e.g., generation of private and public keys, are configured. In order to transmit information, each node sends its data, protected by calculated session keys, directly to the Sink Node (SN). Due to the use of PBC, the identity of the source node stays anonymous. Another concept which requires less encryption overhead is Anonymous Routing (AR). AR is a well suited concept to achieve SLP. There are many ways to implement, integrate, and extend this concept for applications that operate with sensitive data where the source has to stay anonymous. Palmieri et al. [11] proposed a protocol for AR between different interconnected networks. It is designed specifically for IoT applications and is based on the Spatial Bloom Filter (SBF) data structure. Furthermore, all routing information is encrypted using an additive and multiplicative homomorphic encryption scheme. However, as stated by the authors, this cryptographic system may not be suitable for computationally constrained devices. Another protocol that specifically targets resource-constrained mobile ad hoc networks, was proposed by Moldovan et al [9]. Their group-based anonymous on-demand routing protocol works in a similar way to Tor. After detecting all nodes in network, a secret handshake with all nodes is performed by a dedicated trusted network administrator. Afterwards, for two neighboring nodes a secure common key is computed. Further cryptographic processes ensure resistance against different attacks, e.g., Message Coding Attacks. Specific request and response messages, which are partially broadcasted inside the network, are used to establish a communication path between source and target nodes, comprising pairs of securely linked nodes. Each node is known to others under a pseudonym, which is used to forward a message along the path, while keeping private both source and target. Referring to SLP in IoT, we assume that the source location relates to an IP-Address of a device. Especially in WSN, SLP problems are closely tied to real geolocation privacy of an entity, but the used techniques are similar and related to IP-Address privacy. To achieve geolocation privacy, Mutalemwa et al. [12] proposed a strategic location-based random routing approach. According to their scheme, data packets are encrypted and routed over the network according to the physical location of a source node. To determine a routing path, intermediary strategically positioned diversion nodes are randomly selected based on their distance to the source node. Successive packets are routed through different routing paths. Simulation results demonstrate that their approach makes it difficult and confusing for an adversary to trace back the origin of such data packets. In IoT, especially when dealing with resource constrained devices, such security and privacy mechanisms require several trade-off decisions to be made, as stated earlier. Techniques that leverage broadcasting mechanisms or rely on heavy network traffic in general will automatically cause a higher energy consumption of all devices. Computational intensive encryption mechanisms on the other hand are infeasible for scenarios where data is sent over the network with high frequency.

In this paper we combine several above mentioned security and privacy techniques and incorporate them into an onion routing system, similar to Tor, targeted for resource-constrained IoT devices, to minimize the above mentioned trade-offs. Compared to other approaches, our system does not rely on heavy cryptographic algorithms to provide anonymity. On the other hand, ORIOT avoids network broadcasting strategies as used by different proposed SLP systems. By setting a specific path length for our message transfer, a well balanced trade-off

between network load and SLP level is achieved.

## III. Onion Routing System

In this section we describe our proposed onion routing system called ORIOT. First we present an overview of the systems architecture. Second we explain the setup phase that is performed by a device when integrated into the system. Third we describe the path assembling strategy that is performed before sending data. Finally we present the encryption and actual routing processes.

### A. Overview

The architecture comprises multiple devices (nodes) which are constrained in their computing power, and a single more powerful device acting as a central form of registry, therefore simply called *Registry*. Figure 1 shows the overall architecture. We proceed to explain each step, as marked by the corresponding number in blue circles.

