Light-SAE: A Lightweight Authentication Protocol for Large-Scale IoT Environments Made With Constrained Devices

Pedro Rosa, André Souto, and José Cecílio

***Abstract*—Due to the increasing demand for Internet of Things (IoT) applications with sensitive data, the use of encryption on constrained devices is crucial for protecting and ensuring privacy and security. The devices used in such applications have limited computational power, memory, and energy resources, making them vulnerable to attacks that exploit these limitations. Lightweight cryptography has been growing fast in recent years aiming to circumvent the lack of security in these types of devices. We propose a modular solution that capitalizes on the strengths of lightweight cryptography to offer an approach that can be easily adapted to meet the needs of private wireless sensor networks. The solution is designed to work in multi-hop communication networks, where Nodes out of range of the Gateway can be part of the network, offering the same security level that a Node in the communication range of the Gateway has. To further enhance the security and lifetime of the network, our solution incorporates a key renewal mechanism and supports the use of signature schemes for integrity. Additionally, it is designed to be distributed to achieve high levels of scalability. Experimental results show that using our solution on top of standard communication protocols does not introduce significant overhead in terms of performance.**

***Index Terms*—Internet of Things and sensor networks, security management, security services, communication protocols, lightweight cryptography.**

I. Introduction

In a world full of digital technology, the *Internet of Things* (IoT) has allowed countless devices to connect over networks, creating an ecosystem that brings many advantages to our daily lives [1]. There are several areas where IoT is used, such as smart cities, transportation, sales, farming, and tourism [2]. The main goal of these networks is sensing and transferring the collected data to a Gateway or to the cloud to be processed and stored centrally [3].

Approximately 46 billion IoT devices are currently being used, and this number is expected to reach 125 billion by 2030 [4]. With such an increase, networks are bound to become more complex with the amount of data that will circulate. A substantial part of this data needs to be protected, mainly confidential information, bringing to light a critical challenge in IoT: security and privacy.

Most devices in IoT do not use secure encryption or authentication algorithms in their communication [5]. Failure to use proper encryption and other security methods can lead to severe consequences, such as unauthorized access and manipulation of data.

IoT devices are usually small and are placed in constrained environments [6]. Therefore, they are resource-limited in terms of processing power, memory, and energy. Hence, using encryption algorithms specific to IoT is necessary since traditional encryption schemes require unavailable resources in those devices. Furthermore, lightweight authentication and integrity mechanisms are also essential in IoT devices, especially on sensor networks deployed in public places.

Consequently, the scientific community started to consider *Lightweight Cryptography* [6], aiming to develop cryptographic primitives and algorithms that can execute faster, requiring less memory footprint and using very little energy. The main goal is to achieve the best trade-off between security, cost (in terms of resources), and performance [6].

National Institute of Standards and Technology (NIST) has a contest running to evaluate and standardize lightweight cryptography algorithms suitable for the constrained environments of IoT [7]. The process started with 57 submissions in 2019, and, by the time of writing this paper, is in the final stage with ten finalists (ASCON [8], ELEPHANT [9], GIFT-COFB [10], Grain128-AEAD [11], ISAP [12], Photon-Beetle [13], Romulus [14], Sparkle [15], TinyJAMBU [16], and Xoodyak [17]).

However, using the correct encryption algorithms is not the only problem in some IoT applications. Before each device can encrypt and communicate the data, they must establish encryption keys among themselves.

Our work aims to develop a Lightweight Secure Access Enhancement for Multi-hop IoT Networks (Light-SAE) key distribution scheme. It can be used with any communication protocol, as we are only concerned with the information in

Manuscript received 31 October 2022; revised 9 March 2023 and 4 May 2023; accepted 5 May 2023. Date of publication 10 May 2023; date of current version 9 October 2023. This work was supported by Fundação para a Ciência e Tecnologia through funding of the AQUAMON project (ref. PTDC/CCI-COM/30142/2017) and the LASIGE Research Unit (ref. UIDB/00408/2020 and ref. UIDP/00408/2020) and funds of Instituto de Telecomunicações Research unit (UIDP/50008/2020). The associate editor coordinating the review of this article and approving it for publication was M. Tornatore. *(Corresponding author: José Cecílio.)*

Pedro Rosa and José Cecílio are with the LASIGE, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisbon, Portugal (e-mail: fc55862@alunos.ciencias.ulisboa.pt; jmcecilio@ciencias.ulisboa.pt).

