IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 14, 2021, accepted August 31, 2021, date of publication September 6, 2021, date of current version September 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3110707

A Robust Device-to-Device Continuous Authentication Protocol for the Internet of Things

ARWA BADHIB, SUHAIR ALSHEHRI, AND ASMA CHERIF
Department of Information Technology, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia

Corresponding author: Arwa Badhib (adhib@stu.kau.edu.sa)

This project was funded by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, under grant No. (DG-10-612-1441). The authors, therefore, gratefully acknowledge the DSR technical and financial support.

ABSTRACT The Internet of Things (IoT) is a heterogeneous environment that connects billions of devices. Thus, it is a significantly high-value target for attackers and suffers from several threats, especially impersonation attacks during the session. Moreover, the denial of service attack (DoS) threatens IoT environments, as it affects the availability and energy of communicating devices. Continuous authentication solves session hijacking since it checks user legitimacy during the session. Several continuous authentication schemes were proposed to authenticate users to IoT devices, while few works addressed device-to-device authentication. Therefore, it is essential to authenticate devices because if one device gets compromised, then the whole system is at risk. Continuous authentication between devices differs from user-to-device authentication since it cannot rely on biometrics and passwords. This research proposes a fast and secure device-to-device continuous authentication protocol that relies on devices' features (token, battery, and location), and mitigates DoS attacks using shadow IDs and emergency keys. Moreover, it takes the sensor movement into account while preserving privacy. To evaluate the robustness and validate the security of the proposed protocol, we conducted informal and formal analyses using Scyther. In addition, we tested its performance to establish computation costs relative to the system counterparts. The results show the protocol is robust against security threats, incurring reasonable computational costs.

INDEX TERMS Continuous authentication, device-to-device, DoS, IoT, privacy, security.

I. INTRODUCTION

The Internet of Things (IoT) is the next generation in communication devices. It is also a network that combines a large number of devices, including sensors, radio-frequency identification (RFID), actuators, etc. The aim of this technology is to provide better services for humans by enabling smart homes, smart healthcare, smart industries, and smart cities [1]. Making human lives easier, smartwatches monitor our movement and blood pressure and mobile devices manage our plans/times and events, while lights manage our electricity, and so on. According to Gartner's report, almost 80 percent of organizations and companies now use the IoT to manage their requirements [2]. The main concept behind IoT technology is not only to connect inter-network components to the Internet but also to connect non-IP components (lights, fans, washing machines, televisions), so that they can collect and exchange data. According to estimations, the market for IoT devices is expanding rapidly, and is currently expected to reach around 75 billion by 2025 [3].

Despite the expansion and variety of IoT applications, and its capacity to improve human life, major security and privacy issues remain to be resolved. As IoT devices are constrained in terms of memory, CPU, battery, and power, it is impossible for them to secure themselves against attacks [4]. As a result, they can be readily compromised; for example, the Mirai botnet attack in 2016. This attack compromised millions of IoT devices, to perform a DoS attack resulting in damage to critical servers, such as Amazon, Twitter, Netflix, and New York Times [5]. Two additional DoS attacks have been performed since Mirai: Hajime and Reaper. These attacks involved exploitation of a large number of IoT devices that mostly used default passwords, rendering them easy for attackers to compromise [6]. The IoT is a remarkable target for attackers, as in the last three years, 20 percent of organizations have discovered attacks via IoT devices [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Maaz Rehan.

Enforcing security via IoT devices imposes several requirements; i.e., confidentiality, availability, integrity, and authentication [7]. In this research, we focus on authentication for security, as it allows only authorized users/devices to access the system. Authentication acts as a gatekeeper protecting systems from intruders and incursions by strangers. Authentication refers to processes for verifying the identity of the claimer, using identifiers or credentials [8]. These identifiers can be something the user knows: ID number, passwords, secure question, etc. or something the user possesses, e.g., token, ID card, etc. or the user's biometric, such as: iris, fingerprint, voice, etc. These credentials or identifiers need to be unique, easy to collect, store, simple and acceptable to the user [9]. Users can use one or more of these identifiers to prove his/her legitimacy.

IoT communicates a large number of various devices, so one of the most important key aspects here is to ensure IoT security as a means of authenticating communication devices [10]. Device authentication is just as important as user authentication. It ensures all devices for communicating are trusted, and can communicate and share resources safely. It is noteworthy that if any device is compromised by an attacker, then the entire system will be in danger, resulting in catastrophic damage. For example, an attacker can control devices, the network or the system; he can also access critical information and privilege escalation by performing a chain attack, as happened recently in the case of the SolarWinds supply chain attack, which caused other systems (FireEye and others) to become infected [11], [12].

Authentication of IoT devices incurs several challenges, since they have limited capabilities in terms of both software and hardware. Additionally, the heterogeneous nature of the IoT environment renders traditional authentication techniques inapplicable. Thus, new techniques must be developed to match IoT characteristics [13]. Recently, several researchers have proposed lightweight authentication schemes for IoT. Many of these solutions employ static authentication, which verifies the user/device just once at the beginning of each session. This type of authentication is vulnerable to certain attacks, such as session hijacking because it checks the user only at the beginning of each session. Accordingly, an attacker may interrupt and steal a legitimate session by acting as a legitimate user. To illustrate this, consider a smart home containing sensors designed to collect data about the house's condition (i.e., temperature, electricity usage, camera data, etc.) to be sent directly to the gateway for storage in the cloud server. For each session, sensors are authenticated by the gateway, but a serious problem arises if an attacker steals this valid session and impersonates a legitimate user. In this case, all the information and privileged services can then be accessed by the attacker. As a result, the attacker might try to engage in several harmful actions, such as opening the door lock, spying using a camera, manipulating electricity usage, etc.

To overcome problems related to static authentication, continuous authentication has been introduced. Continuous authentication will periodically check the legitimacy of the user/device during the session, preventing impersonation at any time during the session [14]. Additionally, it ensures rapid authentication of the large body of data transmitted between devices over a short period; so instead of re-authenticating the devices each time prior to re-transmission, continuous authentication is then used to speed up the authentication process.

Several authors proposed continuous authentication, and mainly focused on authenticating user-to-device (U2D), e.g., [15]–[18], etc. Therefore, this research focuses on continuous authentication between devices referred to as device-to-device (D2D), where devices deal with and communicate with each other in the IoT environment. D2D covers several applications, such as: smart farming, smart cities, smart grids, etc. [19]. Thus, D2D authentication has become a major issue that needs to be taken into consideration [20]. The majority of the current D2D authentication techniques support static authentication, which may lead to some problems as stated previously. In D2D authentication, the system deals with devices; thus, the authentication mechanisms vary from the users' authentication. Thus, strategies like biometric, memorable patterns, passwords, etc. are insufficient [21]. One way to authenticate devices is to use some of their features or contextual information to confirm their legitimacy. The device's features should be carefully selected to improve the authentication process. Moreover, the use of multiple features or contextual information to authenticate users would increase the reliability and security of the authentication process [18].

Furthermore, the IoT consists of a large number of devices and sensors. The majority of these sensors can move freely; i.e., wearable sensors, monitoring patient sensors, vehicle sensors, etc. Critical environments (e.g., medical, governments, financial facilities) use IoT devices that might otherwise be misused by authorized users. For instance, a Covid-19 patient recommending applying social distancing might break guidelines and spread the virus. Using a wearable device like a non-removable bracelets would ensure patients do not leave designated quarantine areas while allowing free movement within a specified region. Meanwhile, it is important to preserve a user's privacy (identity and location).

The main contributions of this research are:

- We design a D2D continuous authentication protocol, that utilizes devices' features (i.e. battery capacity, location, token) to continuously verify devices during the session.
- We balance between security and IoT constraints by using lightweight computations (HMAC, hash, XOR), as IoT devices have limited software and hardware capabilities.
- We mitigate DoS attacks, by using a set of shadow IDs and emergency keys to prevent any synchronization loss between devices. Furthermore, we take the gateway unavailability into consideration to enhance system's availability.

- We define a movement model to ensure a secure movement for system devices, besides preserving their privacy (identity and location).
- We conduct formal and informal security analysis to prove the effectiveness and robustness of the proposed model using Scyther. We also compare it with its competitors. The results show our protocol performs well in achieving all the essential functionalities, and is robust against the defined attacks.

The remainder of the paper is organized as follows: section II, presents authentication schemes for the IoT including both static and continuous. In section III, the proposed model is described including the architectural design and communication protocol. Section IV presents the mitigation of the DoS attack. Section V provides how the protocol handles sensors movement. Section VI tests the protocol's robustness by providing security and a performance analysis. In section VII, discussion and results are provided while comparing the proposed protocol with its competitors, and finally section VIII concludes the research.

II. RELATED WORK

The IoT is an open environment, comprising a heterogeneous number of devices that communicate with each other. This open environment is vulnerable to different security threats; as a result authentication schemes must be applied to communicate with these devices. In this section, we present several previous research that has provided authentication schemes for the IoT. These schemes are categorized by static and continuous authentication.

A. STATIC AUTHENTICATION

Gope and Hwang [22] proposed a lightweight D2D authentication protocol for IoT distributed Wireless Sensor Network (WSN). This scheme achieves privacy by providing anonymity and untraceability for both location and identity. It considers the sensor's movement. This solution can be applied to RFID systems and healthcare environments. The protocol prevents several attacks, such as: DoS, cloning, impersonating and replay attacks. However, it is vulnerable to man-in-the-middle attacks, since there is no key agreement. Additionally, the key for this protocol is not secure, because it is just XORed with the random value without providing any cryptographic mechanisms [23].

Amin *et al.* [24] proposed a lightweight authentication protocol for IoT in the cloud. It provides anonymity to protect the privacy of users. This protocol uses the username, password and smartcard to authenticate users. Zhou *et al.* [25] demonstrated that Amin *et al.* [24] protocol is susceptible to user tracking and off-line guessing attacks. As a result, they improved the previous protocol by providing user untraceability and auditability. However, both schemes [24] and [25] still suffer from a single point of failure, have high computations in the cloud [11] and fail to provide mutual authentication [26].

Kim and Lee [27] proposed an authentication and authorization framework for the IoT environment, which is locally centralized and globally distributed. This framework uses open-source software called Auth [28], which can be located in edge devices as a way to authorize local registered IoT devices. Additionally, these Auths have to manage trusted relationships among each other globally. All the credentials and access control policies of local IoT devices are stored in the Auth. If a device in one Auth wants to communicate with a device in another Auth, both Auths must participate to authorize these devices. This scheme prevents from DoS attack because it uses the globally trusted relationship among Auths [29]. Nevertheless, the process of initialization is still high, as is the problem of dealing with mobile devices' authorization. It relies upon symmetric encryption, which makes the computation cost high.

Several research works were proposed to authenticate devices in RFID systems. Gope *et al.* [30] presented an authentication protocol for RFID based systems used in smart cities. This work improves upon previous work presented in [22] for RFID systems. It provides user anonymity and untraceability. This scheme was developed to solve the problem of synchronization loss. Additionally, this protocol provides secure localization if the server asks the tag for its own location. However, it does not prevent man-in-the middle attack and fails to prevent a collision attack when two tags have the same track sequence number [31]. Another RFID protocol was proposed by Safkhan and Vasilakos [32] to authenticate devices in telecare medicine information systems (TMIS). It is a lightweight scheme based on implementing hash function, XOR and concatenation operations to authenticate devices. It provides mutual authentication, synchronization and prevents from replay attacks. On the other side, both the tag and the reader depend on their IDs in order to generate secret keys. However, these IDs are constant and not updated after each process. Thus, the solution does not provide backward and forward secrecy as the attacker can gain the ID of the devices, then sniff other values from the current session to find the previous and future secret keys [33]. Also, this protocol does not provide anonymity and untraceability.

All the previous techniques are static. In the next section, we are going to discuss continuous authentication techniques.

B. CONTINUOUS AUTHENTICATION

Bamasag and Youcef-Toumi [16] presented a U2D continuous authentication protocol. This protocol employs Shamir's secret sharing [34] integrated with time-bound data, in order to continuously authenticate users. To perform the authentication process, both the verifier and the claimer have to register to a trusted authority and obtain their secret. The portions of this secret are referred to as shares and are generated using Shamir's secret key, these shares act as tokens, so the verifier can authenticate the claimer using them. Time is employed in this scheme such that every share is tied to a specific time frame and revealed accordingly. This protocol provides

secure communication when sending consecutive messages; however, the computation cost over a finite field is high and the storage cost on the claimer side is also high [35]. Additionally, it is a one-way authentication process, rather than a mutual one.

Nespoli *et al.* [36] proposed a framework for the IoT environment, that provides continuous authentication and authorization. This framework comprises three main ontologies: person, IoT device and location. The decisions are made based on two policies, i.e. authentication, and authorization policies. According to the user's authentication policy, he/she will attain the authorization policy to access devices and locations. Another framework for smart homes is presented by Ashibani *et al.* [18]. It provides continuous authentication based on contextual information. This contextual information is used to validate users continuously during the session. Based on doing so, it assigns access and security levels for the user. However, both frameworks [18] and [36] require large datasets to achieve higher accuracy.

Several research works provide U2D continuous authentication based on their users biometric (biometric authentication). Peris-Lopez *et al.* [17] presented an authentication scheme, that continuously verifies the user using ElectroCardioGramECG signals (ECG). They built their model to address three types of attackers: blind, known and unknown. This scheme lacks depth when dealing with different user states, such as walking, training, running, etc. and also fails to preserve the user's privacy. Another scheme, proposed by Yeh *et al.* [15], provides biometric authentication. This scheme provides continuous authentication based on measuring the user's plantar pressure. It utilizes a wearable device to collect foot pressure data and classify users with machine learning. Although this biometric offers a means to uniquely identify every user, it requires each user to wear customized shoes to deliver accurate results. Additionally, human noises affect the results. Moreover, Ekiz *et al.* [37] proposed a continuous authentication scheme based on heart rate variability (HRV) extracted features. The HRV is collected using a wrist-worn smartband to continuously authenticate users during the session, after the user enters his/her password or fingerprint. It prevents from session hijacking and spoofing attacks. However, this scheme needs a large amount of trained data to have accurate results. Additionally, the results can be affected by noises and also depend on the quality of the wrist band; for example, sport straps give better outcomes than other straps.

For D2D continuous authentication, we observed two works proposed by Wang *et al.* [38] and Chuang *et al.* [39]. Wang *et al.* [38] suggested a continuous authentication scheme that authenticates edge devices in the cloud server. It uses Electromagnetic radiation (EMR) to conduct continuous authentication between devices. We can conceptualize these EMR signals as unique device fingerprints for identification purposes. This mechanism uses an elliptic curve and the hash function to secure the authentication process. It is a one-way authentication process, such that the edge devices do not authenticate the cloud server. Noises can interfere with EMR signals generating incorrect outputs.

Chuang *et al.* [39] presented a D2D lightweight continuous authentication protocol, which employs the dynamic feature of IoT devices (battery) using a token technique. Continuous authentication happens over a specific time frame. Once this period ends, the devices need to relaunch a static authentication process. This scheme employs lightweight functions, such as: HMAC and hash to secure the authentication process. The performance in terms of computation costs for this protocol is relatively better than other techniques that use encryption. However, it does not provide backward secrecy, and nor does it prevent DoS attacks from happening. Furthermore, the secret key for the sensor remains the same, rendering it vulnerable in cases where an attacker gains the key using a memory dump or any other process [40]. It is also vulnerable to an impersonation attack.

Table 1, summarizes the advantages and limitations of the previously discussed schemes.

Discussing existing research around authentication schemes and realizing their limitations inspired us to look for areas of improvement within this field. Additionally, the work done in D2D continuous authentication has been limited. As a result, our aim is to design a D2D continuous authentication protocol, that overcomes the previous weaknesses, mitigates DoS and provides privacy as well as ensures security.

III. THE PROPOSED MODEL

The proposed protocol design comprises two major parts; the architectural design and the protocol design. In the architectural design, the authentication component resides on fog nodes, close to the end of the IoT devices, and can achieve real-time response and reduce latency. In the protocol design, we provide communication details.