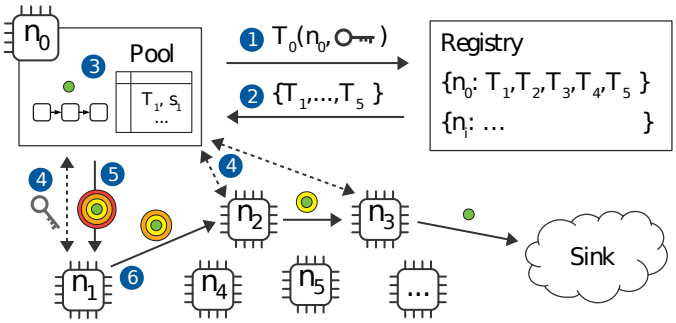


Fig. 1. Overview of the ORIOT architecture

Devices that are part of a network are called nodes (depicted as $n_0, n_1, ...$ in Figure 1), which collect or process data provided by various sources such as sensors or system events. In our experiments we use various microcontroller development boards to act as our typical resource constrained IoT devices. In our experimental setup, all devices are part of a local network and are fully connected to each other. The systems architecture focuses on mechanisms implemented for and operating on the Transportation Layer of the OSI model. The idea is, that if a node wants to send a message to a specific destination (e.g., the cloud), the data is encrypted several times (depicted as layered circles in Figure 1) and sent to the destination via multiple hops (depicted as continuous arrows in Figure 1) across the network. In our setup, each node sends its data to a specific destination, e.g., a cloud server or sink. In WSN terminology this would refer to a *sink node*. Preliminary, nodes are provided with the public key of the cloud server. All messages are initially encrypted in a way so that only the cloud server can read the message (end-to-end encryption). However, our system design is not limited to sending messages to only one particular destination. The first step for a node that wants to send a message, is to ask the Registry for a designated *IP-Pool*. This IP-Pool comprises tuples of IP-Address and corresponding public key of nodes in the network. After receiving the IP-Pool, the node randomly selects a subset of the given tuples, which will represent the path along data is subsequently routed and transferred to its final destination. A more detailed description of this IP-Pooling mechanism will be provided in the path assembling section later on. Similar to Tor, the first node of a routing path acts as *Entry Node* (the only node which is actually able to see the source IP-Address) and the last node acts as *Exit Node* which sends the data to the final destination. A layered encryption strategy ensures that every node along a routing path only knows the IP-Address of its predecessor node and its successor node.

### B. Setup Phase

The setup phase is a specific piece of code that is executed only once when a device is started. It can be divided into two essential steps:

*1) Encryption Setup:* In our setup we incorporate an end-to-end encryption scheme, i.e., only our designated destination cloud server is able to read a message in plaintext. Similar to TLS we use a combination of asymmetric and symmetric encryption. First, a node generates a random secret key which is later on used to encrypt the message. Second, the generated key is encrypted with the public key of the cloud server. With its private key, the cloud server is able to decrypt the shared secret which is then used to decrypt the actual message.
The next step in setup phase is the generation of a public and private key pair. As described later, those are used for our layered encryption scheme.

*2) Network Setup:* After the encryption setup has finished, every node publishes its previously generated public key to the Registry, which saves this information as a tuple of $< PublicKey_{Node}, IP\text{-}Address_{Node} >$ in a list. The Registry then randomly adds those tuples to a specifically sized pool (e.g., 10 tuples stored in one pool), depending on the size of the network, and the available storing capacities of the device hosting the registry. However, a pool must contain at least a minimum of three nodes. To further increase the level of anonymity we recommend using one specific pool for each node in the network, although pools could be reused if storage on the registry node is limited. Pools should be randomly reorganized, based on a configurable *staleness factor*. The actual value of this staleness factor (e.g., 10 minutes) is determined by the expected traffic over the network, i.e., how frequently data will be sent from a node. After pools are created, every node in the network will be assigned one pool. After the encryption and pooling steps are completed, a node starts listening for incoming data. All communication (except for the key exchange described in the path assembling section, which is based on TCP) is ideally based on, but not limited to, UDP because of two major reasons:

- A node only sends data and never expects an answer (except during the key exchange).
- UDP has a noticeable network performance advantage over TCP.

To that end, the node opens a predefined common port and waits for specific instructions.

### C. Path Assembling

A routing path comprises five nodes in total. We refer to the first node as the source node and the last node as the cloud server. Intermediary nodes are called relay nodes. Though it would be possible to add more than three relay nodes to a path, we advise against it, as this increases network load without providing any more security or anonymity [22]. In our example the IP-Address of the cloud server is known to each node, therefore it is not a part of the path assembling scheme. If a node wants to send a message, it opportunistically starts building a transfer route. Opportunistically means, that to this end, a node building a path will not know if another node, that will be selected as part of this path, is actually online or working properly. When assembling a routing path, the node randomly selects exactly three tuples out of its stored IP-Pool. These tuples correspond to our relay nodes and will form the intermediary path to the cloud server. For each relay node, the source node starts a key exchange (depicted by dashed arrows in Figure 1) by sending a *InitiateKeyExchange* request. If an addressable relay node receives this request, the process of generating a common symmetric key is initiated. If no response is received by the source node after a predefined timeout, it removes the corresponding tuple out of its IP-Pool and notifies the Registry about the faulty node. If there are too many faulty nodes (this threshold can be adjusted at design time), a node requests a new pool from the registry. However, if the node receives a response, the common symmetric key generation is established via the Elliptic-Curve Diffie-Hellman (ECDH) key exchange protocol [10]. All calculated symmetric keys are stored for each relay node for the layered encryption process later on. It is up to the developer if and how long previously negotiated symmetric keys are cached on the source node and the relay nodes for later reuse or not. This becomes particularly interesting if a node in the network is replaced, possibly resulting in an IP-Key (either asymmetric or shared) mismatch. The corresponding pseudocode is shown in Algorithm 1, but does not cover any caching mechanism or requests for a new entire pool.

