# A Secure Key Management Technique Through Distributed Middleware for the Internet of Things

Tamanna Tabassum(✉), SK Alamgir Hossain, and Md. Anisur Rahman

Computer Science and Engineering Discipline, Khulna University, Khulna, Bangladesh
tamannaku07@gmail.com, alamgir@cseku.ac.bd, anis@cseku.ac.bd

**Abstract.** Internet of Things has made dramatic change in our life. Consumer, governmental and business trends are pushing us towards the IoT. 1.9 billion devices are being used today, and more than 50 billion will be used by 2020. So with the increase of internet connected devices, there is an increasing need for strong security measures. The sensors are manufactured by different companies with different processing power. In this huge number of low power sensors how the data will be protected from different attacks? As IoT data need to transit through different gateways and cloud to reach its final destination so what will happen if any attacker change the data or in any other way take the control and destroy the whole system. Our main target is to ensure a protected and secure IoT by introducing a secure key exchange mechanism, along with a key replacement process. As from our initial experimentation, we found out that the proposed approach of distributed key management technique is appealing and performing good with respect to several factors like robustness, delay, overhead, malicious attacks, etc.

**Keywords:** Internet of Things (IoT) · Encryption · Security protocol

## 1 Introduction

Internet of Things (IoT) is showing a significant role in this modern world of technology that every smart device is trying to connect to the Internet. The IoT connects almost all the facets of individual's life like smart environment and health-care to wearables. Different entities can communicate and interact to provide different services. This smart nature of things lead to many applications like smart cities, logistics, smart agriculture, home automation, health care, military surveillance, security, etc. [1–5].

IoT allows electronic devices in our surrounding environment to be active participants by sharing information with other members of the network making it possible to recognize events and changes in their surroundings and to act and react autonomously without any human interaction [6]. The advantages of IoT


K. Arai et al. (Eds.): SAI 2018, AISC 858, pp. 1128–1139, 2019.
https://doi.org/10.1007/978-3-030-01174-1_86

are almost limitless and its applications are changing the way we work and live by saving time and resources, and opening new opportunities for growth, innovation, and the exchange of knowledge between entities. However, the existence of such a large network [7] of interconnected entities will show new problems like security, privacy, and trust threats that all the connected devices are in a high risk.

Internet is the foundation of IoT hence almost all the security threats that are within Internet propagate to IoT as well. Furthermore, the fast development and wider adoption of IoT devices in our lives signifies the urgency of addressing these security threats before deployment. As the interconnected devices is related to the users lives so there is a need for a well-defined security threat classification and a proper security architecture that can resolve the security problems regarding privacy, data integrity, and availability in IoT [8].

Among different key management approaches the simplest approach is to pre-distribute the cryptographic keys to the sensor nodes. But there are several limitation of this approach: (i) consider a scenario, a dedicated single key is being shared among all the nodes and the problem is if a single device is attacked by an attacker the full system may be compromised; (ii) another approach where a dedicated key is used for each node is also useless as million of nodes in the network and it is very difficult to scale up in that extend. To solve the above problem Eschenauer et al. [9] proposed a random key generation and sharing process. Their scheme is to select a random set of keys from a key table and assign to each node. And they assign the keys in such a way that any two nodes receive a shared key so that they can initiate the secure communication. The problem of this method is if one key is compromised there is a chance to gain access of the full system.

A classical XOR based key construction technique is presented by Pietro et al. [10]. This method is popular as the technique is faster with respect to resource use as well as for protecting from malicious attacks. As the IoT sensor nodes have limited capacity a light weight key management protocol is recommended. Abdmeziem et al. [11] introduced a similar technique in electronic health scenario. According to their technique a separate process is responsible for executing the cryptographic primitives and certificate control. But in evaluation they found that using a third party certificate provider impact the performance and scalability. Being motivated by [11] the authors [12] and [13] introduced a group key exchange mechanism for sensor nodes but still in reality most of the cases the nodes are independent and is very difficult to group them in physically or logically.

Our contribution in this paper is: at first we present a distributed key management technique and we provide an optimized key management algorithm that will ensure security for IoT systems. The remaining of this paper is organized as: In Sect. 2 we presented our proposed approach and provide a detail of the proposed system. Further in Sect. 3, we describe the implementation issues, and some evaluation strategies. Also, in Sect. 4, we present the result that we found from our experiment. At the end we provide conclusion of the paper in Sect. 5 and state some possible future work directions.