André Souto is with the LASIGE, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisbon, Portugal, and also with the Instituto de Telecomunicações, 1049-001 Lisbon, Portugal (e-mail: ansouto@ciencias.ulisboa.pt).

Digital Object Identifier 10.1109/TNSM.2023.3275011

the payload. It aims to protect the authentication information (as a payload) needed to establish a secure connection with the Gateway assuming a star topology with multi-hop communications. We propose a distributed scheme where Nodes in the network can authenticate other Nodes attempting to join the network. This mechanism enables Nodes out of the range of a Gateway to get authenticated and join the network similarly to close-ranged Nodes while reducing the computational processing required by the Gateway, making the solution more scalable than others. The Gateway generates a session key with the Node using a Diffie-Hellman key exchange protocol and a lightweight encryption algorithm to establish secure communications between Nodes and the Gateway. One of the key advantages of Light-SAE is being a modular-oriented solution, which can be deployed on the more restrictive class of devices defined in IETF RFC 7228 [18]. However, depending on the selected device's characteristics and the chosen algorithms, the complete solution may not fit in class 0 of the previously mentioned classification. Nevertheless, it can fit any device of class 1 or above.

This work was done in the context of the AQUAMON project [19]. Taking into account the application scenario of the project and its requirements, several key distribution schemes were studied to find an adequate method to distribute the parameters needed between the Nodes and the base station so they can communicate privately. However, those schemes showed some limitations concerning the resources required and the network topology that can be used to build the monitoring system. Based on the NIST ten finalist algorithms, we designed a new authentication protocol to generate symmetric encryption keys among Nodes in a multi-hop communication. Comparing our proposal with the existing ones in the literature, the proposed solution provides low resource cost, confidentiality, integrity, scalability, key freshness, capture resistance, and forward and backward secrecy. Being modular-oriented, it is configurable to fit resources available at the Nodes, providing flexibility to hold different encryption algorithms. At the same time, it is designed to be distributed, allowing the Nodes in the network to authenticate new Nodes avoiding overprocessing at the Gateway. Additionally, a key renewal mechanism provides ephemeral keys to ensure capture resistance and forward and backward secrecy and hence extending the lifetime of security of the network. Furthermore, the protocol provides integrity by using a digital signature algorithm.

In summary, the main contributions of this work are:

- Design a key distribution solution that supports multi-hop communication between Nodes and a base station, providing authenticity, confidentiality, and integrity.
- Study lightweight encryption schemes to find the most suitable ones to be used in Light-SAE to encrypt the data and all parameters needed.
- Design of a reliable network protocol targeted Multi-hop IoT Networks.

II. Related Work

Most current cryptography algorithms were designed for desktop/server environments. Many of these algorithms do not fit into constrained devices. Since 2017, the National Institute of Standards and Technology (NIST) has initiated a call, evaluation and standardization process of lightweight cryptography algorithms suitable for IoT-constrained environments [7]. The process started with 57 submissions, and it is currently in the final stage with ten finalists: ASCON [8], ELEPHANT [9], GIFT-COFB [10], Grain128-AEAD [11], ISAP [12], Photon-Beetle [13], Romulus [14], Sparkle [15], TinyJAMBU [16], and Xoodyak [17]. Table I summarizes the main characteristics of these ten algorithms, such as the mode of operation, recommended key, nonce, and block size, among other features.

Fotovvat et al. [5] made a comparative analysis of the 32 algorithms (from Round 2) that were presented for IoT sensors where different cases were tested. TinyJAMBU was the lowest energy-consuming algorithm on a Raspberry Pi 3B. This algorithm has high expectations with small and very organized code and minimal execution time. ELEPHANT showed worse results on a Raspberry Pi 3B in terms of execution time but was very similar in RAM and CPU usage compared to TinyJAMBU.

TinyJAMBU [16] was developed by Wu and Huang. It is a variant of the JAMBU [9], a block cipher authenticated encryption scheme based on key permutation. The algorithm uses a 128-bit key permutation, a state size of 128 bytes, and a message block size of 32 bytes. It also uses a nonlinear feedback shift register to update the state.

ELEPHANT [9] is an authenticated encryption scheme created by Beyne et al. It is based on a nonce-based encrypt-then-MAC construction, and the mode of operation is permutation-based. The ELEPHANT scheme consists of three instances: Dumbo, where the permutations are smaller with 160 bits and is better suited for hardware. With a 170-bit permutation, Jumbo provides higher security under the same conditions. Finally, the Delirium instance achieves the best level of protection with 200-bit permutation and is stated to be better for software use but still performs well in hardware.