A. ARCHITECTURAL DESIGN

In the architectural design, we focus on distributing an authentication system on fog nodes and close to the end IoT devices, to achieve a real-time response and reduce latency. We propose a three-layered architecture here, as shown in Figure 1. This is composed of the following layers:

1) Cloud layer: this layer comprises one or many servers. These servers support other layers for the registration of devices and the establishment of a secure communication. Additionally, it allows end users to register and communicate with the system.
2) Edge layer: the edge layer can be composed of several sub-layers for better distribution. It contains mainly the Home IoT server (HIoT), which is responsible for the region. Each HIoT manages the registration and authentication of gateways. All HIoT servers need to register to the cloud to communicate and authenticate one another. Additionally, this layer contains several gateways. The gateway is responsible for authenticating the sensors residing in its region using continuous authentication.

TABLE 1. Advantages/limitations of previous authentication schemes.

Reference	Type	Advantages	Limitations
[22]	Static	Scalable, prevents impersonation and replay attacks	Vulnerable to man-in-the-middle attacks, session hijacking
[24]	Static	Two-factor authentication, privacy	Single point failure, lacks mutual authentication and forward secrecy, no user auditability [26]
[25]	Static	User auditability	Communication and computation costs are high, vulnerable to replay attacks
[27]	Static	Scalable, efficient, secure communication, mitigate DoS attacks	High cost, no detection if Auths get compromise
[30]	Static	Secure localization, privacy, forward secrecy	Vulnerable to collision, man-in-the-middle and session attacks, powerful back-end server
[32]	Static	Lightweight, reasonable computation cost, prevents replay and impersonation attacks	Lacks backward/forward secrecy, needs secure secret key, no devices' privacy
[16]	Continuous	Efficient, scalable, forward/backward secrecy	High storage and computation costs, no mutual authentication, third party is needed
[36]	Continuous	Utilizes user's behaviour, various authorization level	Needs large training dataset, more processing time if users increase
[18]	Continuous	Different security levels, no need for users involvement	High rate of false positive, Needs large training dataset, vulnerable to insider attackers
[17]	Continuous	Real-time, considered different attacker models	Needs large dataset, lacks in preserving privacy, high computation costs
[15]	Continuous	The plantar pressure can not be forgotten or stolen and it is unique	Results are effected by noises, large dataset, special device to collect the plantar pressure
[37]	Continuous	Prevents spoofing attacks and session hijacking, no need for user intervention	Require large training dataset, the results depend on the quality of the smartband, the performance still unknown
[38]	Continuous	Prevents spoofing, replay, tempering attacks, EMR uniquely authenticate devices	EMR can be affected by noises, no mutual authentication, dynamic behaviour leads to fault results
[39]	Continuous	Distributed, lightweight, prevents session hijacking and replay attacks, preserves privacy	Vulnerable to impersonation and DoS attacks, no backward/perfect forward secrecy

3) End devices layer: this layer comprises of sensors and these sensors are organized in the form of clusters, each of which is managed by the corresponding gateway.

As shown in Figure 1, the architecture is composed of several networks, each managed by the HIoT server. Inside each network there are several clusters each managed by a gateway. Notice that the authentication is handled by the gateway with the support of HIoT and relies on the sensor location. Sensors are allowed to move within the cluster or from one HIoT to another. For this, we define two movement policies:

1) Inter-cluster movement: if the sensor moves from one cluster to another under the same HIoT, the authentication is handled by the new hosting gateway.
2) Inter-network movement: if the sensor moves from one HIoT to another, the authentication is handled by the hosting gateway with the support of the original HIoT.

B. COMMUNICATION PROTOCOL

In the protocol design, we provide a continuous authentication protocol to mitigate DoS attacks. This protocol is principally based on [39] and [30]. Additionally, we added contextual information of the device; i.e., the location beside the battery and token proposed in [39], as a way to continuously authenticate devices during the session when sending sensed data. We also took into consideration the sensors' movement, anonymity and untraceability.

The proposed protocol is based on a two authentication process that is both static and continuous:

- **Static authentication**: In static authentication the gateway and sensors mutually authenticate each other at the beginning of each session.
- **Continuous authentication**: Continuous authentication is performed during the authentication period after which the device needs to relaunch new static authentication. Unlike static authentication, continuous

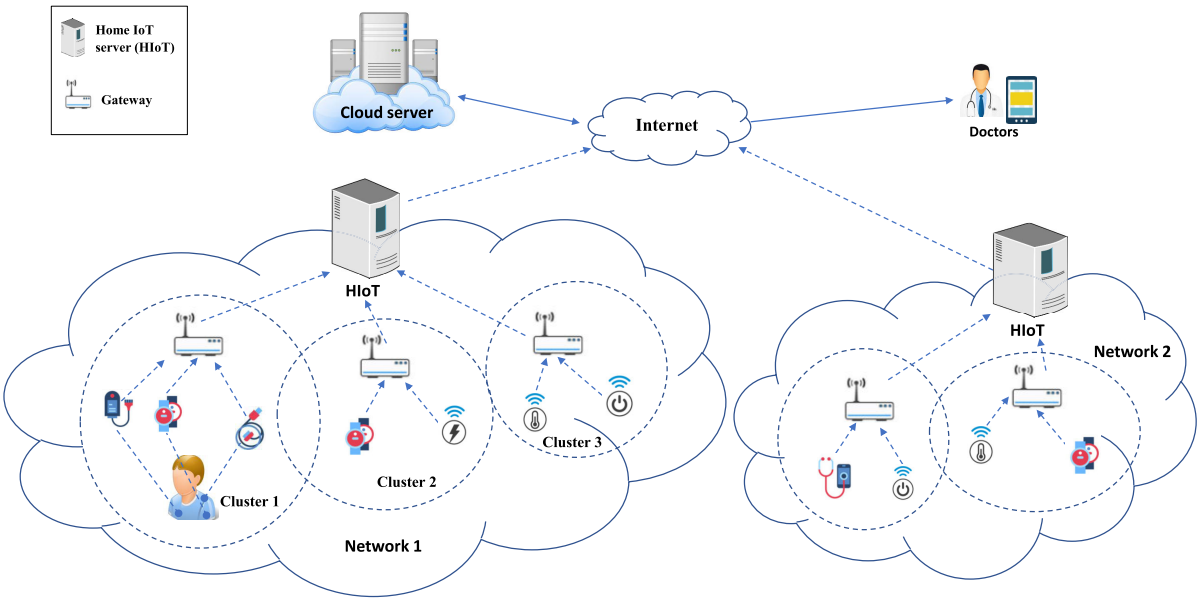


FIGURE 1. Architectural design example.

authentication is performed during the session using a token, and devices' contextual information (e.g., location and battery capacity).

The protocol consists of three main phases as shown in Figure 2: initialization, static, and continuous authentication.

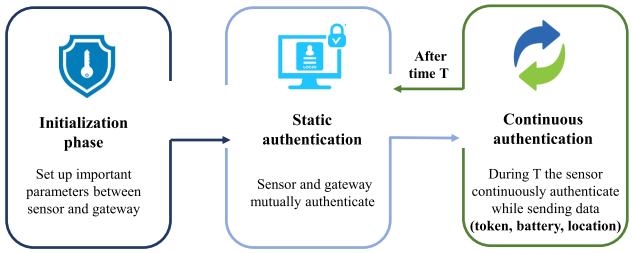


FIGURE 2. Protocol phases.

Our protocol follows the assumptions set out below:

1) Each sensor has one or more batteries.
2) Each sensor can be precisely localized inside and outside buildings.
3) Each sensor can perform lightweight computations, such as: generating random numbers and performing hash functions.
4) Sensors are of type class-2 and beyond to be able to establish secure communication with a gateway.[1]
5) The gateways and HIoT servers are secure and scalable.
6) The gateways and HIoT servers have sufficient space to store information related to the sensors they manage.

In addition to the support for continuous authentication, the proposed protocol takes sensor movement into account. Indeed, it allows for the authentication of moving sensors that transit from one cluster to another, or from one network to another. Moreover, the protocol provides a robust mechanism, as a means to recover after a synchronization loss that might occur following a DoS attack. The notations used in the sequel are declared in Table 2.

We discuss in the following the authentication phases namely initialization, static and continuous authentication.

1) INITIALIZATION PHASE

In this phase, both sensor and gateway need to set up some important parameters through a secure channel as illustrated in Figure 3. The steps of this phase are discussed in what follows.

1) The sensor sends its own parameters. This phase starts with the sensor sending its ID, battery, and location information to the gateway.
2) The gateway receives sensor's parameters and performs the following steps:
   - Once the gateway receives the request, it checks if the sensor's location is within its range, otherwise, the gateway terminates the request. Then the gateway stores the sensor's location as the initial location (IL), defines the sensor movement model (see section V) and stores a secure range (SR) for that sensor.
   - After that, the gateway generates a secret value SK_{SN}. Furthermore, it calculates the estimated

[1]Sensors have several classes, which are classified according to their processing and memory capabilities. Classes 0 and 1 are restricted in memory and code which makes the establishment of a secure communication in the initialization phase hard [41]. However, it is worth noting that there are some solutions to overcome this problem [42].

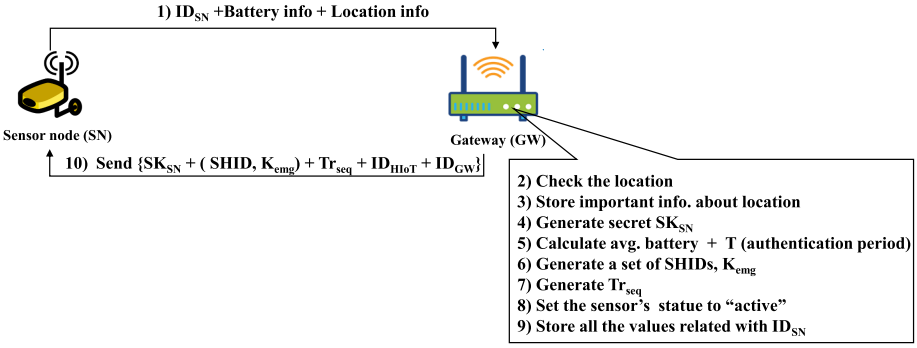


FIGURE 3. Initialization phase.

daily average battery (EBC) based on the information received from the sensor [39].

- The gateway also generates a set of unlinkable values, called shadow-IDs ($SHID$), each has a corresponding K_{emg}. This set of ($SHID$, K_{emg}) pairs is sent to the sensor to be used in the case of losing synchronization (see section IV-A).
- Another important step handled by the gateway is the generation of a random track sequence number (Tr_{seq}). The Tr_{seq} allows the prevention of replay attacks. It also speeds up the authentication and is used to negotiate a new SK_{SN} after a successful static authentication phase.
- The gateway stores the most recent Tr_{seq} and sends it to the sensor. It stores in its database all the values related to the sensor with its ID and sets the status of the sensor to "**active**" which allows the gateway to know the sensor is alive.
  Each sensor may be either active or passive. In the passive case, all the sensor's related information is deleted because the status remains unchanged for a predefined period of time. The status flag has the advantage of efficiently managing the memory space of the gateway.
- The gateway defines and stores an authentication period (T) for each sensor, so after this period, the sensor needs to relaunch a new static authentication session.

3) The gateway sends back important parameters to the sensor. At the end, it will send to the sensor via a secure channel {SK_{SN}, Tr_{seq}, ID_{HIoT}, ID_{GW}, set of ($SHID$, K_{emg})}. The ID_{HIoT} is the HIoT ID and the ID_{GW} is the gateway ID, that the sensor is connected to.

In this phase, the gateway has to send the previous information to the HIoT server so that they are stored remotely. This step is crucial to correctly handle the authentication process in case the sensor moves under another cluster or HIoT.

2) STATIC AUTHENTICATION PHASE

After the initialization phase, the static authentication phase comes. In this phase both the gateway and the sensor mutually authenticate each other. The gateway sends a new Tr_{seq} to the sensor in order to negotiate a new SK_{SN}. Notice that generating a new secret key SK_{SN} frequently prevents from a memory dump attack that may allow the attacker to steal the secret key and provides backward secrecy. Additionally, they both gateway and sensor negotiate a token, which is used in the continuous authentication phase during T.

The gateway retrieves the initial location (IL) and secure range (SR), to compute the authorized movement ($AMvt$), which is used to check whether the sensor resides into its allowed location. Moreover, the gateway computes the estimated remaining battery capacity threshold (BCT) [39], also used to check the correctness of the remaining battery capacity sent by the sensor and thus better enforcing the security. The authentication steps are presented in Figure 4.

The communication between the sensor and gateway proceeds as follows.

a: SENSOR MESSAGE PREPARATION

The sensor message preparation contains three main steps, in order to send MSG_1 to the gateway to start the static phase:

1) First the sensor generates a random number v, gets the value of the current battery cb plus the current location value cl, and extracts SK_{SN}, Tr_{seq} from its storage to mask the battery and location values as follows:
   - $mb = cb \oplus H(SK_{SN} \oplus Tr_{seq})$
   - $ml = cl \oplus H(SK_{SN} \oplus Tr_{seq})$
2) Then the sensor computes the value $X = v \oplus H(cb||cl)$, in order to mask v since it is used to negotiate the token for the next phase and thus should be securely exchanged. It also computes $M_1 = H[(v||ID_{SN}) \oplus H(SK_{SN})]$. At the end, it computes AID to preserve the privacy of the sensor, and M_2 to ensure the integrity of the message while transmission, as follows:
   - $AID_{SN} = H(ID_{SN}||SK_{SN}||Tr_{seq})$
   - $M_2 = HMAC_{SK_{SN}}[AID_{SN}, X, M_1, mb, ml, ID_{GW}, ID_{HIoT}]$
3) Finally, the sensor sends the resulting message $MSG_1(AID_{SN}, mb, ml, X, M1, M2, ID_{GW}, ID_{HIoT})$ that contains all the values required for authentication to the gateway.

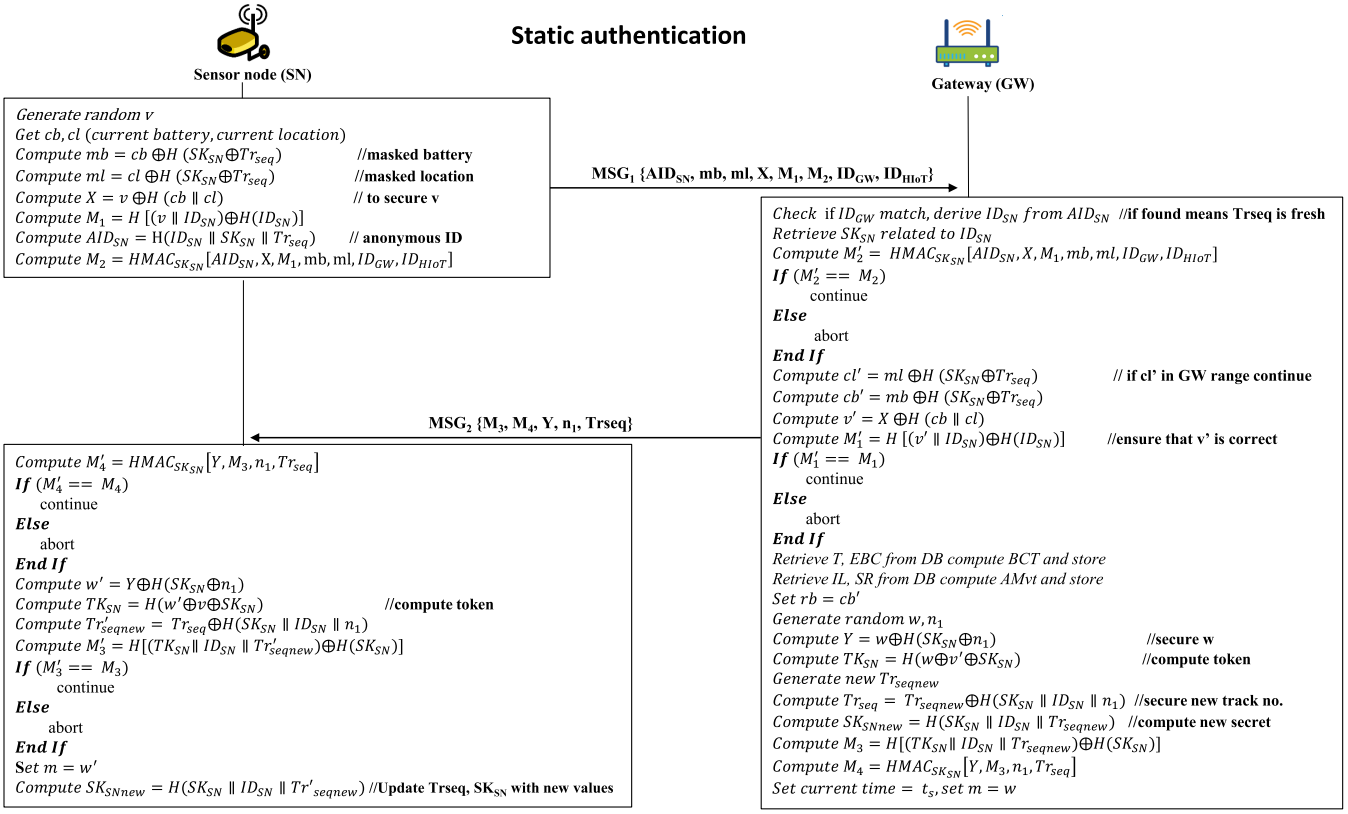


FIGURE 4. Static authentication phase.

b: GATEWAY MESSAGE RECEPTION

Once the gateway receives the message MSG_1 from the sensor it performs the following steps:

1) First, it checks ID_{GW} to make sure that the sensor is within the cluster.
2) The gateway derives the ID from the AID of the sensor by generating $AID'_{SN} = H(ID_{SN}||SK_{SN}||Tr_{seq})$ for the identities of the stored sensors. If there is a match, then the Tr_{seq} is recent and the authentication process will continue, otherwise the session is aborted. [2]
3) The gateway retrieves the corresponding SK_{SN} and computes $M'_2 = HMAC_{SK_{SN}}[AID_{SN}, X, M_1, mb, ml, ID_{GW}, ID_{HIoT}]$, to check the message integrity.
4) The gateway also computes the sensor current location $cl' = ml \oplus H(SK_{SN} \oplus Tr_{seq})$ and checks if it is within its range, in order to prevent any outsider malicious sensor from entering and pretending to be in the system.
5) Similarly, the gateway computes $cb' = mb \oplus H(SK_{SN} \oplus Tr_{seq})$, the resulting values cb',and cl' are used to compute $v' = X \oplus H(cb'||cl')$. Then M'_1 is computed, if it is equal to the received message M_1, then all the values (cb', cl', v') are correct.