### D. Encryption and Routing

After a routing path has been determined, the message encryption process is initiated. The corresponding pseudocode, denoted in Algorithm 2 covers the encryption layering process. First the message $M$ is encrypted with the key $K$ generated in the setup phase resulting in the message $M_K$.

The encrypted message for the cloud server, i.e., the innermost layer of the onion $L_0$, will be in the form of:

$$L_0 = \{M_K, K_{EC}\} \tag{1}$$

where $K_{EC}$ is $K$ encrypted with the public key of the cloud previously done in the setup phase.

After that the first layer $L_1$ of encryption is added to $L_0$ in Form of:

$$L_1 = Enc_{KN_2}\{L_0, IP_{CS}\} \tag{2}$$

**Input:** pool $[T_0, T_1, T_2, ...T_n]$
**Result:** path[3]$< T, sharedKey >$ of tuples $T_x$ where $x \in [0..n]$, and corresponding shared keys, respectively
**while** *count(path)* $< 3$ **do**
  rT = getRandomTupleFromPool;
  **if** *path !contains(rT)* **then**
    `// perform ECDH key exchange`
    sharedKey = InitiateKeyExchange(rT);
    **if** *sharedKey* **then**
      `// save established key to corresponding tuple`
      keyAdd(path, sharedKey);
    **else**
      `// handle timeout and maxFaults`
    **end**
  **end**
**end**

**Algorithm 1:** Path Assembling

Subsequently, for each remaining node, an additional corresponding encryption layer will be added in form of

$$\begin{aligned} L_2 &= Enc_{KN_1}\{L_1, IP_{N_2}\} \\ L_3 &= Enc_{KN_0}\{L_2, IP_{N_1}\} \end{aligned} \tag{3}$$

where $Enc_{KN_i}$ is an encryption function with the symmetric key previously exchanged with node $N_i$ and $IP_{N_i}$ is the IP-Address of the successor relay node, with $i \in [0, 1, 2]$.

**Input:** msg, path[3], cloudPubKey, rndKey, cloudIP
**Result:** Multiple Encrypted Message (Onion)
`// encrypt message for cloud server`
mk = encryptCloudMessage(msg, rndKey);
kec = encryptKey(rndKey, cloudPubKey);
lyr = composeCloudMessage(mk, kec);
ip = cloudIP;
`// add encryption layers`
**for** *i = count(path) - 1 .. 0* **do**
  key = getKeyFromNodeInPath(path, i);
  **if** *i != 2* **then**
    ip = getIPFromNodeInPath(path, i+1);
  **end**
  addLayer(lyr, ip, key);
**end**

**Algorithm 2:** Layered Encryption

The final multiple encrypted message will be sent from the source node to the first relay node via a specific *ForwardMessageRequest*. If a relay node receives such a request it will then be able to decrypt one layer with its symmetric key and will forward the message to the next relay node, respectively, or in case of the last relay node, to the cloud server.

TABLE I
TESTBED OVERVIEW OF RESOURCE CONSTRAINED IOT DEVICES

| Device Name | Processor | CPU Speed | SRAM | Flash Memory |
|---|---|---|---|---|
| Arduino MKR1000 WiFi | Cortex-M0+ 32-Bit | 48 MHz | 32 KB | 256 KB |
| Wemos ESP8266 D1 mini | Xtensa LX106 32-Bit | 80-160 MHz | 160 KB | 4 MB |
| Espressif ESP32-WROOM-32 | Xtensa LX6 32-Bit DualCore | 160-240 MHz | 512 KB | 4 MB |