Based on the characteristics described, we chose the TinyJAMBU algorithms for this work, and therefore all messages and parameters will be encrypted by it.

We will review the work concerning Device/Node authentication and message integrity upon selecting the lightweight cryptography algorithm that fits the IoT devices. We will discuss the key distribution mechanisms for wireless sensor networks (WSNs) and the digital signature mechanisms that can be used in constrained devices.

A. *Key Distribution in WSN*

Secure authentication and session key agreement protocols for the Internet of Things (IoT) environments have received considerable attention from the research community.

The authors of [20] comprehensively review dynamic key management systems in wireless sensor networks and introduces evaluation criteria for assessing these systems. They categorize dynamic key management schemes based on the type of keys, key distribution mechanisms, key cryptography methods, and network models.

TABLE I
NIST FINALIST ALGORITHMS SPECIFICATIONS

Name	Language	Mode	Authenticated encryption	Hash	Key	Nonce	Block size	Rounds	Target
ASCON [8]	C	Permutation	Yes	Yes	≤160	128	64, 128	18, 20	Software
ELEPHANT [9]	C	Permutation	Yes	Yes	128	128	64, 128	80	Soft/Hard
GIFT-COFB [10]	C	Block cipher	Yes	No	128	128	128	40	Hardware
Grain-128 [11]	C/C++	Stream cipher	Yes	No	128	96			Software
ISAP [12]	C	Permutation	Yes	Yes	128	128		31, 33, 48, 56	Hardware
PHOTON-B [13]	C	Permutation	Yes	Yes	128	128	Arbitrary	12	Hardware
Romulus [14]	C	Block cipher	Yes	Yes	128	128	128	40	Hardware
SPARKLE [15]	C	Block cipher	Yes	Yes	128, 196, 256	256	16, 24, 32	4	Hardware
TinyJAMBU [16]	C	Block cipher	Yes	No	128, 196, 256	96	20	n	Hardware
Xoodyak [17]	C/C++	Permutation	Yes	Yes	≤180	128		12	Hardware

Szymoniak and Kesar [21] conducted a comprehensive review of communication protocols addressing secure authentication and session key agreement. Their study focused on computational and communication costs. Based on their analysis, they raised the key requirements that a secure protocol must meet to be effective in an IoT environment. Light-SAE builds on this analysis and incorporates these requirements into its design.

SKEW [22] is a Self Key Establishment Protocol for Wireless Sensor Networks that manages encryption keys with less storage, communication, key transmission frequency, and computational overhead compared with similar protocols. The Gateway selects the best Nodes for coverage as cluster heads, encrypts the messages with a group key, and broadcasts them to all the Nodes in its range.

SKEW decreases the overhead by combining key refreshing with usual network messages. Only one message is required to establish a pair-wise key and one message to establish a group key. Keys are not directly transmitted within messages, only information about how they may be generated. Therefore it is less probable that keys can be disclosed. No mechanisms for integrity are mentioned, and their results show high scalability.

Messai et al. in [23] proposed a lightweight sequence-based key management (SKM) scheme. SKM involves pre-distributing the first term and recursive formula of a numerical sequence to the sensor Nodes, after which pairwise keys are established between each sensor Node and its neighboring Nodes. To effectively resist external attacks, periodic updates to the key or revocation of compromised Nodes are necessary, depending on the network's security level.

In our proposal, all the messages sent in the network are encrypted (including messages with information related to keys), achieving a good level of confidentiality.

Based on pair-wise and group key management schemes, the work in [24] proposes an alternative method where a membership authentication mechanism is added to the key pre-distribution process. A control center is used to schedule the entire WSN communication. It is also stated that the scheme resists capturing attacks more efficiently and performs better than similar schemes. However, when it comes to real-life scenarios, it is hard to ensure the credibility of the control center. The control center authenticates the sensors by comparing the parameters known only by itself and the object. A timestamp prevents replay attacks from achieving a secure authentication process. The scheme achieves a good level of confidentiality and integrity by using a hash function to authenticate messages. Sensors can randomly join and leave the network. As long as the sensor passes the authentication of the control center when entering, it can obtain the session key, proving good scalability. As for key refreshing, the method contains a heartbeat feature, and if a Node is captured, the sensor heartbeat feedback will exceed the time interval set so that the control center will revoke the key.