## 2 Proposed Key Management Method

In this section we present our proposed key management technique and their working process. A secure key exchange mechanism, with a key replacement technique, is essentials for message transfer. In order to transfer the message between sources a flexible middle-ware is necessary. In our research we considered a Smart Middleware that consists of a collection of Smart Objects (SO) that have the facility to store, register and process IoT node data. In a complete network millions of SO can be functional and their main duty is to minimize the gap between the data processing unit to the source nodes. If any source node wants to exchange its data it may send registering or in an anonymous way. If it wants to send data in a secure way the node must be registered with SO and SO assigns the proper keys for encrypting any future transmitted information. The encryption algorithm may vary one SO to another SO. As multiple source nodes may continuously send data to the SO, data will be stored in raw data storage. Later the SO processes the data from the storage and normalizes the data and finally send data to the users/subscribers.

In our approach each message has mainly two parts: one is the data part and other is the control part. Consider a scenario where a source node $A$ wants to send message $m$ to a end node $B$ (and there is a set of intermediate nodes $N_0$, $N_1$,..., $N_r$). The control part (is important for message routing) of the message is only read by the intermediate nodes and the data part is only read by the end node. In order to establish a secure communication between the network nodes, we proposed a secure encrypted key that is linked with every pair of consecutive nodes. By using this we want to take advantage of the symmetric-key encryption technique. So consider $K_0$ is a cryptographic key that is shared between the source node $A$ and the next receiving node $N_0$; according to our technique the control part of the message is encrypted in the source node $A$ and decrypted in node $N_0$ using the current shared key $K_0$. The similar process ($K_1$ will be shared between $N_0$ and $N_1$, $K_2$ will be shared between $N_1$ and $N_2$ and so on) will continue until the message reaches the end node $B$. The data part of the message is encrypted by the source node $A$ which is only decrypted by the end node $B$.

### 2.1 Keys, $K_i$

In our method two sets of keys are used for each node: one is the global and another one is the local keys. Each key is further divided into two parts: name and the value part. The format of local key name is $K = \langle K_{node}, K_{incr} \rangle$, if the network has up to $2^d$ nodes then $K_{node}$ is placed into the $d$ most significant bits of $K$ and $K_{incr}$ is placed into the $d$ least significant bits of $K$. At the beginning of the system operation the local key is assigned to 0 and after each action it is incremented by 1. The format of global key name is the order number of that key when the key is generated.

### 2.2 Communication Channels, $C_i$

Two nodes are adjacent if they can send message to each other in any way either a direct link or through some intermediate links. Two nodes (say $A$ and $B$) are called adjacent if they can distribute message through a direct link, in other words if they can hold a same key $K$. The connection of this two adjacent node is denoted by $\{A, B\}_K$, which means $A$ is directly connected with $B$ sharing the key $K$ for any secure message transfer. In each node $M$, 3 tables will be used for communication channels (Fig. 1):

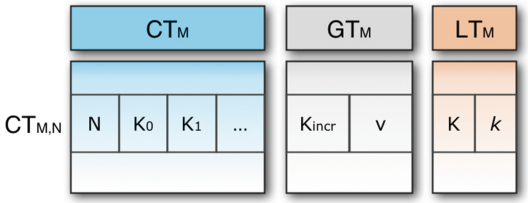


**Fig. 1.** Tables used for communication channels.

- *Connection table,* $CT_M$: This table contains a single entry for each node that are connected to one another. The row $CT_{MN}$ is for node $N$ and a list of local keys $K_0$, $K_1$, etc. It should be noted here that as the connections are bi-directional, $CT_{M,N}$ and $CT_{N,M}$ hold the same key.
- *Global key table,* $GT_M$: This table contains one entry for each key $K$ for which $K_{node} = M$. The entry reserved for key $K$ contains the incremental name $K_{incr}$ of this key and the value $v$ of this key.
- *Local key table,* $LT_M$: This table contains a single entry for each local key $K$ generated by another node (i.e. $K_{node} \neq M$). $LT_M$ is required because if a source node tries to send message to a destination where it is not registered before. In this case the communication will be established through the help of local key table. As source node do not generate local keys, they only store 2 local tables: the connection table, and the global key.

Now we would like to discuss the process and the different key table structure of message $m$ transfer from a source node $A$ to destination node $B$ through a direct link. The process is demonstrated in Fig. 2.