[2]Note that in this step the Tr_{seq} will speed the authentication process since if it's not recent the session will be aborted and no need to continue.

c: GATEWAY MESSAGE PREPARATION

After verifying all these values, the gateway retrieves EBC, T, IL and SR from its database. EBC and T are required to compute the BCT for the sensor [39]. Additionally, IL and SR are required to compute the authorized movement for the sensor ($AMvt$). The gateway then stores the BCT, $AMvt$ to be used later in the continuous authentication phase. The gateway sends MSG_2 to the sensor to mutually authenticate each other and to negotiate the token, the steps for preparing MSG_2 are:

1) The gateway assigns $rb = cb'$, generates two random numbers n_1 and w, then masks w to be secure, because it is used in token negotiation. It calculates $Y = w \oplus H(SK_{SN} \oplus n_1)$ and computes the token $TK_{SN} = H(w \oplus v' \oplus SK_{SN})$.
2) The gateway generates a new $Tr_{seq_{new}}$ to be used in the next authentication process and to compute a new secret key $SK_{SN_{new}}$. This new track number will be masked to be hidden during transmission, $Tr_{seq} = Tr_{seq_{new}} \oplus H(SK_{SN}||ID_{SN}||n_1)$. The gateway computes the new secret key $SK_{SN_{new}}$, M_3, and M_4 as follows:
   - $SK_{SN_{new}} = H(SK_{SN}||ID_{SN}||Tr_{seq_{new}})$
   - $M_3 = H[(TK_{SN}||ID_{SN}||Tr_{seq_{new}}) \oplus H(SK_{SN})]$
   - $M_4 = HMAC_{SK_{SN}}[Y, M_3, n_1, Tr_{seq}]$
3) Finally, it sets the current time to t_s and m to w.

TABLE 2. Notations.

Variables	
ID_{SN}	Identity of sensor node
ID_{GW}	Identity of gateway
ID_{HIoT}	Identity of home server HIoT
AID_{SN}	Anonymous identity of sensor node
$SHID$	Shadow identity, used in synchronisation loss
SK_{SN}	The shared secret key between sensor and gateway
K_{emg}	Emergency key generated by gateway for synchronisation loss
TK_{SN}	Token negotiated between sensor and gateway for continuous authentication
T	Authentication period, used to continuously authenticate the sensor
t_c, t_s	Timestamps
cb	Current battery of the sensor
EBC	Estimated daily average battery of the sensor
BCT	Estimated remaining battery capacity threshold
rb	Remaining sensor's battery life after last authentication session
cl	Current location of the sensor
IL	The initial location of the sensor
SR	Secure range for the sensor
$AMvt$	Authorized movement that the sensor is allowed to move
sd	Data sensed by the sensor
mb	masked value of the current battery
ml	masked value of the current location
ms	masked value of the current sensed data
X, Y	Variables used to mask secure values
ACK	Acknowledgment
M_1,M_2,M_3,M_4, M_5,Y_1	Messages send between sensor and gateway
m	Intermediate variable used to store random numbers generated by the gateway
Tr_{seq}	Track sequence number generated randomly by gateway
v, r_2	Random numbers generated by the sensor
w, n_1, n_2	Random numbers generated by the gateway
Operators	
$\|\|$	Concatenation operation
$\oplus$	Exclusive-OR operation
Operations	
H	Hash function
$HMAC_K$	Hash function using key K

Once previous values are calculated, the gateway sends the message $MSG_2 = \{M_3, M_4, Y, n_1, Tr_{seq}\}$ that contains all the values to the sensor node to negotiate the token, new secret and update the track number.

d: SENSOR MESSAGE RECEPTION

Once the sensor receives MSG_2, it proceeds as follows:

1) The sensor uses SK_{SN} and computes $M_4' = HMAC_{SK_{SN}}[Y, M_3, n_1, Tr_{seq}]$ to ensure the integrity of the received message, if it is equal to the received M_4, then the values are not changed. After that, the sensor computes: $w' = Y \oplus H(SK_{SN} \oplus n_1)$, to compute the token $TK_{SN} = H(w' \oplus v \oplus SK_{SN})$.
2) The sensor updates Tr_{seq} with the new value by computing $Tr'_{seq_{new}} = Tr_{seq} \oplus H(SK_{SN}||ID_{SN}||n_1)$ and computes M_3'. If it is equivalent to the received M_3, then the computed TK_{SN} and new $Tr'_{seq_{new}}$ are correct. As a result, the authentication process is successful.
3) At the end, the sensor sets $m = w'$, updates SK_{SN} with the new value $SK_{SN_{new}} = H(SK_{SN}||ID_{SN}||Tr'_{seq_{new}})$ and stores TK_{SN} to be used in the continuous authentication phase.

The continuous authentication process is illustrated in the next section.

3) CONTINUOUS AUTHENTICATION PHASE

After a successful static authentication phase, the continuous authentication is be performed during a period time T. We recall that at the end of the static authentication the sensor has stored the values TK_{SN}, BCT and $AMvt$ to be used in this phase during authentication time T.

First, the sensor sends the sensed data plus the verification values to the gateway. Then, the gateway derives the sensor's ID from the AID and checks the validity of T as well as the reasonability of cb and cl. Finally, it sends an acknowledgement ACK back to the sensor based on T, either to relaunch a new static phase or to continue sending data.

The continuous authentication phase is illustrated in figure 5 and proceeds as follows.

a: SENSOR MESSAGE PREPARATION

In order to start the continuous authentication phase, the sensor sends the message MSG_{C1} to the gateway. The preparation of this message is performed as follows:

1) First, the sensor generates a new random number r_2 and extracts both recent values of the location and the battery cl, cb. Then, it calculates their respective masked values mb and ml to be sent securely by using the stored TK_{SN} as follows:
   - $mb = cb \oplus H[TK_{SN}||(m \oplus r_2)]$
   - $ml = cl \oplus H[TK_{SN}||(m \oplus r_2)]$
2) The sensor also gets the sensed data sd and masks it $ms = sd \oplus H((TK_{SN} \oplus m)||r_2)$, m being a value calculated during the static authentication phase and stored in both gateway and sensor that is continuously changed during the continuous authentication phase.
3) Next, the sensor computes the AID to preserve privacy and M_5 to ensure message integrity as follows:
   - $AID_{SN} = H(ID_{SN}||TK_{SN}||r_2)$
   - $M_5 = HMAC_{TK_{SN}}[AID_{SN}, mb, ml, ms, r_2]$

Finally, the sensor sends the message MSG_{C1} to the gateway.

b: GATEWAY MESSAGE RECEPTION

Once the gateway receives MSG_{C1}, the following steps are performed:

1) The gateway derives the ID from the sensor's AID to get TK_{SN}, BCT and $AMvt$ from its database.
2) It sets t_c = current timestamp, then computes $(t_c - t_s)$ to decide whether the static authentication should be

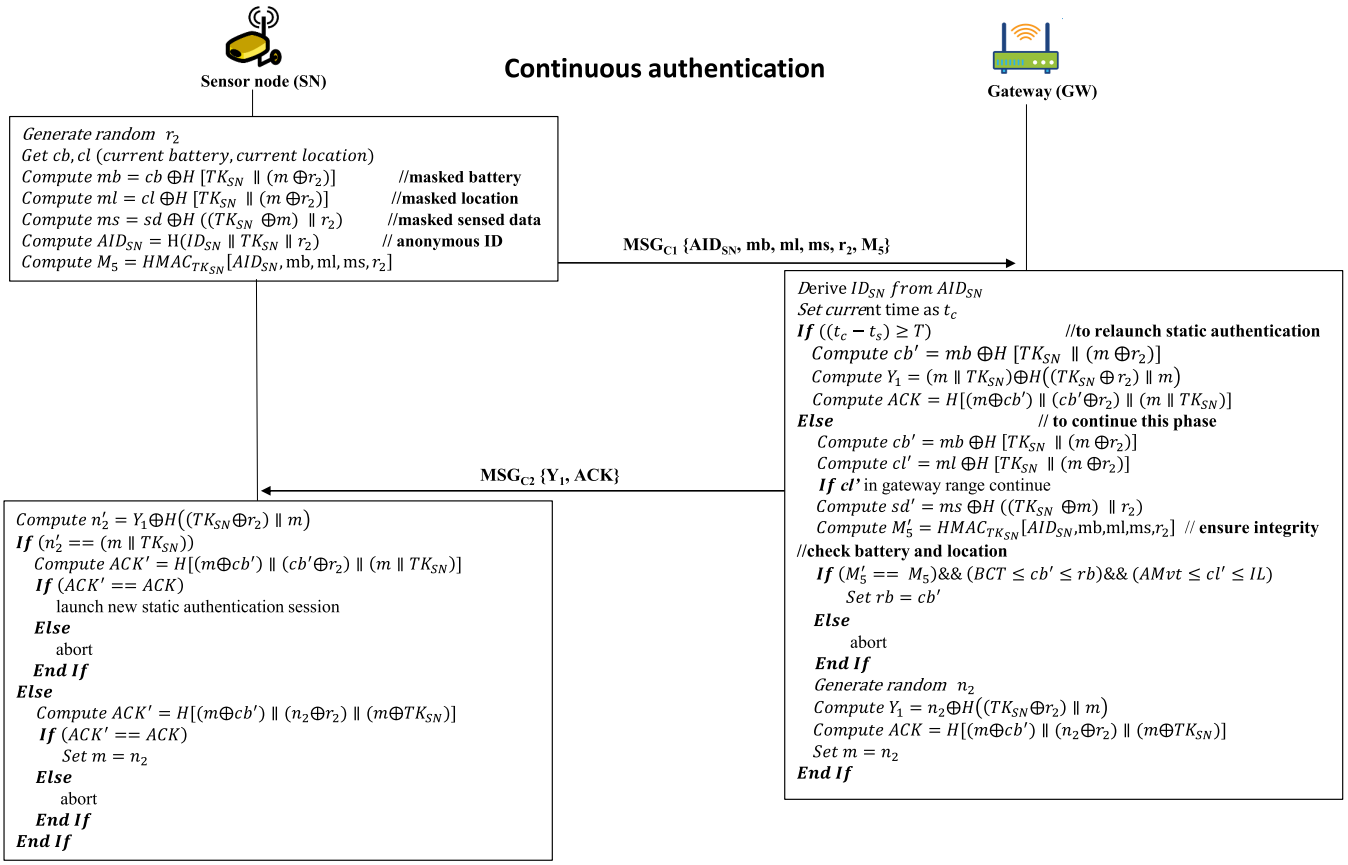


FIGURE 5. Continuous authentication phase.

relaunched (i.e. the result is greater than T) or not. There are two cases:

- If $(t_c - t_s > T)$, then the gateway computes $cb' = mb \oplus H[TK_{SN}||(m \oplus r_2)]$, as well as $Y_1 = (m||TK_{SN}) \oplus H[(TK_{SN} \oplus r_2)||m]$ and $ACK = H[(m \oplus cb')||(cb' \oplus r_2)||(m||TK_{SN})]$.
  After that, Y_1 and ACK are sent to the sensor to relaunch the static authentication.
- Otherwise, if $(t_c - t_s)$ is still in the T range, then the gateway does the following:
  First, it computes cb' similarly, as well as $cl' = ml \oplus H[TK_{SN}||(m \oplus r_2)]$ and $sd' = ms \oplus H((TK_{SN} \oplus m)||r_2)$. After computing cl' the gateway checks if the value is in a valid range, otherwise it terminates the session.
  Second, it verifies and checks the following three values, if any of them is false then session will be aborted:
  - $M_5' = HMAC_{TK_{SN}}[AID_{SN}, mb, ml, ms, r_2]$ and should be equal to M_5.
  - The remaining battery value cb' should be reasonable (i.e. $(BCT \leq cb' \leq rb)$).
  - The location cl' should be allowed by the gateway i.e. between the initial location and the authorized movement $(AMvt \leq cl' \leq IL)$.

  Third, if the previous conditions are satisfied, then the sensor is authenticated. As a result, the gateway updates the value of $rb = cb'$ to be used for the next battery check and generates a random number n_2, to be used for computing Y_1 and ACK as follows:
  - $Y_1 = n_2 \oplus H((TK_{SN} \oplus r_2)||m)$
  - $ACK = H[(m \oplus cb')||(n_2 \oplus r_2)||(m \oplus TK_{SN})]$

At the end, before sending MSG_{C2} the gateway sets $m = n_2$.

c: SENSOR MESSAGE RECEPTION

When the senor receives MSG_{C2}, it calculates $n_2' = Y_1 \oplus H((TK_{SN} \oplus r_2)||m))$. If it results to $\{m||TK_{SN}\}$ i.e. the continuous phase has expired, then the sensor relaunches a new static authentication phase. Otherwise, it sets m to n_2' and the continuous authentication phase is maintained. In both case, the ACK should be verified.

IV. MITIGATING THE DOS ATTACK

The main goal of a DoS attack is to violate the availability of the system. In IoT environments, a DoS attack can be performed in several ways (jamming, flooding, desynchronization, etc.) [43]. Additionally, the advent of IoT technology renders devices vulnerable to DoS attacks, according to the 2020 McAfee report, DoS is ranked among the top ten

attacks [44]. In this section, we propose two cases to mitigate DoS attack, while continuously authenticating users. The first is conceived to face attacks arising from malicious behavior, which lead to synchronization loss between the gateway and the sensor. The second is to address gateway unavailability, so that it cannot provide its services to the sensors it governs.

A. RECOVERING FROM SYNCHRONIZATION LOSS

In this protocol, we take into consideration the synchronization loss since we need to update the key between the sensor and gateway after each successful authentication phase and ensure forward secrecy. In this case, the gateway might update the key while the sensor doesn't. As a result, a DoS attack may occur due to the loss in synchronization between the sensor and gateway [45].

One of the system goals is to protect the system from DoS attacks. That's why we intend to use a set of emergency key and pseudo-identity pairs proposed in [30] to overcome this problem. In our protocol, this pseudo-identity is known as Shadow IDs (*SHID*) [22]. These are unlinkable IDs that are generated in the gateway and sent to the sensor in the initialization phase. The emergency key (K_{emg}) is used to replace the secret key if there is any loss in communication.

Each ($SHID$, K_{emg}) is a one-time identity. It is used when the sensor loses synchronization with the gateway and does not receive a new track sequence number. The goal of the ($SHID$, K_{emg}) is to prevent the DoS attack that can happen when we have key-update. However, additional steps should be performed when makes the gateway unavailable wherein sensors can be authenticated by the nearest gateway [29].