Kumar et al. [25] propose a decentralized scheme for homogeneous cluster-based architecture. The scheme uses a hash chain to generate a key. Nodes use the strategy proposed in [26] to find common keys. The architecture achieves a high level of confidentiality as all parameters are sent encrypted with a timestamp. To further maintain the integrity of the messages, the hash value of the message is also sent. As for scalability, if a new Node is added to the network, a set of keys from a key pool is loaded to the Node so it can find common keys with its neighbors. Keys can also be refreshed (for a specific Node or all Nodes). To do so, the base station sends a new parameter that will be used to generate the key. If a chain is compromised, so is the Node. There is also a trade-off between computation and resiliency by increasing the length of the chains.

Kadri et al. proposed an efficient key management scheme for hierarchical wireless sensor networks in [27]. The method proposed is a simplified public-key infrastructure based on the authenticity of the base station as a secure entity. The base station and cluster heads are crucial to securing data transmission and establishing a session key with the Nodes in its cluster. After all the proceedings, each sensor shares a symmetric key with its cluster head and the base station. The scheme is based on symmetric key encryption, making it more efficient regarding resource consumption. The architecture has an authentication system for all devices, and the integrity of messages is secured by a message authentication code MAC encrypted with the session key. As usual in these solutions, a key refreshing mechanism is used to increase the life of these networks. The cluster heads launch the key update. However, if a Node leaves the network after the refresh, he will no longer have the valid key providing forward secrecy. In our proposal, we redesign the approach to deal with this problem.

An improved energy-efficient key distribution and management scheme is proposed in [28]. This solution has less overhead than the above schemes because there is no requirement for encryption and decryption of the parameters.

Here, cluster heads perform almost all the work. After establishing keys, all information is encrypted throughout the network. There is no integrity mechanism, and as the parameters to establish keys are plain text, messages could possibly be modified without any detection. Though the scheme can support up to 1000 Nodes, the system does not scale as more memory will be required to store the keys. Nodes can refresh the keys using a random number sent by the Gateway and bitwise XOR operation. This solution seems resilient to Node capture attacks as long as the number of captured Nodes does not exceed a certain threshold.

A secure multifactor authenticated key agreement scheme to be used in industrial IoT devices is proposed by Vinoth et al. [29]. It allows authorized users to access the sensing device for maintenance remotely. It uses a combination of passwords, biometrics, and smart card to authenticate the user, and it is secure against several common attacks and provides mutual authentication and perfect forward secrecy.

In the literature, it is also possible to find mechanisms that allow efficient authentication using physical unclonable functions (PUFs) [30]. Light-SAE does not consider any specific hardware feature, like PUF. It is designed to be easily implemented at the application level of IoT devices.

B. Digital Signatures

Yavuz and Ozmen [31] proposed a lightweight multi-time digital signature scheme called SEMECS. The proposed scheme is highly efficient due to minimal and fast processes to generate a 32-byte signature, which translates to energy efficiency in constrained processors that are used in IoT applications. The scheme also uses a compact 32-byte private key and requires little storage. The authors provide a security analysis where they exploit the fact that SEMECS uses multiple-time signatures, meaning it has higher security for a limited number of times. Comparison aspects of the scheme with others are also provided, such as signature generation time, verification time, private/public key sizes, and energy usage.

Costello and Longa present a lightweight digital signature scheme called SchnorrQ [32], which combines the well-known Schnorr scheme and elliptic curves from FourQ [33]. FourQ is a high-security and performance elliptic curve achieving a 128-bit security level. Results show that FourQ is around five times faster than methods offered by NIST, like the NIST P-256 curve and Ed25519 [34]. SchnorrQ offers extremely fast and highly secure 64 bytes digital signatures using a 32 bytes public/private key pair. SchnorrQ efficiency translates not only to reduced latencies but also to significant savings in energy, making this signature scheme an excellent way to achieve integrity in lightweight implementations and low-power applications such as IoT.

III. Light-SAE: Lightweight Secure Access Enhancement for Multihop IoT Networks

The Lightweight Secure Access Enhancement for Multihop IoT Networks (Light-SAE) is a protocol to generate encryption keys among Nodes in multi-hop communications. It is designed to handle large-scale networks built from constrained devices.