TABLE II
PERFORMANCE RESULTS OF CRYPTOGRAPHIC ALGORITHMS ON A SINGLE NODE

| | Key Generation (ECDH_1) | Symmetric Key Calculation (ECDH_3) | AES-256 | ChaCha20-256 |
|---|---|---|---|---|
| MKR-1000 | 0.492s | 0.501s | 68.04kB/s (14.35$\mu$s/B) | 543.94kB/s (1.80$\mu$s/B) |
| ESP8266 | 0.082s | 0.097s | 211.56kB/s (4.62$\mu$s/B) | 3,149.09kB/s (0.31$\mu$s/B) |
| ESP32 | 0.026s | 0.027s | 1,859.84kB/s (0,53$\mu$s/B) | 4,078.84kB/s (0,24$\mu$s/B) |

## IV. PERFORMANCE AND BOUNDARIES

This section presents performance measurements and resulting boundaries of ORIOT. Our testbed comprises three microcontroller development boards with integrated WiFi capabilities, acting as typical resource constrained IoT devices. Table I provides an overview of the selected hardware.

A prototypical implementation was developed in *C* using the Arduino IDE v.1.8.7 running on 64bit Linux Mint 19.1. Code execution on such microcontrollers is divided in a setup phase (where code runs only once), and a loop phase. We are particularly interested in the performance of cryptographic functions, rather than round trip times (RTT) or data transfer (e.g., pools) which are heavily prone to network latency affected by various unstable environmental factors like signal strength or interferences. In the setup phase, we investigate key generation, specifically a 256bit private and public key pair via Ed25519 elliptic-curve cryptography. Symmetric Key exchange (ECDH) and encryption layering is done in the loop phase. The first phase of ECDH is the generation of a public and private key, already done in the setup phase. In the second phase of ECDH the public keys are exchanged between the two parties. The performance of this step relies solely on network traffic, therefore it is not covered by our measurements. In the third phase, a common secret key is derived from calculations that take the secret key of a communication partner and the previously exchanged public key of the other partner as input. Due to the nature of the algorithm, the third phase of ECDH performs similar to the first phase, but for better readability we measure and present it separately.

All symmetric encryption, i.e., creating the onion, is done using the AES block cipher with a 256bit key in CTR mode of operation. Additionally, we measured the layering process using the ChaCha20 stream cipher with a 256bit key, as a lightweight alternative. However, we remark that modern microprocessors, like the ESP32, come with built in hardware acceleration capabilities for AES and ECC.

Table II displays the obtained results. The performance of symmetric ciphers is expressed as *limitation* and *throughput*. Limitation corresponds to the time it takes for an algorithm to process one byte of data, given in $\mu$s per byte ($\mu\mathrm{s}/\mathrm{B}$), while throughput describes how many bytes can be processed in one second ($\mathrm{B}/\mathrm{s}$). With provided throughput and limitation values, boundaries for a specific network can be then calculated individually. For symmetric ciphers, values outside brackets correspond to limitation, while values in brackets correspond to throughput. Each of those values correspond to adding one layer of encryption. To get a close approximation of the time needed to create all layers of encryption, i.e., the whole onion, the results need to be multiplied by the number of layers. Due to the nature of the used symmetric ciphers, time needed for encryption is almost exactly the same as for decryption. Values of asymmetric operations represent the time needed in seconds for the whole operation to finish, be it either key generation or deriving a common symmetric key.

The execution time of the cryptographic algorithms scale linearly with CPU frequency. This becomes particularly interesting if energy consumption is a critical design aspect of an IoT system using ORIOT. Devices like the ESP8266 and ESP32 can easily be underclocked, i.e., running the CPU at a lower frequency, hence consuming less energy.

## V. CONCLUSION

This paper presents a source location privacy preserving network system and its architectural concepts, specifically designed to operate on resource constrained IoT devices. Similar to Tor, it leverages techniques of the onion routing principle. Symmetric and asymmetric cryptography are combined with a path assembling strategy to realize anonymity of a node transmitting messages in a network to a specific destination. Furthermore, we evaluated the performance of incorporated cryptographic algorithms on a set of typical resource constrained IoT devices. Those results can facilitate the design and development of an IoT system implementing ORIOT. However, there are still open challenges we need to address. Future work will include packet padding and noise generation to mitigate attacks like timing/traffic analysis. Furthermore, we want to investigate energy consumption properties of ORIOT and relevant optimizations, if necessary.

## REFERENCES


[1] Vermesan, Ovidiu, and Peter Friess, eds. Internet of things: converging technologies for smart environments and integrated ecosystems. River Publishers, 2013.