In the static authentication phase (see Section III-B2), the gateway generates a new Tr_{seq} after each successful authentication process, in order to negotiate a new secret key between the sensor and the gateway. If there is a connection loss between the parties, the gateway will update with new values, while the sensor cannot. For this reason, we use the shadow IDs and emergency keys [30] to inform the gateway that there is a problem (i.e., the sensor does not receive these new values). This pair is used only once then both the sensor and gateway must delete the used pair. Figure 6 illustrates the steps of recovery from synchronization loss and proceeds as follows.

1) SENSOR MESSAGE PREPARATION

First, if the sensor has not received MSG_2 for a specific period (defined by admin), it will use one of the pairs of ($SHID$, K_{emg}) from the set that is stored in the sensor and was generated by the gateway in the initialization phase. Then, the sensor assigns $AID = SHID$ and computes $M_1 = HMAC_{K_{emg}}[AID_{SN}, ID_{GW}, ID_{HIoT}]$. Then the sensor sends MSG_{1R} to the gateway.

2) GATEWAY MESSAGE RECEPTION

Once the gateway receives MSG_{1R}, it checks whether the ID_{GW} matches its own. If yes, then it derives the ID of the SN from AID. If the gateway finds that there is a match with one of the SHID, i.e. then it deduces the sensor has lost the communication. Thus, the gateway retrieves the K_{emg} that is related to this SHID. Next, it computes $M_1' = HMAC_{K_{emg}}[AID_{SN}, ID_{GW}, ID_{HIoT}]$. If it matches M_1, then the message has not been changed during transmission.

3) GATEWAY MESSAGE PREPARATION

Then, the gateway generates random n_1, generates a new $Tr_{seq_{new}}$ and masks this new track number as the following, $Tr_{seq} = Tr_{seq_{new}} \oplus H(K_{emg}||ID_{SN}||n_1)$. After that it computes the new secret key as $SK_{SN_{new}} = H(K_{emg}||ID_{SN}||Tr_{seq_{new}})$, and computes M_2 and M_3, $M_2 = H\ [(K_{emg}||ID_{SN}||Tr_{seq_{new}}) \oplus H(K_{emg})]$, $M_3 = HMAC_{K_{emg}}[M_2, n_1, Tr_{seq}]$. Finally, the gateway sends MSG_{2R} to the sensor.

4) SENSOR MESSAGE RECEPTION

Once the sensor receives MSG_{2R}, it computes $M_3' = HMAC_{K_{emg}}[M_2, n_1, Tr_{seq}]$ and compares it with the received M_3 to check the integrity of the message. Then, the sensor extracts the new $Tr_{seq_{new}}$ from the masked value by computing: $Tr'_{seq_{new}} = Tr_{seq} \oplus H(K_{emg}||ID_{SN}||n_1)$. Next, the sensor computes $M_2' = H\ [(K_{emg}||ID_{SN}||Tr'_{seq_{new}}) \oplus H(K_{emg})]$ and compares it with the received M_2 to ensure the correctness of $Tr_{seq_{new}}$. Afterwards, the sensor computes the new secret key $SK_{SN_{new}}$ in the same way as the gateway. Finally, the sensor updates the values and performs a new static authentication phase with these new values. Notice that since ($SHID$, K_{emg}) are used only once, if the sensor needs another set of shadows then it can send a request to the gateway for a new set [45].

B. GATEWAY FAILURE

In this section, we present our solution for dealing with gateway unavailability consisting of authenticating sensors by the nearest gateway.

Figure 7 identifies the steps in the recovery process. If gateway (GW1) fails due to DoS, the sensors it governs will be authenticated via the nearest gateway (GW2). GW2 requires access to all the information related to these sensors from the HIoT. Certain constraints need to be taken into consideration here:

- The sensors covered by the failed gateway should be in the wireless coverage range of the recovery gateway.
- If more than one recovery gateway is available, then the selection decision rests on several factors (i.e., nearest, less cost, less overloaded).

Next, we describe the recovery process in detail. First we explain how sensors can detect failure. Then, we discuss the recovery process.

1) FAILURE DETECTION

During the period when the gateway is unavailable, the sensors can detect failure via the following process:

1) First, the sensor sends a regular request MSG_1, to start a static authentication.
2) Then, it waits for a specific period (defined by admin). If it does not receive a response from the gateway,

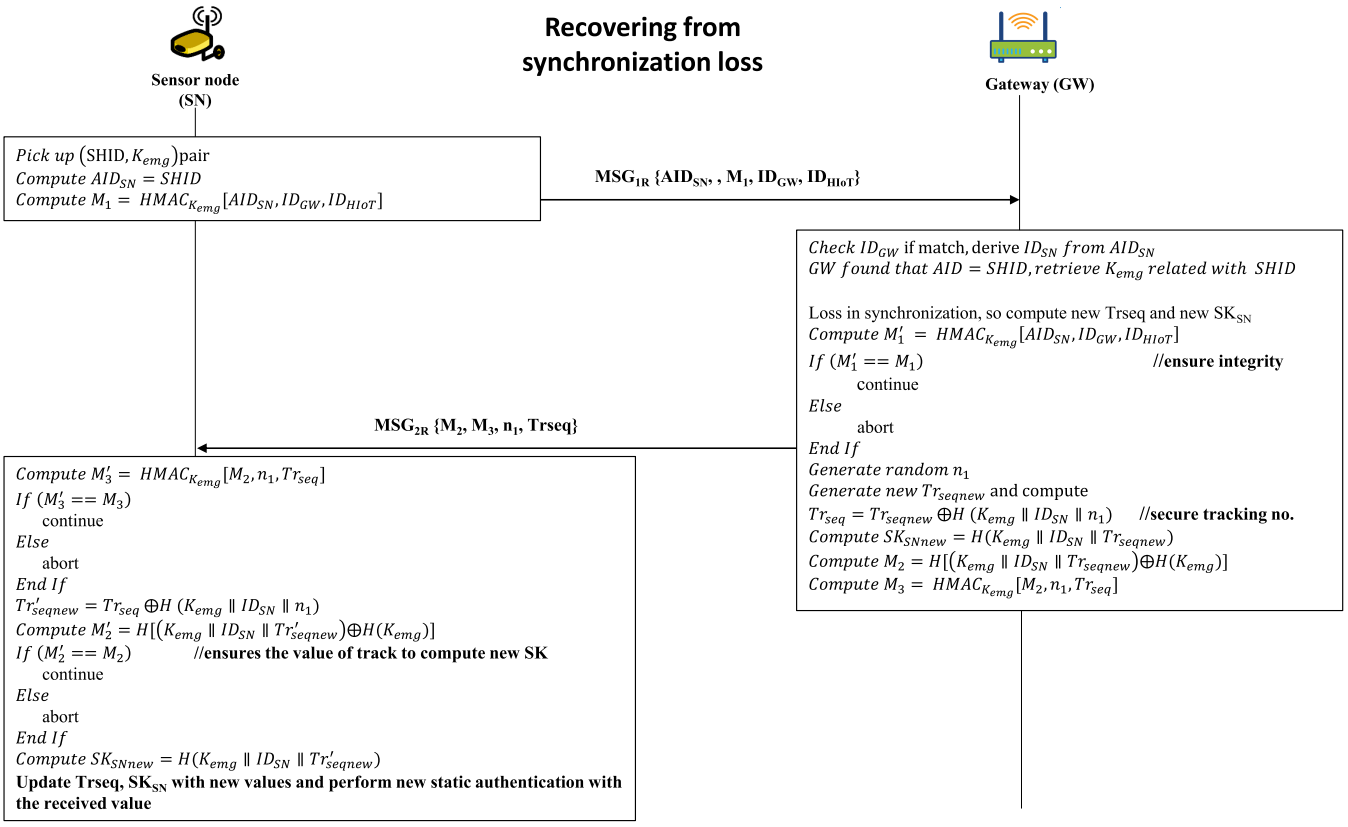


FIGURE 6. Recovering from synchronization loss.

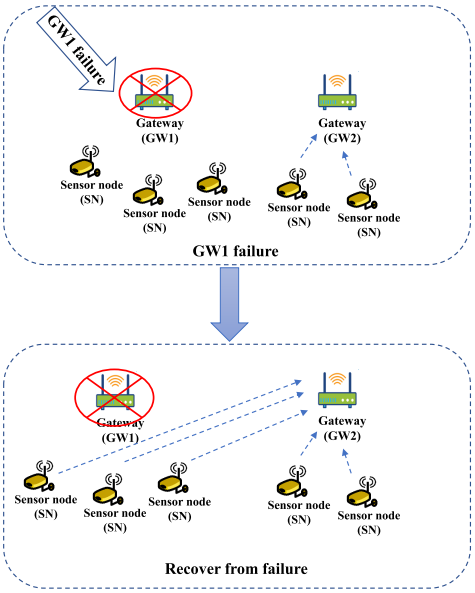


FIGURE 7. Recover from gateway failure.

it sends a pair of (*SHID*, K_{emg}) (using the same steps in the recovery process).

3) If the sensor still receives no response from the gateway. It sends another pair. After several consecutive (*SHID*, K_{emg}), the sensor will detect there is a problem with the related gateway.

At this point, the sensor then begins broadcasting to look for the nearest gateway to begin the recovery process.

2) RECOVERY PROCESS

After sensors have detected a failure and found the nearest gateway, they can be authenticated via the recovery gateway, with assistance from the HIoT server. Figure 8 illustrates the steps in failure detection by SN_R, and the procedure for communication with gateway GW2 with help from the HIoT. However, we need to consider the movement model for each sensor, since we need to consider two types: static and dynamic. Static sensors are not allowed to move, while dynamic sensors can move and have three categories: intra-cluster(move inside the cluster), intra-HIoT (move between gateways inside the HIoT), and intra-network (move between different networks). For further information about these movement models, check section V.

As shown in step 3 Figure 8, the SN_R sends an authentication request to GW2. Then, the GW2 determines that if the received ID_{GWRec} does not match its own, then it should check the received $ID_{HIoTRec}$. If the HIoT ID matches that to which it is related, it asks the HIoT server for SN_R information to authenticate it.

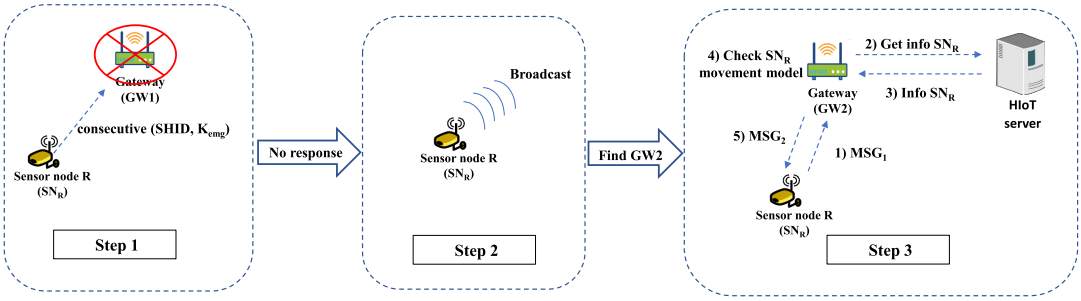


FIGURE 8. Recover process steps.

Once GW2 receives all the information related to the sensor, it performs the following steps:

- First, it checks the sensor's model movement, which is specified and stored in the initialization phase for each sensor. For each movement model, there are two types: static and dynamic. Both will be explained.
  1) If the sensor is static, then the gateway checks the initial location (IL), which is specified in the initialization phase. It can be compared with the received location (ml). If both values are the same, this ensures the gateway can authenticate the sensor securely and continue the process.
  2) If the sensor is dynamic, we then consider two sub-cases:
     a) Intra-cluster sensor: in this case, the gateway checks the secure range of the sensor (SR). The SR is specified in the initialization phase. If the received masked location ml, is within SR, then the gateway can authenticate the sensor securely.
     b) Intra-HIoT and Intra-network: for this case, the gateway authenticates directly. As these sensors are allowed to move freely between clusters, the authentication process is similar to that for the inter-cluster movement (check section V-A).
- After this, both can start a continuous authentication process during time T until the original gateway recovers.

V. SENSOR MOVEMENT

Since the IoT devices can move easily, it is vital to define the specific procedures that handle this feature by re-authenticating devices when they move from one area to another.

Our architectural design allows for two main types of movement:

- Inter-cluster movement: refers to the movement of sensor from one cluster to another, and between different gateways under the same HIoT.
- Inter-network: refers to movement between different HIoTs.

To manage the movement of sensors in a flexible way, we assume a sensor can be either *static* or *dynamic*. A static sensor is not allowed to move. This applies to certain IoT medical devices such as room thermostats and humidity controllers. As for dynamic sensors, we can define the following movement models:

- Intra-cluster movement model: the sensor is allowed to move within a specific gateway and in a defined secure range (SR). For instance, this model applies for the sensors within a surgical robot and sensors determining anesthetic levels during surgery.
- Intra-HIoT movement model: here the sensor can move between different clusters in a specific HIoT within a defined SR. This may be the case with smart beds that contain pressure and heartbeat sensors, designed to monitor the patient's condition.
- Intra-network movement model: the sensor is free to move between different clusters and between different HIoTs. For example smart badges that assist in locating medical personnel and in providing balanced resource allocation around the hospital.

A. INTER-CLUSTER MOVEMENT

Both intra-HIoT and intra-network models allow sensors to move between clusters. In both instances, sensors are handled with the same HIoT. Thus, it is the responsibility of the current gateway to authenticate a moving sensor with support from the corresponding HIoT.

Figure 9 provides an example of an inter-cluster movement model. Here, sensor (SN_R) moves from one cluster to another under the same HIoT. Thus, SN_R has to communicate with the new gateway to become authenticated. The authentication procedure is then conducted according to the following steps:

1) The gateway checks the received ID_{GWRec}. If these do not match its own, then it checks the received $ID_{HIoTRec}$. Since both gateways come under the same HIoT, the match passes.
2) The gateway sends a request to the HIoT that contains the AID of SN_R plus the ID_{GW}.
3) The HIoT checks the ID_{GW} and searches for sensors under a specific gateway, then extracts the SN_R ID from

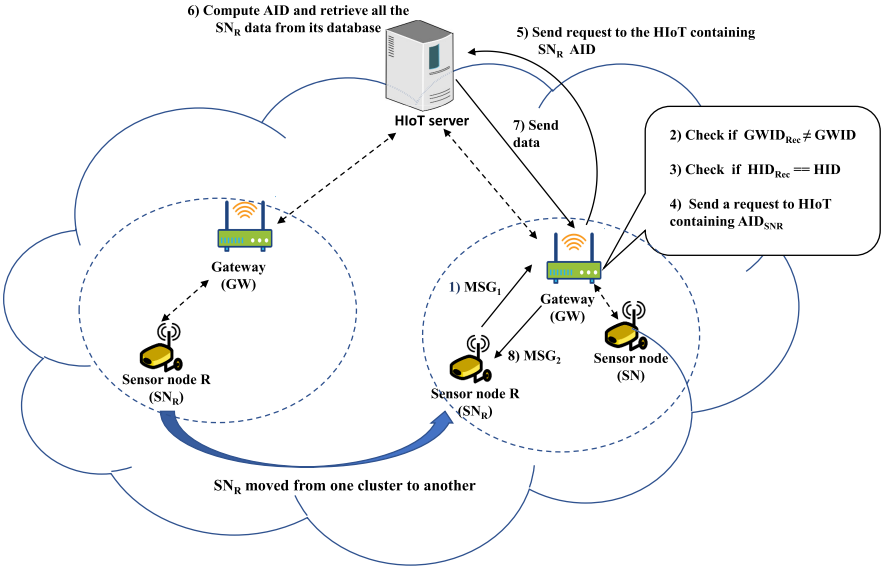


FIGURE 9. Inter-cluster movement.

the *AID*, and retrieves all the related data to return it to the gateway to continue the authentication process.

4) The gateway updates the initial location (IL), with the masked location *ml* received from the SN_R in MSG_1.
5) The gateway sends MSG_2 and commences the continuous authentication phase.

B. INTER-NETWORK MOVEMENT

The inter-network movement allows the sensor to move freely between different gateways belonging to different HIoTs. This movement is permitted only for sensors with an intra-network model. The visited gateway then has the responsibility of authenticating the moving sensor with the assistance of the original HIoT. This process is illustrated in Figure 10.