TABLE II
Variable Definition

Abbreviation	Meaning
DK	Default Network Key
PK	Collection of keys
K_i	i-th key of the Collection of keys
$R1$	Random value 1
$R2$	Random value 2
X	Second parameter of challenge
SK	Session key
f	Encryption function
f_{K_i}	Encryption function with K_i as key

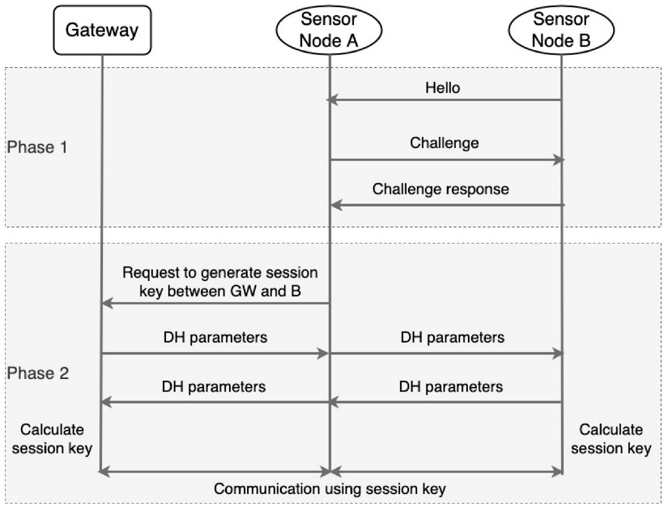

Fig. 1. Overview of information exchanged in Light-SAE.

Light-SAE uses already authenticated Nodes to authenticate new Nodes. By doing this, we provide a mechanism for a Node to authenticate and join a large-scale network, dropping high computational processing at the Gateway or other central server. The Gateway or a central server just needs to generate a session key with the Node using the Diffie-Hellman Key exchange protocol and a lightweight encryption algorithm.

Light-SAE involves two main phases: Node authentication and session key generation. Besides these two main phases, Light-SAE has other necessary mechanisms like session key renewal and data signature.

Light-SAE is an application-level scheme that can be used with any communication protocol, focusing solely on the information in the payload. Its purpose is to safeguard the authentication information required to establish a secure connection between Nodes and the Gateway, considering multi-hop communications with a star topology. Since it is a decentralized mechanism, it reduces the overhead in the Gateway and enhances the network's scalability. In order to understand the definition of the protocol, Table II shows the variable description used.

Considering Nodes A, B, and a Gateway, Figure 1 represents the proposed approach, which allows the distribution of keys in a multi-hop way. The proposed approach allows, for instance, if Node B is out of range of the Gateway, it can communicate using an intermediate and already authenticated

Node A. The protocol requires a hardcoded Default Network Key *DK*, an encryption algorithm *f*, a collection of keys *PK* (pre-stored in the Node), a prime value p and g for the Diffie-Hellman key exchange protocol, and finally, a digital signature algorithm to sign and/or verify the integrity of the messages. Every message exchange in this proposal, including establishing the key, will be encrypted with a lightweight encryption algorithm. Any encryption algorithm may be used in Light-SAE as long as the keys are the same size as the ones defined.

The protocol assumes that Nodes will perform several processes in different phases. Hence, acknowledgment steps are necessary at specific execution points of the protocol. For this purpose, different kinds of messages with different IDs are used. When Nodes or even the Gateway receive them, they can proceed accordingly. The types of messages are as follows:

- *Authentication message* - These are messages sent by Nodes trying to join the network to already authenticated Nodes or the Gateway. Contents of the message are the Node ID and a Key K_i;
- *Authentication response message* - These are the response to the authentication messages and are sent by authenticated Nodes or the Gateway. Contents are a challenge meant to be answered by the Node seeking authentication;
- *Challenge response message* - As the name suggests, it consists of the challenge response and is meant for the Node/Gateway that sent the challenge in the first place;
- *Request to gateway message* - Authenticated Nodes send this message to the gateway. It is a request for the Gateway to initiate the session key generation process with the recently joined Node through himself. The contents of the message are the Node ID that wants to join and the K_i used during its authentication.
- *Public value message* - Every device eventually sends this message. This message's contents are the parameter needed (public value) by the other device to generate the common session key. Nodes can send other parameters with the public value in this message, such as its own ID for verification purposes, and a public key from the digital signature algorithm;
- *Session key renewal message* - This message is sent by the Nodes once per interval of time. It consists of the same contents as the *public value message*, and it is meant for the Gateway to calculate a new session key with the new public value sent.
- *Data message* - Message containing collected data by the Nodes.
- *Key refresh time message* - This is a message sent by the Gateway. It tells the Node it is time to refresh the session key in case the Node does not begin the process itself.
- *Acknowledge message* - Message sent by the Gateway to the Node to acknowledge receiving of the new public value from the *session key renewal message*.
- *Confirmation to intermediate node message* - Once an authenticated out-of-range Node needs to communicate with another intermediate Node to reach the Gateway, the Node keeps sending *data messages* to the Gateway. When this happens, the Gateway sends this type of message to the new intermediate Node to inform it that the Node out-of-range is authenticated and can forward its data.