(1) If $CT_{AB}$ exists in $CT_A$ and $CT_{BA}$ also exists in $CT_B$ this means the connection are bidirectional. If $CT_{AB}$ exists but $CT_{BA}$ does not exist then $A$ can send message to $B$ demanding the appropriate global key. When $B$ receives message then it looks up for appropriate global key $G$ and acknowledge the global key to the source $A$. If no global key exists for $A$ and $B$ then acknowledge an empty global key. In this case node $A$ tries to send the message in another route. If no paring is possible it means that the network is disconnected.

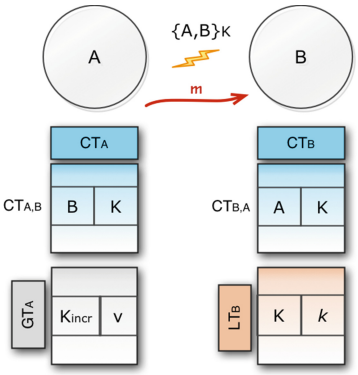


**Fig. 2.** Process and the different key table structure of a message transfer from node $A$ to node $B$.

(2) Source $A$ generates a new local key $K = \langle K_{node}, K_{incr} \rangle$ when it receives the acknowledgement (global key) from $B$ and also inserts an entry in global key table $GT_A$. Finally $A$ calculates the value $k = f(v)$ ($f(v)$ is a key conversion function of $K$) and send the pair $\langle K, k \rangle$ to node $B$.
(3) When $A$ sends $\langle K, k \rangle$ to node $B$, it then stores it in its local table $LT_B$ and reverses entry for $CT_{B,A}$. The key setup will be closed by sending a positive reply message from $B$ to $A$. Finally the key setup is done and $A$ can send message to $B$ encrypted by using $K$ while $B$ decrypts the message.

We discussed for a direct communication now we would like to present the process when the message is sent via an intermediate node. The process is demonstrated in Fig. 3.

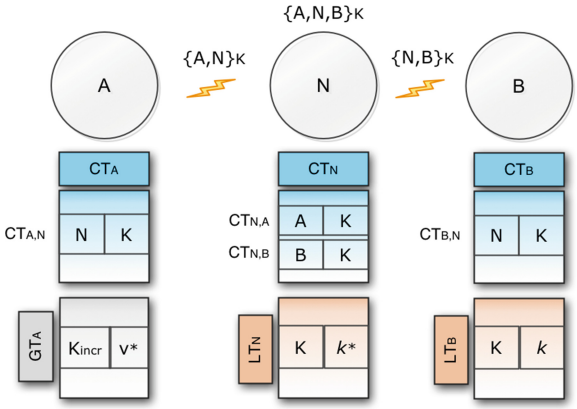


**Fig. 3.** Message transferring via an intermediate node $N$.

(1) In this case node $A$ sends message $m$ to end node $B$ through an intermediate node $N$. So $A$ and $N$ have $CT_{AN}$ and $CT_{NA}$ respectively for bidirectional link between $A$ and $N$. Similarly $N$ and $B$ have $CT_{NB}$ and $CT_{BN}$, respectively for bidirectional link between $N$ and $B$.
(2) $A$ uses connection $\{A, N\}_K$ for sending message to $N$ as well as $N$ uses connection $\{N, B\}_K$ for sending message to $B$.
(3) Now if $A$ wants to send message to $B$ it first pairs with $A$ and $N$ with the same process as we discussed in case of a direct link. When $N$ receives the message it then pair with $B$ to send the message to $B$ in the same manner. Finally the secure communication channel $\{A, N, B\}_K$ is setup.

The technique can be extended to any sequence of nodes where $r+3$ number of nodes $(A, N_0, N_1, ..., N_r, B)$ are available. And finally we can say that a secure communication is possible for the message being sent from node $A$ to node $B$ through an arbitrary list of intermediate nodes $N_0, N_1, ..., N_r$.

### 2.3 Key Update

In many cases nodes need to update the local key with the new value. When the local key needs to update a flooding/broadcasting, local key update message is initiated by a node so that all the associated nodes can update their local key table. The key update processes is as follows:

(1) In the first step $A$ lookup $CT_A$ for all entry of $K$ then send message change command to all the link nodes with $A$ that are sharing $K$. It should be noted here that if no such key is found for the link, global key is used as we discussed in the earlier section.
(2) Now say $B$ receives change message from $A$. In order to ensure security $B$ verifies message $m$ by using the $f(v)$ which is the key conversion function of $K$ which it stored earlier in its $LT_B$ table. So it extracts $v$ from the message and match in $LT_B$. If no match is found then it discards the request, if found then updates its entry for the key $K$.

The above key generation and distribution technique is illustrated in the following Algorithm 1.