In this figure, the sensor SN_R is moving from one network to another (from building 1 to building 2) between the different HIoTs. Consequently, the gateway in building 2 is responsible for the authentication process. The process proceeds as follows. First, the new gateway sends a request to the original sensor HIoT to bring its related data. Once SN_R sends MSG_1 to the gateway, the latter checks to establish whether the sensor gateway and HIoT identifiers align with its own values. If they do not match, then the gateway sends a request to HIoT 2. This request contains $\{SN_RAID, ID_{GW}, ID_{HIoT}\}$. Note that if the sensor's movement model is not on the intra-network, then the gateway rejects the connection and terminates the session, because the sensor is not authorized to move between different HIoTs. When HIoT 2 receives the sensor request and determines that the sensor belongs to HIoT 1, it sends a request containing $\{ID_{GW}, SN_R, AID\}$ to the corresponding HIoT here 1. Once HIoT 1 receives the request, it fetches the sensor related information from the specific gateway and extracts the ID from AID, and then sends the related data to HIoT 2. The latter then forwards this response to its gateway. Finally, the gateway performs the required computations, sends MSG_2 to SN_R and implements the continuous authentication phase.

VI. SECURITY AND PERFORMANCE ANALYSIS

This section evaluates the security and performance of the proposed protocol. The security analysis is conducted to demonstrate that this protocol is robust and able to prevent security threats. We used Scyther to verify the robustness of the protocol. Additionally, we evaluated performance efficiency in terms of communication and computation cost.

A. INFORMAL SECURITY ANALYSIS

In this section, we provide an informal security analysis of the proposed protocol. We discuss several functionalities, such as mutual authentication, anonymity, forward/backward secrecy, etc. and illustrate how they are supported by the proposed protocol. Furthermore, we prove that our protocol is robust against multiple types of attacks.

1) MUTUAL AUTHENTICATION

Mutual authentication implies both communication parties must authenticate each other. This functionality is essential to deliver security to any IoT system [46].

In the static phase, the gateway validates the sensors by verifying values AID_{SN} and M_2. If the gateway computes M_2' and established that it is equivalent to the received M_2, then the sensor is deemed valid. Additionally, if the gateway derives the sensor's ID from the received AID_{SN}, it then also validates the sensor. Both the sensor and gateway verify that they can communicate with valid devices because values SK_{SN} and Tr_{seq} are secret. In contrast, the sensor validates the gateway using M_4, since M_4 employs the secret key SK_{SN}.

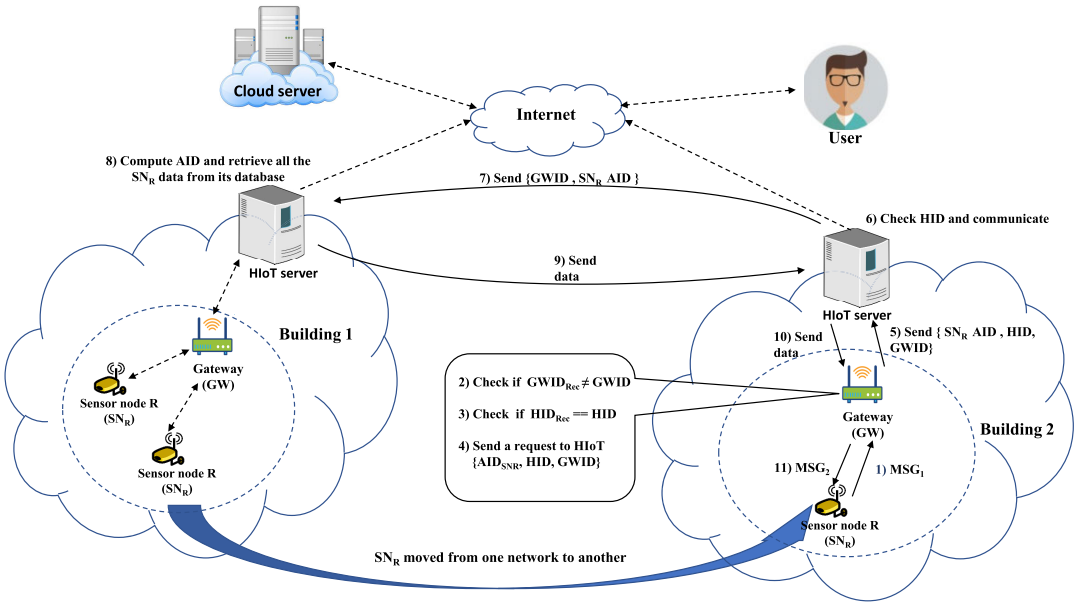


FIGURE 10. Inter-network movement.

If the sensor computes M_4' and finds it matches the received value, this means that the gateway is valid.

Furthermore, in a case of synchronization loss, the gateway authenticates the sensor with $AID_{SN} = SHID$ and M_1. If the computed M_1' matches the received M_1, and the AID_{SN} matches one of the $SHIDs$, then the gateway validates the sensor. It is worth noting that the unused pair ($SHID$, K_{emg}) is shared only between the parties and used only once. Regarding the sensor; it authenticates the gateway using M_3 because this value uses K_{emg} as a key in the HMAC.

In the continuous authentication phase, M_5 and AID_{SN} are used by the gateway to validate the sensor because M_5 uses the token TK_{SN} as a key for the HMAC, which is only known to the communication parties. Additionally, if the gateway derives the sensor's ID from the AID_{SN}, it also verifies the sensor to be valid. Conversely, the sensor validates the gateway by ACK. In the end, in each phase, the communicating parties can mutually authenticate each other.

2) ANONYMITY

Our protocol provides both privacy and untraceability. Indeed, the sensors use one-time anonymous IDs to mask their original IDs and preserve their privacy. Additionally, freshly generated randoms Tr_{seq} and r_2, ensure no two messages sent by the same sensor use the same identity across both phases (static and continuous), which also guarantees untraceability. Moreover, in the recovery phase, the sensor used a one-time $SHID$ as an AID, preserving privacy and making it hard to trace.

3) FORWARD SECRECY

Forward secrecy indicates that if an adversary compromises the secret key for the current session (or the long-term key), then they will not be able to derive previous session's keys [47].

In the static authentication phase, if the attacker compromises the recent secret key SK_{SN} he will want to learn the SK_{SN} for the previous sessions. However, he will not be able to do that, because after each successful session, both the sensor and the gateway compute a new secret key $SK_{SN_{new}} = H(SK_{SN}||ID_{SN}||Tr_{seq_{new}})$. To retrieve the old sessions, the attacker must know the previously generated Tr_{seq} and SK_{SN}, which cannot be extracted because the hash is a one-way function. The attacker is thus unable to derive any previously SK_{SN} simply by knowing the current secret key.

In the continuous authentication phase, the token TK_{SN} provides forward secrecy. If the adversary knows the current TK_{SN} for the sensor and gateway via computing $TK_{SN} = H(w \oplus v \oplus SK_{SN})$, he cannot derive the past tokens. Indeed, to do so he would need to know the previous w and v, which are impossible to derive as they are computed using a one-way hash function. Finally, we conclude from this that the proposed protocol provides forward secrecy.

4) BACKWARD SECRECY

Any security protocol needs to achieve backward secrecy to protect future sessions or communication [31]. Backward secrecy ensures that even if the attacker gains the secret session key, he will not be in a position to compromise the future session keys [48]. The proposed protocol ensures backward secrecy in both phases.

In the static phase, if the attacker compromises the current SK_{SN}, he might gain some information about the current session. However, he cannot extract any information about future sessions because the SK_{SN} frequently changes after each session, and uses the Tr_{seq}, which is secure and freshly

generated every time. The same applies with continuous authentication, and the new token TK_{SN}, which is generated for all sensed data using secure values generated by both the sensor and the gateway. As a result, an attacker cannot extract any future TK_{SN} by compromising the current TK_{SN}.

5) AVAILABILITY

Availability is one of the most important security objectives, that must be achieved. It is essential in IoT environments, because it ensures timely access to system's resources, especially for critical systems such as medical applications [13]. To ensure availability in our proposed protocol, we consider two cases when gateway becomes unavailable to the sensor:

1) If the gateway updates its values and the sensor does not: problems may occur resulting in synchronization loss [30]. In the proposed protocol, we solved this problem by employing a set of shadow IDs and emergency keys ($SHID$, K_{emg}) to recover from this failure.
2) The gateway becomes unavailable during a DoS attack: in this case, the sensors detect its unavailability by sending consecutive ($SHID$, K_{emg}) pairs. If there is no response from the gateway then the sensors can be authenticated by the nearest gateway.

6) SECURE LOCALIZATION

In the proposed protocol, we used the location beside the battery, as well as the token, to verify the sensor during the continuous phase. Additionally, the location is used in the static phase to ensure the sensor is located in the secure range of the gateway. Consequently, we made sure to secure the location values by masking them in the static and continuous phases to prevent any location from becoming compromised. The ml is masked as the following:

Static: $ml = cl \oplus H(SK_{SN} \oplus Tr_{seq})$,

Continuous: $ml = cl \oplus H[TK_{SN}||(m \oplus r_2)]$.

Moreover, if someone tries to disrupt the masked value ml, this would not be accepted by the gateway. As the M_2 value is used to ensure the integrity of all the sent values including ml.

7) DATA INTEGRITY

One of the most important security requirements here is data integrity. This is for security when the data is being sent through an insecure channel [49]. Integrity ensures the data is received free from any modification during transmission. In this protocol, we used HMAC to guarantee integrity.

In the static phase, we used M_2 and M_4 with the secret key SK_{SN} to ensure the data is received without modification. An adversary cannot modify these two values because he does not know the secret key SK_{SN}. The gateway will verify these values by computing M_2', M_4', and testing equality with the received value.

The same takes place in the continuous authentication phase, using the values M_5 and ACK, which are secured using TK_{SN}. If an attacker wished to tamper with these values, he would need the TK_{SN}, which is secure. Additionally, for the recovery phase M_1 and M_3 ensure integrity, since the K_{emg} is secure.

8) MAN-IN-THE-MIDDLE ATTACK

In the man-in-the-middle attack, the attacker intercepts the messages between two communicating nodes and secretly relays or alters these messages. The nodes are unaware of any interception and believe they are communicating with one another [49]. In the proposed protocol, if an attacker wanted to perform man-in-the-middle, he would need to know the SK_{SN}, the current location cl, and the current battery cb. Knowledge of these values is not possible simply by eavesdropping since the SK_{SN} is secure, and the other values are masked by ml and mb.

An attacker cannot perform the attack as part of the recovery process, since he needs to know the value of K_{emg}. However, this value is secure and known only to the sensor and the gateway. In the continuous authentication phase, this is the same as the static case. However, here the attacker requires the TK_{SN}, cl, cb and sd, which are secured with ml, mb, ms. Additionally, the TK_{SN} is secure, and the attacker cannot then modify any of the following messages. Accordingly, this protocol prevents the man-in-the-middle attack.

9) IMPERSONATION ATTACK

The attacker in this case tries to masquerade as a legal device [26]. The attacker needs to fake the message MSG_1, with a valid Tr_{seq} and secret key SK_{SN}. The adversary cannot masquerade in the case of this message, because neither Tr_{seq} nor SK_{SN} is known. The same is also true for the continuous authentication phase, in which the attacker needs the token to fake MSG_{C1}, but the token is secure and changes frequently.

During the recovery phase, the attacker needs K_{emg} to serve as a legitimate device, but K_{emg} is secret, and the attacker is unable to fake MSG_{1R}.

Additionally, the sensor provides its secure location (using masked location) to the gateway. This prevents malicious devices from pretending to be legal devices inside the gateway's secure range.

10) REPLAY ATTACK

Replay attacks involve an attacker trying to intercept a session and choose messages to send in subsequent sessions [30]. In our protocol, we use Tr_{seq}, fresh random numbers, and $SHID$ to prevent such an attack. In the static phase, the sensor uses the recent Tr_{seq} sent by the gateway. If an adversary wishes to send an old message from a previous session, then the gateway will terminate the session because the Tr_{seq} is not fresh. Also, the gateway will generate a new random when sending the message to the sensor. During recovery, both a one-time $SHID$ and a freshly generated random n_2 prevent a replay attack. Similarly, in the continuous authentication phase, when the sensor and gateway generate randoms and use them with M_5 and ACK both sides can ensure the message is fresh.

11) DoS ATTACK

Our protocol recovers from synchronization loss, which can be caused by a DoS attack [45]. It recovers from this loss by a pair of ($SHID$, K_{emg}) to recover from the DoS attack. Additionally, if the gateway has failed, the related sensors can be authenticated via the nearest gateway.

12) USER TRACKING ATTACK

In the static and continuous phase, all the messages from the sensor have a different *AID*. As a result, the attacker cannot track any device using the *AID*, as it changes frequently. The *SHID* is only used once, in the recovery process, thus it is hard to track a device.

13) SESSION HIJACKING

Session hijacking refers to an attacker trying to steal a legitimate session, pretending to be a legitimate user or device [50]. To prevent this attack, the gateway continuously checks the sensor's legitimacy during the session while sending the data, using three elements (the token, location, and battery). This continuous authentication process prevents any session hijacking.

Table 3, provides a summary of all the security functionalities as well as the prevented attacks and how they are achieved by the proposed protocol.

B. FORMAL SECURITY ANALYSIS

In this section, we evaluate the security robustness of the proposed protocol using Scyther. Scyther is an analytical tool used to verify security protocols [51]. Scyther [51] is also an automatic tool that can be used for analyzing and verifying security protocols developed using Python. Scyther can be used to check whether a given security protocol is likely to prove robust against multi-protocol attacks [52]. It uses Security Protocol Description Language (SPDL) to describe messages, roles, and security parameters. Scyther can be used to verify a protocol by implementing an unbounded or bounded number of roles and sessions [53]. It also has a graphical user interface and represents any attacks found using graphs. The claims in Scyther can be either defined by the user or automatically by the tool. Additionally, it uses the Dolev-Yao as an adversary model. This is one of the commonly used models, and is presented in [54]. The adversary in this model can control the network between the communication parties. The adversary can intercept messages, learn content, create new messages, and delete. He/She can also expose agents and compromise their secrets [55].

To implement the proposed protocol, we used Scyther (Compromise-0.9.2), which provides various adversary compromise models [56]. It extended the Dolev-Yao adversary model to a more powerful adversary model with additional compromising capabilities for long-term and short-term data [56]. This version supports the following settings in addition to the Dolev-Yao:

TABLE 3. Informal security analysis summary.

Functionality/Attack	Proposed solution
Mutual authentication	Both sides authenticate each other (using SK_{SN}, TK_{SN}), which is only known by both sides
Anonymity	AID and SHID
Availability	$(SHID, K_{emg})$ for synchronization loss Contact nearest gateway if the original gateway failed
Forward secrecy	The attacker can not use the current SK_{SN}, TK_{SN} to derive the past secrets or tokens
Backward secrecy	Static phase: different SK_{SN} for each process Continuous phase: different TK_{SN} for each process
Secure localization	Masking sensor's location to prevent from any tracking
Integrity	HMAC for the messages (using SK_{SN}, TK_{SN})
Replay attack	Use of fresh random numbers plus track sequence Tr_{seq}
Impersonation	Secure the SK_{SN} and the TK_{SN} to prohibit the attacker from forging the messages Also, the location ensures that outsider sensors can't pretend to be legal insider sensors
Man-in-the-middle	Static Phase: secure the SK_{SN} + masked battery + masked location Continuous Phase: secure the TK_{SN} + the battery, location and sensed data are masked. Thus, it is hard for the attacker to get any of these values
DoS	Use of Shadow IDs and emergency keys (loss synchronization) Authenticate by nearest gateway (in case of gateway failure)
User tracking attack	Frequent change of the anonymous ID and use of one time shadow ID
Session hijacking	Use of token, location and battery to continuously authenticate sensors during the session

- Key compromise impersonation (KCI): This allows the attacker to compromise the long-term key of agents.
- Testing perfect forward secrecy (PFS) and weak forward secrecy (wPFS): These options allow the user to test if the protocol provides forward secrecy. If the claim of the data remains (*Secret*) after choosing the after (PFS) option, then the protocol provides forward secrecy. Otherwise, the user can test the protocol using aftercorrect (wPFS) to determine if the protocol provides weak forward secrecy (the claim of the data after using this option will remain secret).
- Adversaries can reveal states and sessions: These options allow an attacker to compromise short-term data, such as states, randoms, and sessions.