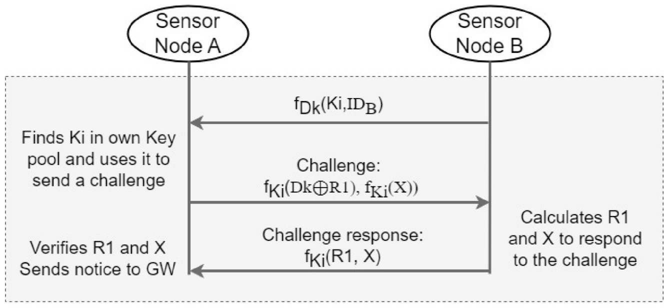
Sensor Node A
Sensor Node B
fDk(Ki,IDB)
Finds Ki in own Key pool and uses it to send a challenge
Challenge: fKi(Dk⊕R1), fKi(X))
Calculates R1 and X to respond to the challenge
Challenge response: fKi(R1, X)
Verifies R1 and X Sends notice to GW

Fig. 2. Node Authentication.

Algorithm 1 Node Trying to Join the Network

```
$K_i = Kp[rand()\%size]$
sends authentication message
decrypts M using $K_i$
$R1 = (DK \oplus R1) \oplus DK$
sends challenge response ($R1$, $X$)
```

Next, we describe the Node authentication, session key generation and renewal, and digital signatures procedures defined by Light-SAE.

A. Phase 1 - Node Authentication

When a Node wants to join the network, it sends an *authentication message* to neighbor Nodes that are near to authenticate it (if the new Node is near the Gateway, the Gateway will receive the request. Otherwise, another authenticated Node will deal with the request). This message contains a random key K_i from a collection of keys *KP* and its ID, encrypted with the encryption algorithm using as a key the *DK*. The collection of keys is assumed to be loaded by an administrator during the programming phase of the Node. Then, when the authenticated Node receives this message, decrypts it with *DK* and checks whether K_i is in its own *KP*. It puts the Node ID on its denylist if it does not find it. If the K_i is found, it will send a challenge to the Node using K_i to verify the identity of the new Node. The challenge consists of: $f_{K_i}(DK \oplus R1, f_{K_i}(X))$ where $R1$ is a random value generated by Node A. X can be different values, from a variable that characterizes the Node that sent the challenge with its Mac $(f_{K_i}(MacA))$, a variable that uses a timestamp $(f_{K_i}(Timestamp \oplus ID_A))$ or a simple random variable ($R2$). After the new Node receives the challenge, it has to calculate $R1$ and X to send the challenge response: $f_{K_i}(R1, X)$. In Figure 2, it is possible to see the entire Node authentication process. Algorithms 1 and 2 describe this process for the Node trying to join the network, and the one authenticating it.

Nodes already authenticated are waiting to receive messages from other Nodes trying to join the network. After receiving the message, it checks if the K_i sent by the Node is in the PK collection. If the key is valid, the Node keeps the K_i to encrypt further messages until the session key gets generated. If not, the Node ID is added to a denylist for future reference.

Algorithm 2 Authenticator Node

```
$flag = 1$
decrypt(M) with $DK$
**for** $i < size$ **do**
  **if** $PK[i] = K_iB$ **then**
    $flag = 0$
    $K_iA = K_iB$
    break
  **end if**
**end for**
**if** $flag = 1$ **then**
  insert $ID_B$ to denylist
**end if**
$R1 = rand()$
$X = rand()$
challenge $= f_{K_i}(DK \oplus R1, X)$
sends authentication response message
decrypt(M) with $K_i$
**if** $R1_A = R1_B$ and $X_A = X_B$ **then**
  send request to Gateway
**else**
  insert $ID_B$ to denylist
**end if**
```

The Node will now create and send a message consisting of a challenge to further authenticate the Node trying to join the network: M$\{f_{K_i}(DK \oplus R1, X)\}$, where *R*1 is a random value generated by Node A. X can be different values, from a variable that characterizes the Node who sent the challenge with its Mac $(f_{K_i}(MacA))$, a variable that uses a timestamp $(f_{K_i}(Timestamp \oplus ID_A))$ or a simple random variable (*R*2). Then, the Node has to wait for the challenge response. After receiving it, it decrypts the message and checks if *R*1 and X are correct, placing it on the denylist if that is not the case. If everything is correct, the Node will send a *request to gateway message* (encrypted with the session key between both of them) containing K_i used in the process above and the ID of the Node so that the Gateway can compare it later: M$\{f_{SK_{GWA}}(ID_B, K_i)\}$. This request for the Gateway is meant to initiate the session key generation process with the joining Node.