## 3 Implementation

In this section we describe the implementation process of our prototype that represents the technique we presented in Sect. 2.

### 3.1 Experimental Setup

In our prototype (Fig. 4), we have a layer of Smart Objects (SOs) that are responsible for processing and controlling the messages. SOs are smart nodes that have a computational capability and usually deployed in a distributed manner.

**Algorithm 1.** *Key Generation and Distribution*

```
/* if the algorithm fails, A is disconnected from the network.     */
if {A, B}_K exists in CT_A then
    φ(A, B, m, K);
else
    G = findGlobalKey(A, B);
    if G ≠ null then
        A generates a new local key K = ⟨A, K_incr⟩;
        φ(A, B, m, G);
        sendAck(A, B);
        updateCT(A, B);
        A send message m to B, using K and can decrypt m from B with G;
    else
        A link with B via adjacent node N with link {A, N}_K;
        φ(A, N, m, K);
        if {N, B}_K exists then
            φ(N, B, m, K);
            updateCT(A, B);
            setConnection(A, N, B, K);
        else
            if {N, B}_K' exists, where K ≠ K' then
                φ(N, B, m, K') with K=⟨K_node, K_incr⟩ received by A;
                sendAck(B, N);
                updateCT(N, A, B);
                setConnection(A, N, B, K);
            else
                if {N, B}_G exists then
                    φ(N, B, m, G) with K=⟨K_node, K_incr⟩ received by A;
                    sendAck(B, N);
                    updateCT(N, A, B);
                    setConnection(A, N, B, K);
                else
                    No G exist that is using N, B;
                    N link with B via adjacent nodes;
                    while islink(B) or isAvailable(adjacent nodes) do
                        searchAdjacentNodes();
```

**Algorithm 2.** *φ (A, B, m, K)*

```
/* Node A sends message m to node B encrypt with key K and finally
   B decrypts encrypted m with K                                    */
m^e ← encrypt(m, K);
sendMessage(A, m^e, B)
m ← decrypt(B, m^e, K);
```

It should be noted that the SOs have the capacity to connect with the nearby IoT sensors or devices by using different communication channel like NFS, WiFi, direct wire connection or through the Internet. In our prototype we used Apache Cassandra [14] for storage management, and Mosquitto [15] for message broker for MQTT [16] publish/subscribe mechanism. Our preference on Cassandra over other NoSQL database like MongoDB its ability to scale while still being reliable, is because it is possible to deploy Cassandra across multiple servers built-in without a lot of extra work.

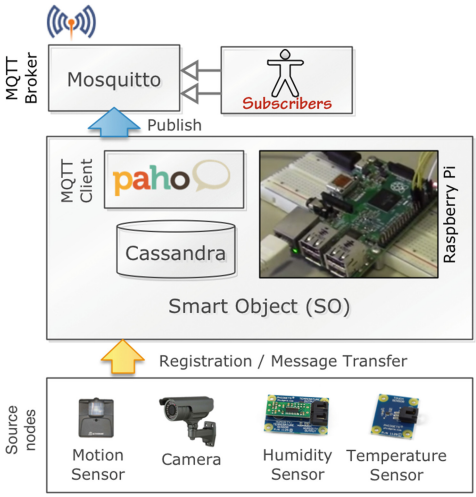


**Fig. 4.** Our implemented prototype.

In an IoT environment as resource consumption and power is a big issue so all the implementation should be in such a manner that the resource-constrained devices can work with maximum life time. We suggested in our approach the MQTT [16] which is a pub/sub protocol and specially developed for resource constrained environment like IoT. So whenever a data source wants to send message it publish the message with a pool of pre selected topic. The sensors or smart devices that want to receive the message they need to subscribe with a published topic. In this simple model the full message communication can be easily handled in the complex IoT environment. A sample topic is demonstrated in Fig. 5. It is important for pub/sub model that sources may subscribe in a topic at runtime and at any-time can un-subscribe as well.