Using Scyther, we test all the protocol phases (static, continuous, and recovery). It is worth noting that Scyther has some limitations in terms of its syntax, as it does not support conditions inside the code (e.g., if-else statements). Accordingly, we provided two different codes for the continuous authentication phase, as follows.

- state 1: corresponds to a case where time T expires, and a new static authentication phase needs to be relaunched.

```
hashfunction HMAC, h;        //hash function
const XOR: Function;         //XOR operation
const Concat: Function;      // Concatenation
usertype Getbattery;         // get current battery
usertype Getlocation;        //get current location
const GWID, HID;             // gateway  and home server IDs

protocol AutenticationIoT(SN, GW){
//sensor computation for MSG1
//macro represent abbreviation for a specific term
macro  mb= XOR( cb, h (XOR(SK,Trseq)));     //masking battery
macro ml=  XOR( cl, h (XOR(SK, Trseq)));    //masking location
macro X= XOR  (v, h(Concat(cb,cl)));        //securing v
macro M1= h(XOR(Concat(v, ID),h(SK)));
macro M2= HMAC(Concat(SK, AID, X, M1, mb, ml, GWID, HID));     //for integrity
macro AID= h(Concat(ID, SK,Trseq));                            //anonymous ID

// gateway computation for MSG1
macro cb'=  XOR( mb, h (XOR(SK, Trseq)));
macro  cl'= XOR( ml, h (XOR(SK, Trseq)));
macro v'= XOR(X , h(Concat(cb',cl')));
macro M1'= h(XOR(Concat(v', ID),h(SK)));                //verifying computed values
macro M2'= HMAC(Concat(SK,AID, X, M1, mb, ml, GWID, HID));

//gateway computataion for MSG2
macro Y= XOR(w, h(XOR(SK, n1)));                        //securing w
macro M3= h(XOR(Concat(TK, ID, Trseqnew), h(SK)));
macro M4= HMAC(Y, M3, n1, Trseq);                       //for integrity
macro Trseq= XOR(Trseqnew, h(Concat(SK, ID, n1)));      //secure new track sequence

//sensor computation for MSG_2
macro M4'= HMAC(Y, M3, n1, Trseq);
macro w'= XOR(Y, h(XOR(SK, n1)));
macro M3'=  h(XOR(Concat(TK, ID, Trseqnew'), h(SK)));
macro Trseqnew'= XOR(Trseq, h(Concat(SK, ID, n1)));

role SN{
fresh v: Nonce;
var n1,w: Nonce;

fresh cb: Getbattery;
fresh cl: Getlocation;
const  AID, ID, SK, TK, Trseq, Trseqnew', Trseqnew: Ticket;

send_MSG1(SN, GW, AID, mb, ml, X, M1, M2, GWID, HID);
recv_MSG2(GW, SN, M3, M4, Y,n1, Trseq);
macro TK= h(XOR(w',v, SK));                          //compute token
macro newSK= h(Concat(SK, ID, Trseqnew'));    //compute new secret
match (M3, M3');
match (M4, M4');
match (Trseq, Trseqnew' );
match (SK, newSK);
claim(SN, Running, GW,  M3, M4, Y,n1, Trseq );
claim_SN1(SN, Secret, SK);              //test the secret secrecy
claim_SN2(SN, Secret, TK);              //test token secrecy
claim_SN3(SN, Secret, Trseqnew);    //test new track sequence secrecy
claim_SN4(SN, Secret, v);                //test v secrecy
claim_SN5(SN, Alive);                    //aliveness for protocol roles
claim_SN6(SN, Niagree);                 //agreement
claim_SN7(SN, Commit, GW,  M3, M4, Y,n1, Trseq  );
claim_SN8(SN, Nisynch);                  //synchronization
claim_SN9(SN, Secret, newSK);        //test new secret secrecy

}
role GW{
fresh Trseqnew: Nonce;
var v: Nonce;
fresh n1,w: Nonce;
var cb: Getbattery;
var cl: Getlocation;
const AID, ID, SK, TK, Trseq, Trseqnew: Ticket;

recv_MSG1(SN, GW, AID, mb, ml, X, M1, M2, GWID, HID);
match (M2, M2');
match (M1,M1');
macro TK= h(XOR(w,v', SK));              //compute token
macro newSK= h(Concat(SK, ID, Trseqnew));
send_MSG2(GW, SN,M3, M4, Y,n1, Trseq);
match (Trseq, Trseqnew' );
match (SK, newSK);
claim(GW, Running, SN,  AID, mb, ml, X, M1, M2, GWID, HID );
claim_GW1(GW, Secret, SK);
claim_GW2(GW, Secret, TK);
claim_GW3(GW, Secret, Trseqnew);
claim_GW4(GW, Secret, w);
claim_GW5(GW, Alive);
claim_GW6(GW, Niagree);
claim_GW7(GW, Commit , SN,  AID, mb, ml, X, M1, M2, GWID, HID );
```

FIGURE 11. The SPDL code for static phase.

- state 2: corresponds to the case wherein time T is within the range during the continuous phase.

First, we tested the security for the static authentication phase. The SPDL code in Scyther for this phase is shown in Figure 11.

The results show no attacks are present in this phase. As shown in Figure 12, the following claims can be verified to check the protocol's robustness.

- The claim (*Secret*) proves the secrecy of the following values (SK_{SN}, TK_{SN}, v, w, $Tr_{seq_{new}}$, $SK_{SN_{new}}$). Indeed, preserving the secrecy of these values guarantees confidentiality and prevents several attacks arising (see section VI-A).
- The claims (*Alive, Niagree, Nisynch*) ensure the authenticity of the protocol, by testing the aliveness of roles using (*Alive*). In addition, this requires testing the agreement of values using (*Niagree*) and ensuring the synchronization of messages using (*Nisynch*). Furthermore, the claim (*Nisynch*) proves the protocol prevents replay attack and provides mutual authentication.
- The (*commit*) claim verifies that our protocol prevents an impersonation attack. This proves the proposed protocol is secure.
- The (*match*) is used to check the validity of HMAC values (M_2, M_4)

We also tested the continuous authentication phase for state 1 and state 2. The SPDL code for both states are presented in Figures 13 and 14 respectively.

The results for the SPDL codes are shown in Figures 15a and 15b, respectively. They prove the continuous authentication phase is secure and was not compromised by attacks. The claims tested are discussed below.

- The claim (*Secret*) is used to test the location secrecy of the location, battery, sensed data, and token, and it was found that all values are secure. This proves the data is confident, and the protocol can prevent the defined attacks (see section VI-A).
- We verify all the authentication claims by verifying the agreement for values, the aliveness of roles, and the synchronization of messages (*Niagree, Alive, Nisynch*). The synchronization using (*Nisynch*) also proves the protocol provides mutual authentication and prevents the replay attack.
- The commitment is verified using the (*Commit*) claim.
- The (*match*) is used to check the validity of the HMAC values (ACK, M_5).

Furthermore, we tested the security of the recovery process using Scyther. The results are shown in Figure 16, which proves there are no attacks. In this case, we verified the secrecy of the emergency key, the new track number, and the new secret key. We also checked the aliveness of the roles and the synchronization in this phase.

C. PERFORMANCE ANALYSIS

In this section, we evaluated the performance of the proposed protocol in terms of computation and communication costs.

1) COMPUTATION COST

We compute the costs of the operations used in the presented protocol, which consists of two authentication phases: static

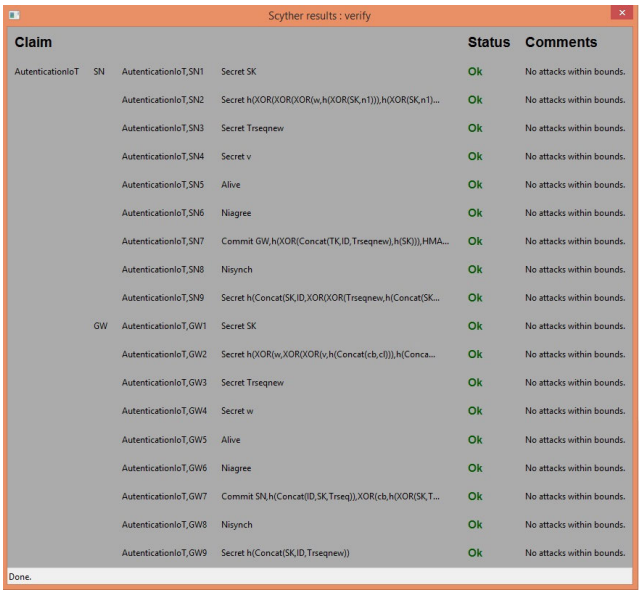


FIGURE 12. Static phase verification.

and continuous. The proposed protocol utilizes HMAC, hash, generate randoms, concatenation, and XOR operation. The concatenation and XOR computation costs are less than those of other operations [24], so we decide to ignore the time devoted to these two. To compute the time consumption for the operations hash, HMAC, generating randoms, we use the terms T_H, T_{HMAC}, and T_{RN}, respectively.

After computing the time consumption for our protocol, we found that our protocol consumes $3T_{RN}+23T_H+4T_{HMAC}$ during the static, $1T_{RN}+9T_H+1T_{HMAC}$ during the continuous phase (state 1) and $2T_{RN} + 11T_H + 2T_{HMAC}$) during the continuous phase (state 2).

2) COMMUNICATION COST

We compute the communication cost by calculating the number of bits that transfer during the authentication process [19]. We assume the lengths of secrets (SK_{SN}, TK_{SN}), random numbers ($w, v, n_1, n_2, r_1, r_2, Tr_{seq}$,) and IDs (both gateway and HIoT) are all 128 bits. To compute the total numbers of transferred bits, we assume two conditions:

- Condition1: if the SHA-1 hash function is used, then the hash digest produced is 160 bits.
- Condition2: if the SHA-2 (256) hash function is used, then the hash digest produced is 256 bits.

For both conditions we compute the communication costs for the protocol phases (static and continuous). The results of our protocol in terms of transferred bits are 3,072 using SHA-1 and 4,864 using SHA-2 (256). Our protocol provides reasonable communication cost according to the security it provides.

VII. DISCUSSION AND RESULTS

In this section, we discuss the results of our proposed protocol and compare it with its competitors, namely [30], [39], [57] and [58]. All these schemes are D2D lightweight protocols. [39] is the only D2D continuous authentication protocol that we found, so we decided to compare with static RFID authentication protocols.

The comparative analysis is based on:

- The computation cost.
- The communication cost.
- Several authentication requirements defined for the IoT environment.
- Several security properties, including prevented attacks, to evaluate the protocol's robustness.

A. COMPUTATION COST

We compared our protocol against its counterparts based on the computation cost in terms of T_H, T_{HMAC}, T_{RN} (see section VI-C1), and T_{ECC} (elliptic curve multiplication). The elliptic curve computation time has been applied to the comparison as it is used in [57] and its extension [58] to exchange the secret key used to encrypt the transferred data.

Table 4 contains a computation cost comparison between our protocol, Chuang *et al.* [39], Gope *et al.* [30], Naeem *et al.* [57], and Izza *et al.* [58]. Note that, for [39] we choose to compare with the extended version of this protocol because it computes the anonymous ID in this version.

As apparent from Table 4, our proposed protocol has more hashes in the static authentication. However, it is more secure than its competitors (see Table 8). Naeem *et al.* [57] protocol has the least number of hashes and near elliptic curve cost from [58], but it suffers from many security issues (see Table 8) and this protocol is only between the tag and the reader (i.e., no server included). Additionally, Izza *et al.* [58] is an improvement and extension to the previous protocol [57] that authenticates tag, reader and server. However, this protocol also contains several security threats. Note that for both works [57] and [58], these costs are only for session key authentication. Indeed, there are additional costs for transferring the data using symmetric key encryption with the agreed key which is costly. On the other hand, our protocol sends data during the continuous authentication phase with a minimum cost. Determining the proper balance between security and practicality is essential when designing an authentication protocol [25]. The additional hash functions used in our proposed protocol improve security and maintain practicality in terms of the computation power required. In the continuous authentication phase, our proposed protocol has almost the same hash functions as [39] present in both states, while providing recovery from DoS attack and preventing more attacks than [39].

B. COMMUNICATION COST

The comparison is done in terms of the number of transferred bits during the authentication procedure. It is worth noting that the lengths of secrets, randoms and tokens are 128 bits. According to [58], the estimated length of the timestamp and the elliptic curve output are 32 and 256 bits, respectively.

```
hashfunction HMAC, h;        //hash function
const XOR: Function;         //XOR operation
const Concat: Function;      // Concatenation
usertype Getbattery;         // get current battery
usertype Getlocation;        //get current location
usertype GetData;            //get current data


/* This is the implementation of the continuous authentication if the autentication time is
out and the sensor needs to relaunch a new static authentication
*/
protocol  ContinuousAutenticationPart1(SN, GW){
//sensor computation for MSGC1
//macro represent abbreviation for a specific term
macro  mb= XOR(cb, h(Concat(TK, XOR(m, r2))));      //masking battery
macro ml=  XOR( cl, h(Concat(TK, XOR(m, r2))) );    //masking location
macro ms= XOR(sd, h(Concat(XOR(TK,m), r2)));        //masking the sensed data
macro AID= h(ID, TK, r2);                          //anonymous ID of the sensor
macro M5= HMAC(Concat(TK, AID, mb,ml,ms,r2));      //to ensure integrity


//gateway computation for MSGC2
macro cb'= XOR(mb, h(Concat(TK, XOR(m, r2))));
macro Y1= XOR(Concat(m,TK), h(Concat(XOR(TK,r2),m)));
macro ACK= h(Concat(XOR(m,cb'), XOR(cb',r2), Concat(m, TK)));

role SN{
fresh r2: Nonce;
fresh cb: Getbattery;
fresh sd: GetData;
fresh cl: Getlocation;
const  ID, TK,m: Ticket;
var Y1, ACK;
send_1(SN, GW, AID, M5, mb,ms, r2);
recv_2(GW, SN, Y1, ACK);
macro n2'= XOR(Y1, h(Concat(XOR(TK,r2),m)));
match ((Concat(m,TK)), n2');

macro ACK'= h(Concat(XOR(m,cb), XOR(cb,r2), Concat(m,TK)));
match (ACK, ACK');       //checking the equality

//checking the security properties of the protocol
claim (SN, Running, GW, Y1, ACK);
claim_SN1(SN, Secret, cl);          //checking the secrecy of location
match ((Concat(m,TK)), n2');

macro ACK'= h(Concat(XOR(m,cb), XOR(cb,r2), Concat(m,TK)));
match (ACK, ACK');       //checking the equality

//checking the security properties of the protocol
claim (SN, Running, GW, Y1, ACK);
claim_SN1(SN, Secret, cl);          //checking the secrecy of location
claim_SN2(SN, Secret, TK);
claim_SN3(SN, Secret, cb);      //checking the secrecy of battery
claim_SN4(SN, Secret, sd);
claim_SN5(SN, Alive);
claim_SN6(SN, Niagree);
claim_SN7(SN, Weakagree);
claim_SN8(SN, Nisynch);
claim_SN9 (SN, Commit, GW, Y1, ACK);      //for commitment between partners
}


role GW{
var r2: Nonce;
var cb: Getbattery;
var sd: GetData;
var cl: Getlocation;
const  ID,TK, m: Ticket;

recv_1(SN, GW, AID, M5, mb,ms, r2);

macro Y1= XOR(Concat(m,TK), h(Concat(XOR(TK,r2),m)));
macro ACK= h(Concat(XOR(m,cb'), XOR(cb',r2), Concat(m, TK)));

send_2(GW, SN, Y1, ACK);      //sending the Y1 and ack to sensor
claim (GW, Running, SN, AID, M5, mb,ms, r2 );
claim_GW1(GW, Secret, cl);
claim_GW2(GW, Secret, TK);
claim_GW3(GW, Secret, cb);
claim_GW4(GW, Secret, sd);
claim_GW5(GW, Alive);
claim_GW6(GW, Niagree);
claim_GW7(GW, Weakagree);
claim_GW8(GW, Nisynch);
claim_GW9 (GW, Commit, SN, AID, M5, mb,ms, r2 );
}
}
```

FIGURE 13. The SPDL code for continuous phase state 1.

TABLE 4. Computation costs comparison.