B. Phase 2 - Session Key Generation

To create the session keys between Nodes and the Gateway, the Diffie-Hellman protocol is used. The parameters are encrypted with the K_i sent in the request, making it resilient to man-in-the-middle attacks (MIM). Every Node in the network should have a g and a prime p value required by the DH protocol. This way, Nodes only have to compute a private key to generate the session key. Figure 3 illustrates the process used to generate the session keys.

The Gateway, after receiving the *request to gateway message* (containing the K_i and the Node ID) from the Node that authenticated the new Node, proceeds to execute Algorithm 3. It starts by decrypting the message and saving K_i and ID from the Node. After that, it calculates the value α with the hard coded values "*g*", "*p*" and own private key "*a*". The Gateway sends the message Mα to the Node. The Gateway now waits for the Node to send a *public value message*. Once it does, it decrypts it and saves the *publicKey* from verifying the signatures later. It also checks if the ID_B, sent from the Node B along with the public value, matches the one sent by Node A, which previously has authenticated it (this process is skipped if Node B was authenticated directly by the Gateway and not by an intermediate Node A). If the ID matches, the Gateway generates the session key, allowing them to communicate privately.

Algorithm 4 presents the Node process of generating the session key. After receiving the public value from the Gateway, it starts by generating a private key *b* to compute the value β with that private key and the hard-coded values *g* and *p*. It also generates a private/public key pair that will be used to sign and verify message signatures. The Node sends the message M: $\beta, publicKey$ to the Gateway. Finally, it calculates the session key with the received public value α.

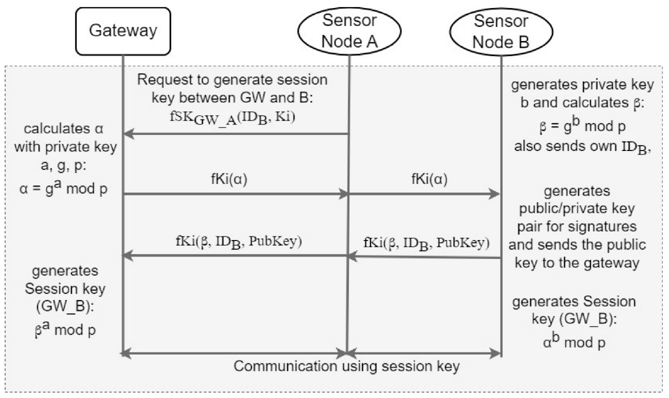
Gateway
Sensor Node A
Sensor Node B
Request to generate session key between GW and B: $fSK_{GW\_A}(ID_B, Ki)$
calculates α with private key a, g, p: $\alpha = g^a \bmod p$
generates private key b and calculates β: $\beta = g^b \bmod p$ also sends own ID_B,
fKi(α)
fKi(α)
generates public/private key pair for signatures and sends the public key to the gateway
$fKi(\beta, ID_B, PubKey)$
$fKi(\beta, ID_B, PubKey)$
generates Session key (GW_B): $\beta^a \bmod p$
generates Session key (GW_B): $\alpha^b \bmod p$
Communication using session key

Fig. 3. Session Key Generation.

Algorithm 3 Gateway

```
decrypt M with $SK_{GwA}$
$char\alpha = g^a \mod p$
encrypt($\alpha$) with $K_i$
sends public value message to Node
decrypts M with $K_i$
**if** $ID_B(sentbyA) = ID_B(sentbyB)$ **then**
  $SK_{GwB} = \beta^a \mod p$
**else**
  insert $ID_B$ to denylist
**end if**
```

Algorithm 4 Recently Joined Node That Needs a Session Key

```
decrypts M with $K_i$
char $b = rand()$
char $\beta = g^b \mod p$
generates $private/public$ key pair
encrypt($\beta, publicKey$) with $K_i$
send public value message to Gateway
char $SK = \alpha^b \mod p$
```

With this solution, the Gateway will have session keys with every Node in the network, and all the communication is encrypted using these session keys and encryption algorithms.