[2] Daubert, Joerg, Alexander Wiesmaier, and Panayotis Kikiras. "A view on privacy & trust in IoT." Communication Workshop (ICCW), 2015 IEEE International Conference on. IEEE, 2015.

[3] Funke, Sebastian, et al. "End-2-End privacy architecture for IoT." Communications and Network Security (CNS), 2015 IEEE Conference on. IEEE, 2015.

[4] Arasteh, Sima, Seyed Farhad Aghili, and Hamid Mala. "A new lightweight authentication and key agreement protocol for Internet of Things." Information Security and Cryptology (ISCISC), 2016 13th International Iranian Society of Cryptology Conference on. IEEE, 2016.

[5] Liu, Fang, and Tong Li. "A clustering-anonymity privacy-preserving method for wearable iot devices." Security and Communication Networks 2018 (2018).

[6] Otgonbayar, Ankhbayar, Zeeshan Pervez, and Keshav Dahal. "Toward anonymizing iot data streams via partitioning." Mobile Ad Hoc and Sensor Systems (MASS), 2016 IEEE 13th International Conference on. IEEE, 2016.

[7] Personal Data Protection Commission of Singapore, "Guide to Basic Data Anonymization Techniques", https://www.pdpc.gov.sg/-/media/Files/PDPC/PDF-Files/Other-Guides/Guide-to-Anonymisation_v1-(250118).pdf

[8] Jebri, Sarra, Mohamed Abid, and Ammar Bouallegue. "An efficient scheme for anonymous communication in IoT." Information Assurance and Security (IAS), 2015 11th International Conference on. IEEE, 2015.

[9] Moldovan, George, Anda Ignat, and Martin Gergeleit. "Group-Based Anonymous On-Demand Routing Protocol for Resource-Restricted Mobile Ad Hoc Networks." International Conference on Ad-Hoc Networks and Wireless. Springer, Berlin, Heidelberg, 2013.

[10] Diffie, Whitfield, and Martin Hellman. "New directions in cryptography." IEEE transactions on Information Theory 22.6 (1976): 644-654.

[11] Palmieri, Paolo, Luca Calderon, and Dario Maio. "An Anonymous Inter-Network Routing Protocol for the Internet of Things." Journal of Cyber Security and Mobility 6.2 (2017): 127-146.

[12] Mutalemwa, Lilian, and Seokjoo Shin. "Strategic Location-Based Random Routing for Source Location Privacy in Wireless Sensor Networks." Sensors 18.7 (2018): 2291.

[13] Hoang, Nguyen Phong, and Davar Pishva. "A TOR-based anonymous communication approach to secure smart home appliances." Advanced Communication Technology (ICACT), 2015 17th International Conference on. IEEE, 2015.

[14] Farooq, Muhammad Umar, et al. "A critical analysis on the security concerns of internet of things (IoT)." International Journal of Computer Applications 111.7 (2015).

[15] Gope, Prosanta, and Tzonelih Hwang. "Untraceable sensor movement in distributed IoT infrastructure." IEEE Sensors Journal 15.9 (2015): 5340-5348.

[16] Landsiedel, Olaf, et al. "Core: A Peer-To-Peer Based Connectionless Onion Router." Proceedings of IEEE GLOBECOM, Globalcommunications Conference (Washington DC, USA. 2007.

[17] Suo, Hui, et al. "Security in the internet of things: a review." Computer Science and Electronics Engineering (ICCSEE), 2012 international conference on. Vol. 3. IEEE, 2012.

[18] Vermesan, Ovidiu, and Peter Friess, eds. Internet of things: converging technologies for smart environments and integrated ecosystems. River Publishers, 2013.

[19] Cao, Jianneng, et al. "Castle: Continuously anonymizing data streams." IEEE Transactions on Dependable and Secure Computing 8.3 (2011): 337-352.

[20] Zakerzadeh, Hessam, and Sylvia L. Osborn. "FAANST: fast anonymizing algorithm for numerical streaming data." Data privacy management and autonomous spontaneous security. Springer, Berlin, Heidelberg, 2011. 36-50.

[21] Kamat, Pandurang, et al. "Enhancing Source-Location Privacy in Sensor Network Routing." The 25th IEEE International Conference on Distributed Computing System, 2005,599-608.

[22] Tor Project, "You should people choose their path length", https://2019.www.torproject.org/docs/faq.html.en#ChoosePathLength