Phase	Gope et al. [30]	Naeem et al. [57]	Izza et al. [58]	Chuang et al. [39]	Ours
Static authentication	$3T_{RN}$ + $14T_H$	$2T_{RN}$+$4T_H$+ $9T_{ECC}$	$3T_{RN}$+$20T_H$+ $10T_{ECC}$	$4T_{RN}$+$19T_H$+ $4T_{HMAC}$	$3T_{RN}$+$23T_H$+ $4T_{HMAC}$
Continuous authentication (state 1)	—	—	—	$1T_{RN}$+$10T_H$+ $1T_{HMAC}$	$1T_{RN}$+$9T_H$+ $1T_{HMAC}$
Continuous authentication (state 2)	—	—	—	$2T_{RN}$+$11T_H$+ $2T_{HMAC}$	$2T_{RN}$+$11T_H$+ $2T_{HMAC}$

We compare our protocol costs (see VI-C2) with the communication costs of the competitor protocols. For condition1 (SHA-1), we provide communication costs in Table 5 and compare our protocol with that of competitors [30], [39], [57] and [58].

From Table 5, it is evident that [30] has the highest computation cost, although it does not provide continuous authentication. In contrast, [57] has the lowest cost, but it does not provide continuous authentication and it is vulnerable to several attacks (see Table 8). Reference [58] which extends [57], achieves more reasonable communication cost for the authentication process with better security. Our proposed protocol and [39] both provide continuous and static authentication. As shown in the table, the communication cost of our protocol is slightly higher than in [39]. However, it is also more robust, since it prevents all known attacks while existing schemes are still vulnerable to threats. Figure 17a, presents a comparison graph detailing communication costs in term of bits for the static and continuous authentication using SHA-1.

Next, we provide the communication cost for condition2 (i.e., in case SHA-2 (256) is used for the hash operation). The proposed solution cost and when comparing them to [30], [39], [57], and [58] is provided in Table 6.

TABLE 5. Communication costs for condition1 (SHA-1).

Schemes	Static phase	Continuous phase	Total bits	Messages
Naeem et al. [57]	832	—	832	3
Izza et al. [58]	1,984	—	1,984	5
Gope et al. [30]	3,104	—	3,104	4
Chuang et al. [39]	1,536	1,088	2,624	4
Our protocol	1,824	1,248	3,072	4

TABLE 6. Communication costs for condition2 (SHA-2(256)).

Schemes	Static phase	Continuous phase	Total bits	Messages
Naeem et al. [57]	1,024	—	1,024	3
Izza et al. [58]	2,464	—	2,464	5
Gope et al. [30]	4,352	—	4,352	4
Chuang et al. [39]	2,304	1,664	3,968	4
Our protocol	2,944	1,920	4,864	4

```
hashfunction HMAC, h;        //hash function
const XOR: Function;         //XOR operation
const Concat: Function;      // Concatenation
usertype Getbattery;         // get current battery
usertype Getlocation;        //get current location
usertype GetData;            //get current data


/* This is the implementation of the continuous authentication if the autentication time is
valid and the continuous phase will keep executing
*/
protocol  ContinuousAutenticationPart2(SN, GW){
//sensor computation for MSGC1
//macro represent abbreviation for a specific term
macro  mb= XOR(cb, h(Concat(TK, XOR(m, r2))));      //masking battery
macro ml=  XOR( cl, h(Concat(TK, XOR(m, r2))) );    //masking location
macro ms= XOR(sd, h(Concat(XOR(TK,m), r2)));        //masking the sensed data
macro AID= h(ID, TK, r2);                          //anonymous ID of the sensor
macro M5= HMAC(Concat(TK, AID, mb,ml,ms,r2));      //to ensure integrity


//gateway computation for MSGC2
macro cb'= XOR(mb, h(Concat(TK, XOR(m, r2))));
macro cl'= XOR(ml, h(Concat(TK, XOR(m, r2))));
macro sd'= XOR(ms, h(Concat(XOR(TK,m), r2)) );
macro M5'= HMAC(Concat(TK, AID, mb,ml,ms,r2));      //to ensure integrity

role SN{
fresh r2: Nonce;
fresh cb: Getbattery;
fresh sd: GetData;
fresh cl: Getlocation;
var n2: Nonce;
const  ID, TK,m: Ticket;
var Y1, ACK;
send_1(SN, GW, AID, M5, mb,ms, r2);
recv_2(GW, SN, Y1, ACK);

macro n2'= XOR(Y1, h(Concat(XOR(TK,r2),m)));
not match ((Concat(m,TK)), n2');
macro ACK'= h(Concat(XOR(m,cb), XOR(n2',r2), XOR(m,TK)));
match (ACK, ACK');      //checking the equality

claim_SN1(SN, Secret, cl);
send_1(SN, GW, AID, M5, mb,ms, r2);
recv_2(GW, SN, Y1, ACK);

macro n2'= XOR(Y1, h(Concat(XOR(TK,r2),m)));
not match ((Concat(m,TK)), n2');
macro ACK'= h(Concat(XOR(m,cb), XOR(n2',r2), XOR(m,TK)));
match (ACK, ACK');      //checking the equality

claim_SN1(SN, Secret, cl);
claim_SN2(SN, Secret, TK);
claim_SN3(SN, Secret, cb);
claim_SN4(SN, Secret, sd);
claim_SN5(SN, Alive);
claim_SN6(SN, Niagree);
claim_SN7(SN, Weakagree);
claim_SN8(SN, Nisynch);
}

role GW{
var r2: Nonce;
var cb: Getbattery;
var sd: GetData;
var cl: Getlocation;
fresh n2: Nonce;
const  ID,TK, m: Ticket;

recv_1(SN, GW, AID, M5, mb,ms, r2);
match (M5, M5');
macro Y1= XOR(n2, h(Concat(XOR(TK, r2), m)));
macro ACK= h(Concat(XOR(m,cb'), XOR(n2,r2), Concat(m, TK)));

send_2(GW, SN, Y1, ACK);     //sending the Y1 and ack to sensor

claim_GW1(GW, Secret, cl);
claim_GW2(GW, Secret, TK);
claim_GW3(GW, Secret, cb);
claim_GW4(GW, Secret, sd);
claim_GW5(GW, Alive);
claim_GW6(GW, Niagree);
claim_GW7(GW, Weakagree);
claim_GW8(GW, Nisynch);
}
}
```

FIGURE 14. The SPDL code for continuous phase state 2.

From Table 6 it can be seen that our protocol has a slightly higher cost than [30]. However, [30] does not provide continuous authentication, so it is vulnerable to session hijacking. Our protocol ensures security by providing backward and forward secrecy, also preventing known attacks to which other protocols are vulnerable. Both [57] and [58] have the minimum cost, but they are still vulnerable to several attacks (see Table 8). It is important to refer that the cost considered for [57] and [58] is only for the authentication process, and there are additional high costs for transferring data using encryption with the agreed session key. Figure 17b shows a comparison graph for communication costs in term of bits for static and continuous authentication using SHA-2 (256). In the next section, we compare the proposed protocol with that produced by competitors; focusing on authentication requirements and security properties.

C. AUTHENTICATION REQUIREMENTS

We compared our protocol with its counterparts against several authentication requirements for IoT (see Table 7).

TABLE 7. Results of authentication requirements comparison.

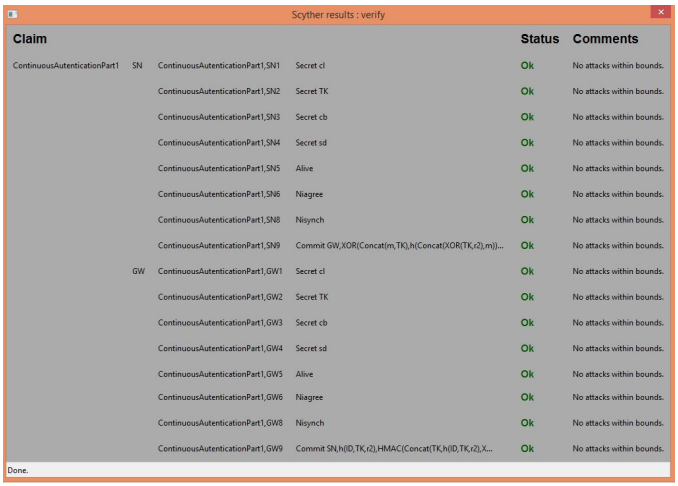
Requirements	[57]	[58]	[39]	[30]	Ours
Continuous authentication	✘	✘	✓	✘	✓
D2D	✓	✓	✓	✓	✓
Distribution	✘	✓	✓	✓	✓
Mutual authentication	✓	✓	✓	✓	✓
Performance	low	low	medium	high	medium
Anonymity	✘	✓	✓	✓	✓
Untraceability	✘	✓	✓	✓	✓

From Table 7, we can observe that [39] and our protocol satisfy all the requirements defined, whereas [30], [57] and [58] do not provide continuous authentication. All the protocols in Table 7 are D2D since the authentication process is between devices. These schemes except [57] are distributed (i.e., use edge computing), and preserve users' identity (anonymity), as well as details about devices' movement or location (untraceability). In terms of performance (i.e., computation cost), our protocol and [39] use HMAC; thus, they have a medium cost, while [30] uses hash only, incurring a low cost (i.e., hash gives high performance). In contrast the protocols [57] and [58] have low performance because they use encryption. Furthermore, these two RFID schemes demand high storage and computation cost [59].

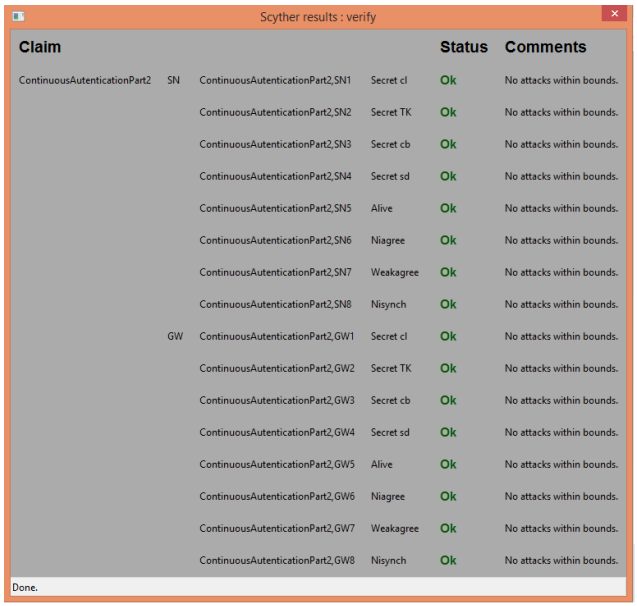
D. SECURITY PROPERTIES

We evaluate the robustness of each protocol to counter several attacks, determining whether they provide backward and forward secrecy. Table 8, provides a comparison of the works to check their security strength.

The results of our comparison show our protocol is robust against different types of attacks, and that it mitigates DoS attacks, providing both forward and backward secrecy. The protocol in [39] does not provide backward secrecy, because it uses only one secret key SK_{SN} for all static authentication sessions; however, in our protocol, the SK_{SN} is updated after each successful session. Thus, our protocol prevents all future sessions from being compromised, even in cases where the attacker has stolen the current secret key SK_{SN}. Additionally, the protocol in [39] does not prevent impersonation [40] and does not consider mitigating DoS attacks.



(a) State-1 verification



(b) State-2 verification

FIGURE 15. Continuous phase verification.

TABLE 8. Security analysis comparison.

	[57]	[58]	[39]	[30]	Ours
Prevent replay attack	✓	✓	✓	✓	✓
Prevent man-in-the-middle attack	✓	✓	✓	✘	✓
Mitigate DoS	✘	✘	✘	✓	✓
Prevent impersonating	✘	✓	✘	✓	✓
Prevent eavesdropping	✓	✓	✓	✓	✓
Prevent user tracking attack	✘	✓	✓	✓	✓
Prevent session hijacking	✘	✘	✓	✘	✓
Provide forward secrecy	✓	✓	✓	✓	✓
Provide backward secrecy	✘	✓	✘	✓	✓

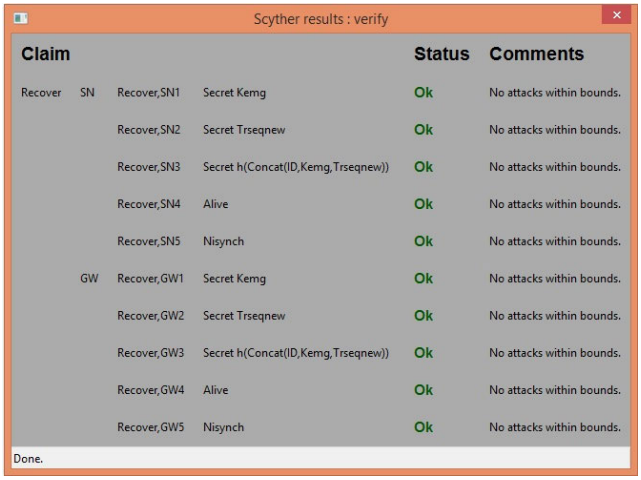
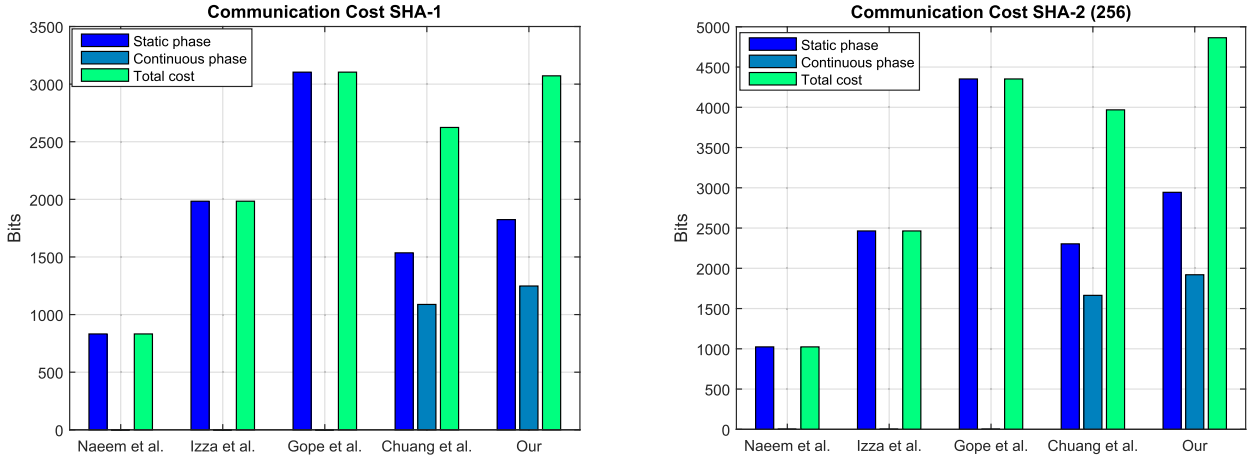


FIGURE 16. Recovery phase verification.

For Chuang *et al.*'s protocol [39], forward secrecy is only provided for partially in the token TK_{SN} (continuous phase). As we stated previously, the secret key SK_{SN} is unchangeable, and so it does not provide forward secrecy for the secret (static phase). Moreover, we can observe that TK_{SN} only provides weak forward secrecy. To explain this; the attacker who gains the secret key SK_{SN} can reveal the secret values that form the token (v and w), but can then only compromise the token for the sessions that he has actively revealed.

The other protocol, as presented in [30] does not prevent man-in-the-middle attack [23] or session hijacking. Additionally, the secret key is sent only by XOR using a random number, and this makes it relatively easily compromised when performing several attacks. The secret should be transmitted securely, such as when using hash or other safety mechanisms. In addition, it prevents only DoS in cases of synchronization loss between devices, and it does not consider cases where there is a DoS attack against the database server or gateway. On the other hand, the protocols [57], [58] do not mitigate DoS attacks and are vulnerable to session hijacking, whereas [58] provides only weak forward secrecy. Moreover, the scheme proposed in [57] suffers from several security threats as the attacker can pretend to be a legitimate reader and extract the tag's secret identity so the attacker then can impersonate, track, identify and localize the device.



(a) Condition1 (SHA-1)

(b) Condition2 (SHA-2(256))

FIGURE 17. Comparing communication cost for SHA-1 and SHA-2(256).