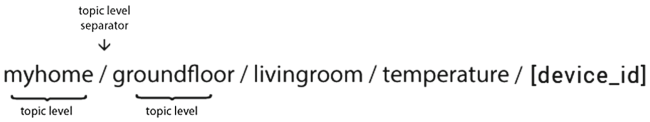


**Fig. 5.** MQTT topic.

## 4 Result

To perform the feasibility study, and to understand the importance of the system, the detailed analysis is performed on various cases. We setup our *SO* nodes on several Raspberry Pi and simulate the outcome in a real environment. In our setup we used 10 sensors (3 temperatures, 2 humidity, 4 motions, and 1 camera) to sense the environment and all the data retrieve through HTTP GET and REST api. To ensure SO based prototype system performs with the sensors and the subscribers we run prototype setup over a week with appropriate performance thread hook so that we can record the behaviour on time. The analysis results are as follow:

### 4.1 Key Generation and Distribution Time

We are using Algorithm 1 for key generation and distribution. In order to identify the processing time of this algorithm we setup our prototype with 5 sensors (3 temperature and 2 humidity) and before sending the data to the SO we complete the registration of the sensors to the SO and each sensor will send 1KB of message to SO. In this case we only consider direct link between SO and the sources. During the registration step each sensor assigned an unique identifier to recognize each sensor uniquely. We run the process 10 times to get the averages processing time. The result is shown in Fig. 6. From this test we identified that the algorithm will cost approximately 21 ms. From the figure we found that sometimes the delay is more because the overall time depends on the current load of the SO as well as the network speed. We perform the same test again but in this case we consider that a node want to send data to another node via the SO by using our key management protocol and we identified that the average delay time is 32 ms (approx.).

### 4.2 Service Response Analysis

We measured the sending time from the source node to the subscriber by using the following (1). Where Key generation and access time is $Z$ unit which we identified in case of direct link is 21 ms and 32 ms in case of via $SO$, the message size is $n$, the HTTP server delay with transmission time is $\Pi$ and $\beta$ for sending time from SO to the MQTT client.

$$T = Z + \frac{n}{\Pi} + \beta \tag{1}$$

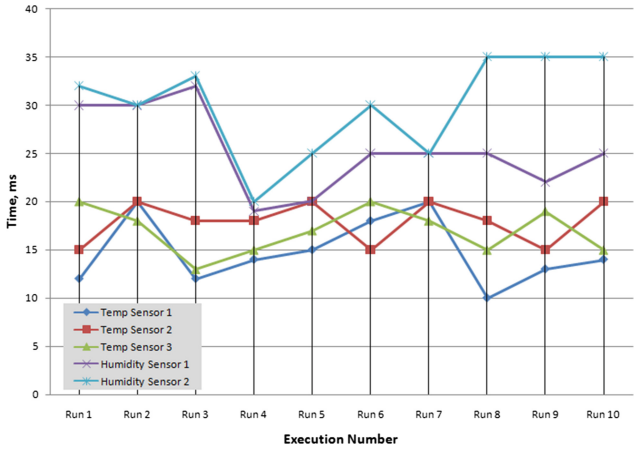


**Fig. 6.** Key generation and distribution time.

So when a source generates a message, the system requires maximum $T = (32 + 270 + 244)\,ms$ (approx.). Here, 32 ms is the average of the key access time, 270 ms is the network overhead and $\beta$ is 244 ms. The above response time is in case of a single source, but practically multiple sensors can interact with each other at the same time. The developed Smart Object(SO) supports transfer requests from multiple nodes. For example, when a node sends message to the SO it creates a queue of request on a First Come First Serve (FCFS) basis. In such cases the Response Time can be calculated by using the Little's Formula, which is a classical conservation equation in queuing theory.

$$E(T) = \frac{E(n)}{\lambda} \tag{2}$$

Here, $E(T)$ is the average delay for a source node message request, i.e. average throughput time, $E(n)$ is the average number of requests to the SO. $\lambda$ is the arrival rate of the nodes requests. However,

$$E(n) = \frac{\rho}{1-\rho}\ and\ \rho = \frac{\lambda}{\mu} \tag{3}$$

Where, $\rho$ is the fraction of time the SO requires to process the requests. Hence, combining (2) and (3) we get,

$$E(T) = \frac{\rho}{\lambda(1-\rho)} = \frac{\frac{\lambda}{\mu}}{\lambda(1-\frac{\lambda}{\mu})} = \frac{1}{\mu-\lambda} \tag{4}$$

Here, $\mu$ is request processing rate from the queue, and (4) is the queuing delay. In this equation $\lambda$ is the arrival rate of a request by a node and $\mu$ is the average service rate. In our experiment we measured that the average service time is approximately 2.5 s. Hence, average service rate $\mu = \frac{1}{2.5} = 0.4$ per second.

For example, in case after every 60 s a requests is triggered to the SO then $\lambda = \frac{1}{60} = 0.0167$ request per *second*. Therefore, from the Little's Formula (4), the total waiting time including the service time $= \frac{1}{0.4-0.0167} = 2.61$ s (approx.).