VIII. CONCLUSION

IoT powerful impact stems from the interconnectivity between devices. In order to secure this infrastructure, these devices must be secure and trusted. In this research, we designed an edge-based device-to-device continuous authentication protocol for IoT. The protocol utilizes device's features (i.e. token, battery and location) to continuously authenticate each other, while taking into consideration the hardware and software limitations of IoT devices by using lightweight cryptography functions, such as hash and HMAC. It also preserves the privacy of communicated devices using anonymity and untraceability. Additionally, it takes into account different movement models for IoT devices and defines a secure allowed area for each moving sensor. To mitigate a DoS attack causing temporary synchronization loss, the protocol relies on the use of emergency keys and shadow IDs to re-initiate communication between sensors and gateways. In case of permanent gateway failure, sensors can be authenticated by the nearest gateway.

To prove the robustness and efficiency of the proposed protocol, we provided both formal and informal security analyses, and measured the communication costs of the suggested model. The formal analysis was conducted using Scyther to demonstrate the robustness of our protocol against several different attacks.

IoT security is still in its early phase. For future improvements, we intend to extend the proposed protocol by (i) exploring and considering more contextual information in addition to the battery and location to enhance the security; (ii) taking the authentication process between the HIoT server and the gateways into consideration to mitigate DoS attacks that cause HIoT failure; and (iii) using machine learning to detect and prevent DoS threats.

ACKNOWLEDGMENT

This project was funded by the Deanship of Scientific Research (DSR) at King Abdulaziz University, Jeddah, under the grant No. (DG-10-612-1441). The authors, therefore, gratefully acknowledge the DSR technical and financial support.

REFERENCES

[1] T. Nandy, M. Y. I. B. Idris, R. M. Noor, L. M. Kiah, L. S. Lun, N. B. A. Juma'at, I. Ahmedy, N. A. Ghani, and S. Bhattacharyya, "Review on security of Internet of Things authentication mechanism," *IEEE Access*, vol. 7, pp. 151054–151089, 2019.

[2] Information Risk Research Team, "IoT security primer: Challenges and emerging practices," Gartner, Stamford, CT, USA, Tech. Rep. G00355851, Jul. 2018. Accessed: Sep. 12, 2020. [Online]. Available: https://www.gartner.com/en/documents/3869271/iot-security-primer-challenges-and-emerging-practices

[3] P. K. Chouhan, S. McClean, and M. Shackleton, "Situation assessment to secure IoT applications," in *Proc. 5th Int. Conf. Internet Things, Syst., Manage. Secur.*, Oct. 2018, pp. 70–77.

[4] M. H. Ashik, M. M. S. Maswood, and A. G. Alharbi, "Designing a fog-cloud architecture using blockchain and analyzing security improvements," in *Proc. Int. Conf. Electr., Commun., Comput. Eng. (ICECCE)*, Jun. 2020, pp. 1–6.

[5] S. Behal, K. Kumar, and M. Sachdeva, "Characterizing DDoS attacks and flash events: Review, research gaps and future directions," *Comput. Sci. Rev.*, vol. 25, pp. 101–114, Aug. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574013717300941

[6] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on IoT security: Application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019.

[7] M. M. Hossain, M. Fotouhi, and R. Hasan, "Towards an analysis of security issues, challenges, and open problems in the Internet of Things," in *Proc. IEEE World Congr. Services*, Jun. 2015, pp. 21–28.

[8] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali, and M. Guizani, "A survey of machine and deep learning methods for Internet of Things (IoT) security," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1646–1685, 3rd Quart., 2020.

[9] R. Clarke, "Human identification in information systems," *Inf. Technol. People*, vol. 7, no. 4, pp. 6–37, Dec. 1994.

[10] R. Falk and S. Fries, "Advanced device authentication: Bringing multi-factor authentication and continuous authentication to the Internet of Things," in *Proc. 1st Int. Conf. Cyber-Technol. Cyber-Syst.*, 2016, pp. 69–74.

[11] M. El-Hajj, A. Fadlallah, M. Chamoun, and A. Serhrouchni, "A survey of Internet of Things (IoT) authentication schemes," *Sensors*, vol. 19, no. 5, p. 1141, Mar. 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/5/1141

[12] P. M. S. Sánchez, A. H. Celdrán, L. F. Maimó, G. M. Pérez, and G. Wang, "Securing smart offices through an intelligent and multi-device continuous authentication system," in *Smart City Informatization*, G. Wang, A. El Saddik, X. Lai, G. Martinez Perez, and K.-K. R. Choo, Eds. Singapore: Springer, 2019, pp. 73–85.

[13] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on Internet-scale IoT exploitations," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2702–2733, 3rd Quart., 2019.

[14] L. Gonzalez-Manzano, J. M. D. Fuentes, and A. Ribagorda, "Leveraging user-related Internet of Things for continuous authentication: A survey," *ACM Comput. Surv.*, vol. 52, no. 3, pp. 53:1–53:38, Jun. 2019, doi: 10.1145/3314023.

[15] K.-H. Yeh, C. Su, W. Chiu, and L. Zhou, "I walk, therefore i am: Continuous user authentication with plantar biometrics," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 150–157, Feb. 2018.

[16] O. O. Bamasag and K. Youcef-Toumi, "Towards continuous authentication in Internet of Things based on secret sharing scheme," in *Proc. Workshop Embedded Syst. Secur. (WESS)*. New York, NY, USA: ACM, Oct. 2015, pp. 1:1–1:8, doi: 10.1145/2818362.2818363.

[17] P. Peris-Lopez, L. González-Manzano, C. Camara, and J. M. de Fuentes, "Effect of attacker characterization in ECG-based continuous authentication mechanisms for Internet of Things," *Future Gener. Comput. Syst.*, vol. 81, pp. 67–77, Apr. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17300407

[18] Y. Ashibani, D. Kauling, and Q. H. Mahmoud, "Design and implementation of a contextual-based continuous authentication framework for smart homes," *Appl. Syst. Innov.*, vol. 2, no. 1, pp. 1–20, 2019. [Online]. Available: http://www.mdpi.com/2571-5577/2/1/4

[19] E. Lara, L. Aguilar, M. A. Sanchez, and J. A. García, "Lightweight authentication protocol for M2M communications of resource-constrained devices in industrial Internet of Things," *Sensors*, vol. 20, no. 2, p. 501, Jan. 2020.

[20] Z. A. Alizai, N. F. Tareen, and I. Jadoon, "Improved IoT device authentication scheme using device capability and digital signatures," in *Proc. Int. Conf. Appl. Eng. Math. (ICAEM)*, Sep. 2018, pp. 1–5.

[21] K. M. Renuka, S. Kumari, D. Zhao, and L. Li, "Design of a secure password-based authentication scheme for M2M networks in IoT enabled cyber-physical systems," *IEEE Access*, vol. 7, pp. 51014–51027, 2019.

[22] P. Gope and T. Hwang, "Untraceable sensor movement in distributed IoT infrastructure," *IEEE Sensors J.*, vol. 15, no. 9, pp. 5340–5348, Sep. 2015.

[23] K.-H. Yeh, "BSNCare+: A robust IoT-oriented healthcare system with non-repudiation transactions," *Appl. Sci.*, vol. 6, no. 12, p. 418, Dec. 2016.

[24] R. Amin, N. Kumar, G. Biswas, R. Iqbal, and V. Chang, "A light weight authentication protocol for IoT-enabled devices in distributed Cloud Computing environment," *Future Gener. Comput. Syst.*, vol. 78, pp. 1005–1019, Jan. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X1630824X

[25] L. Zhou, X. Li, K.-H. Yeh, C. Su, and W. Chiu, "Lightweight IoT-based authentication scheme in cloud computing circumstance," *Future Gener. Comput. Syst.*, vol. 91, pp. 244–251, Feb. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18307878

[26] R. Martínez-Peláez, H. Toral-Cruz, J. R. Parra-Michel, V. García, L. J. Mena, V. G. Félix, and A. Ochoa-Brust, "An enhanced lightweight IoT-based authentication scheme in cloud computing circumstances," *Sensors*, vol. 19, no. 9, p. 2098, May 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/9/2098

[27] H. Kim and E. A. Lee, "Authentication and authorization for the Internet of Things," *IT Prof.*, vol. 19, no. 5, pp. 27–33, 2017.

[28] H. Kim, A. Wasicek, B. Mehne, and E. A. Lee, "A secure network architecture for the Internet of Things based on local authorization entities," in *Proc. IEEE 4th Int. Conf. Future Internet Things Cloud (FiCloud)*, Vienna, Austria, Aug. 2016, pp. 114–122.

[29] H. Kim, E. Kang, D. Broman, and E. A. Lee, "An architectural mechanism for resilient IoT services," in *Proc. 1st ACM Workshop Internet Safe Things*, Nov. 2017, pp. 8–13.

[30] P. Gope, R. Amin, S. K. H. Islam, N. Kumar, and V. K. Bhalla, "Lightweight and privacy-preserving RFID authentication scheme for distributed IoT infrastructure with secure localization services for smart city environment," *Future Gener. Comput. Syst.*, vol. 83, pp. 629–637, Jun. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17313043

[31] K. ul Hassan, A. Ghani, S. Chaudhry, S. Shamshirband, S. Ghayyur, E. Salwana, and A. Morsavi, "Securing IoT-based RFID systems: A robust authentication protocol using symmetric cryptography," *Sensors*, vol. 19, pp. 1–21, Jul. 2019.

[32] M. Safkhani and A. Vasilakos, "A new secure authentication protocol for telecare medicine information system and smart campus," *IEEE Access*, vol. 7, pp. 23514–23526, 2019.

[33] F. Zhu, "SecMAP: A secure RFID mutual authentication protocol for healthcare systems," *IEEE Access*, vol. 8, pp. 192192–192205, 2020.

[34] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979, doi: 10.1145/359168.359176.

[35] M. El-Hajj, M. Chamoun, A. Fadlallah, and A. Serhrouchni, "Analysis of authentication techniques in Internet of Things (IoT)," in *Proc. 1st Cyber Secur. Netw. Conf. (CSNet)*, Oct. 2017, pp. 1–3.

[36] P. Nespoli, M. Zago, A. H. Celdran, M. G. Perez, F. G. Marmol, and F. J. G. Clernente, "A dynamic continuous authentication framework in IoT-enabled environments," in *Proc. 5th Int. Conf. Internet Things, Syst., Manage. Secur.*, Oct. 2018, pp. 131–138.

[37] D. Ekiz, Y. S. Can, Y. C. Dardagan, and C. Ersoy, "Can a smartband be used for continuous implicit authentication in real life," *IEEE Access*, vol. 8, pp. 59402–59411, 2020.

[38] J. Wang, M. Ni, F. Wu, S. Liu, J. Qin, and R. Zhu, "Electromagnetic radiation based continuous authentication in edge computing enabled Internet of Things," *J. Syst. Archit.*, vol. 96, pp. 53–61, Jun. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762118304491

[39] Y.-H. Chuang, N.-W. Lo, C.-Y. Yang, and S.-W. Tang, "A lightweight continuous authentication protocol for the Internet of Things," *Sensors*, vol. 18, no. 4, pp. 1–26, 2018. [Online]. Available: http://www.mdpi.com/1424-8220/18/4/1104

[40] S. Sathyadevan, K. Achuthan, R. Doss, and L. Pan, "Protean authentication scheme—A time-bound dynamic KeyGen authentication technique for IoT edge nodes in outdoor deployments," *IEEE Access*, vol. 7, pp. 92419–92435, 2019.

[41] C. Bormann, M. Ersue, and A. Keranen, *Terminology for Constrained-Node Networks*, document IETF, RFC 7228, May 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7228.txt

[42] J. King and A. I. Awad, "A distributed security mechanism for resource-constrained IoT devices," *Informatica*, vol. 40, no. 1, pp. 133–143, 2016.

[43] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *Computer*, vol. 35, no. 10, pp. 54–62, Oct. 2002.

[44] McAfee, "McAfee labs threats report," McAfee, Santa Clara, CA, USA, McAfee ATR Threats Rep. 11.20, Nov. 2020.

[45] P. Gope, J. Lee, and T. Q. S. Quek, "Resilience of dos attacks in designing anonymous user authentication protocol for wireless sensor networks," *IEEE Sensors J.*, vol. 17, no. 2, pp. 498–503, Jan. 2017.

[46] L. Zheng, C. Song, N. Cao, Z. Li, W. Zhou, J. Chen, and L. Meng, "A new mutual authentication protocol in mobile RFID for smart campus," *IEEE Access*, vol. 6, pp. 60996–61005, 2018.

[47] D. Park, C. Boyd, and S.-J. Moon, "Forward secrecy and its application to future mobile communications security," in *Public Key Cryptography*, H. Imai and Y. Zheng, Eds. Berlin, Germany: Springer, 2000, pp. 433–445.

[48] A. F. Baig, K. M. U. Hassan, A. Ghani, S. A. Chaudhry, I. Khan, and M. U. Ashraf, "A lightweight and secure two factor anonymous authentication protocol for global mobility networks," *PLoS ONE*, vol. 13, no. 4, pp. 1–21, Apr. 2018, doi: 10.1371/journal.pone.0196061.

[49] R. Melki, H. N. Noura, and A. Chehab, "Lightweight multi-factor mutual authentication protocol for IoT devices," *Int. J. Inf. Secur.*, vol. 19, no. 6, pp. 679–694, Dec. 2019, doi: 10.1007/s10207-019-00484-5.

[50] Q. Hu, B. Du, K. Markantonakis, and G. P. Hancke, "A session hijacking attack against a device-assisted physical-layer key agreement," *IEEE Trans. Ind. Informat.*, vol. 16, no. 1, pp. 691–702, Jan. 2020.

[51] C. J. F. Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification*, A. Gupta and S. Malik, Eds. Berlin, Germany: Springer, 2008, pp. 414–418.

[52] M. Kompara, S. H. Islam, and M. Hölbl, "A robust and efficient mutual authentication and key agreement scheme with untraceability for WBANs," *Comput. Netw.*, vol. 148, pp. 196–213, Jan. 2019.

[53] J. Huang and C.-T. Huang, "Design and verification of secure mutual authentication protocols for mobile multihop relay WiMAX networks against rogue base/relay stations," *J. Electr. Comput. Eng.*, vol. 2016, pp. 1–12, Sep. 2016.

[54] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 2, pp. 198–208, Mar. 1983.

[55] C. Cremers and S. Mauw, *Operational Semantics and Verification of Security Protocols*. Berlin, Germany: Springer, Jan. 2012.

[56] D. Basin and C. Cremers, "Modeling and analyzing security in the presence of compromising adversaries," in *Computer Security—ESORICS 2010*, D. Gritzalis, B. Preneel, and M. Theoharidou, Eds. Berlin, Germany: Springer, 2010, pp. 340–356.
[57] M. Naeem, S. Chaudhry, K. Mahmood, M. Karuppiah, and S. Kumari, "A scalable and secure rfid mutual authentication protocol using ecc for Internet of Things," *Int. J. Commun. Syst.*, vol. 33, p. 13, Jan. 2019.
[58] S. Izza, M. Benssalah, and K. Drouiche, "An enhanced scalable and secure RFID authentication protocol for WBAN within an IoT environment," *J. Inf. Secur. Appl.*, vol. 58, May 2021, Art. no. 102705. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212620308516
[59] A. Arslan, S. A. Çolak, and S. Ertürk, "A secure and privacy friendly ECC based RFID authentication protocol for practical applications," *Wireless Pers. Commun.*, vol. 18, pp. 1–39, May 2021.

ARWA BADHIB received the B.S. degree in information technology from Arab Open University, in 2012. She is currently pursuing the M.S. degree in information technology with King Abdulaziz University. Her main research interest includes security and privacy in the Internet of Things.

SUHAIR ALSHEHRI received the Ph.D. degree in computing and information sciences from Golisano College of Computing and Information Sciences, Rochester Institute of Technology, in 2014. She is currently an Assistant Professor with the Information Technology Department, Faculty of Computing and Information Technology, King Abdulaziz University. Her main research interests include security and privacy in computer and information systems and applied cryptography.

ASMA CHERIF received the M.S. and Ph.D. degrees in computer science from Lorraine University, France, in 2008 and 2012, respectively. She is currently an Associate Professor with the Faculty of Computing and Information Technology, King Abdulaziz University, Saudi Arabia. She conducted her research at Loria, the French research laboratory. Her current research interests include access control in distributed systems and collaborative applications, cloud/edge computing, smart systems, and the Internet of Things.