### 4.3 Robustness in Case of Malicious Attacks

Two types of malicious attacks may be happen: attack performed by external entities (those are not part of the network) and attack performed by the registered sources. Our protocol shows robustness in case of external attack. It is difficult to derive easily the global key from a value of another global key. Still if one global key is eavesdropped, the whole network will remain secure and only a part of the network will be affected. To reduce the risk we used local keys as a extra layer of security. The external attacker may eavesdrop the local key name, but if wants to know the value of the key he has to know the encryption algorithm. We did not consider the physical attack by which the attacker may steal the global/local keys. When any source will be registered in the SO, the node has full access to the local and global keys that is shared between them so if the registered sources may attack the network it is hard to protect. We want to focus this issue in the next extension of our research.

## 5 Conclusion

IoT is expanding rapidly and as IoT is using internet as a connection medium so secure connection mechanism is very necessary. In this paper we presented a secure key distribution system, along with a key replacement mechanism. In our proposed system a distributed middleware layer, named Smart Object (SO), able to manage heterogeneous data sources, to provide a uniform, consistent data representation and to evaluate the security and quality level associated to each data unit. In particular, a proper security algorithm has been developed in order to assess the trustworthiness of registered and non registered IoT data sources. The effectiveness of the proposed solution has been validated through the implementation of a real prototype. Through our experiment, we found out that the proposed approach is appealing and performing well in terms of overhead, delay and robustness towards malicious attacks. Here we only consider single smart object and would like to consider our key management protocol in multiple SO cases. However, we believe that our proof will remain as a motivation for further research in this area.

## References

1. Sicari, S., Rizzardi, A., Cappiello, C., Miorandi, D., Coen-Porisini, A.: Toward data governance in the internet of things. In: New Advances in the Internet of Things, pp. 59–74. Springer (2018)
2. Xu, Q., Aung, K.M.M., Zhu, Y., Yong, K.L.: A blockchain-based storage system for data analytics in the internet of things. In: New Advances in the Internet of Things, pp. 119–138. Springer (2018)

3. Amin, R., Islam, S.H., Biswas, G., Khan, M.K., Leng, L., Kumar, N.: Design of an anonymity-preserving three-factor authenticated key exchange protocol for wireless sensor networks. Comput. Netw. **101**, 42–62 (2016)
4. Barreto, L., Celesti, A., Villari, M., Fazio, M., Puliafito, A.: An authentication model for iot clouds. In: Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015, pp. 1032–1035. ACM (2015)
5. Benabdessalem, R., Hamdi, M., Kim, T.-H.: A survey on security models, techniques, and tools for the internet of things. In: 2014 7th International Conference on Advanced Software Engineering and Its Applications (ASEA), pp. 44–48. IEEE (2014)
6. Singh, D., Tripathi, G., Jara, A.J.: A survey of internet-of-things: future vision, architecture, challenges and services. In: 2014 IEEE world forum on Internet of things (WF-IoT), pp. 287–292. IEEE (2014)
7. Bradley, J., Barbier, J., Handler, D.: Embracing the internet of everything to capture your share of $14.4 trillion. White Paper, Cisco (2013)
8. Mattern, F., Floerkemeier, C.: From the internet of computers to the internet of things. In: From Active Data Management to Event-based Systems and More, pp. 242–259. Springer (2010)
9. Eschenauer, L., Gligor, V.D.: A key-management scheme for distributed sensor networks. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 41–47. ACM (2002)
10. Di Pietro, R., Mancini, L.V., Jajodia, S.: Providing secrecy in key management protocols for large wireless sensors networks. Ad Hoc Netw. **1**(4), 455–468 (2003)
11. Abdmeziem, M.R., Tandjaoui, D.: An end-to-end secure key management protocol for e-health applications. Comput. Electric. Eng. **44**, 184–197 (2015)
12. Veltri, L., Cirani, S., Busanelli, S., Ferrari, G.: A novel batch-based group key management protocol applied to the internet of things. Ad Hoc Netw. **11**(8), 2724–2737 (2013)
13. Yu, H., He, J., Zhang, T., Xiao, P.: A group key distribution scheme for wireless sensor networks in the internet of things scenario. Int. J. Distrib. Sens. Netw. **8**(12), 813594 (2012)
14. Apache, Apache cassandra, Technical report. http://cassandra.apache.org
15. Eclipse, Mosquitto, Technical report. https://mosquitto.org
16. OASIS, Mqtt 3.1.1 specification, Technical report. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html