C. Session Keys Renewal

Session keys should be renewed periodically to increase the network lifetime and general security. Depending on each scenario and environment, the period necessary to renovate the keys is different and should be thought out. To renew the key, we repeat the process in phase 2. But this time, all the parameters go encrypted with the old session key instead of K_i. This time the Gateway does not need to send the parameter α from the DH protocol since it is a constant value throughout the entire scheme (because its private key does not change without administrator interference). Node B starts by generating another parameter β, with a distinct and newly generated private key. After computing the parameter β, it sends it to the Gateway, so both can generate a new session key. After the Gateway receives the message, it generates the session key and sends an *acknowledge message* to Node B, ensuring the new session key can be generated and updated on both sides. Then, they discard the old session key and use the new one to exchange messages. However, if the Node does not send the public value, the Gateway must initiate the key renewal process by sending a *key refresh time message*. After some time, if the Node does not send the new public value, it is removed from the network by the Gateway. In this process, the private/public key pair used in the signatures is also renewed by the Node, and the new public key is sent to the Gateway along with the latest public value β.

D. Digital Signature

The proposed protocol provides a scheme with authentication and confidentiality. A digital signature approach was also added to Light-SAE, providing data integrity and making sure no message in the network has been tampered. In Light-SAE, each Node's private key is calculated randomly and used for the DH parameters and session key generation. To implement the digital signature mechanism, Nodes must calculate the private/public key pair used to generate and verify the signatures and send the public key to the Gateway along with the public value, using the *public value message*. The Nodes would use the private key to sign the messages, while the Gateway would use the public key received from the Node to verify their signed messages correctly. The private/public key pair is refreshed along with the session key, and the SchnorrQ digital signature algorithm [32] is used to sign the messages.

IV. Light-SAE: Protocol Operation

During the authentication and session key generation process, the Gateway will receive several messages from a few Nodes and need to use the correct key to decrypt them. For example, when the Gateway receives a request from Node A to generate the session key with a Node B (*request to gateway message*), it will need to decrypt the request with Node A session key and encrypt the public values with the established K_i so Node B can correctly decrypt the message to generate the session key. To do this, we use a message type parameter on each message. This way, Nodes can check the type of message they receive and use the correct key. Using the previous example, the Gateway would see that the message coming from A is a request to generate a session key with another Node (*request to gateway message*), so the key needed is the session key with authenticated Node A. For the next message, as it is a *public value message*, the Gateway must decrypt with the K_i used by Node A and Node B during the authentication.

Taking into consideration a scenario where Node B uses Node A to communicate with the Gateway. Suppose Node A goes offline for some reason. If this happens, Node B will not communicate with the Gateway anymore. To fix this problem, Node B must select another neighbor to communicate and find another path to reach the Gateway.

In this case, Node B does not need to repeat the authentication and session key generation procedure. Since it was a valid Node in the network, meaning after finding the new intermediate Node, it will keep sending *data messages* to the Gateway. However, malicious Nodes could take this to their advantage and do the same. In this situation, since the intermediate Nodes do not decrypt *data messages*, these are not verified and are sent to the Gateway as they are. The Gateway verifies the message, and since a valid Node did not send it, it will discard them and end the communication. The intermediate Node, however, keeps receiving and re-sending messages to the Gateway from the malicious Node. To fix this issue, a *confirmation to intermediate Node message* is sent from the Gateway to the intermediate Node. Hence, it knows it can keep re-sending messages from that Node or stop communications with it.

V. Security Analysis

Light-SAE being a new proposal to be used as a cryptographic scheme in IoT, has to be secure. We present in the next paragraphs a discussion of the security analysis of the methods used in Light-SAE. For example, in our proposal, we use a modified version of the Diffie-Hellman protocol. We discuss the difference between our modified version and the original version and how it prevents some well-known attacks. Finally, we show the effectiveness of the generated encryption keys considering their size and how they are calculated using the brute force attack. Although it does not provide formal proof of security, it provides an indication of how secure the system is. Note that, apart from the modified version of Diffie-Hellman, the rest of the system security is based on the impossibility of accessing the network key for a nonligitamate user. The security of all the processes relies on the fact that we use already-proven lightweight security cryptographic encryption schemes.

A. Modified Diffie-Hellman

Diffie-Hellman key exchange algorithm is a protocol for establishing a common key between two users. That key is then used to encrypt the communications between the parties [35], [36]. Therefore, it is usually used within other

TABLE III
MODIFIED DH DIFFERENCES

Parameter	Diffie-Hellman	Light-SAE	SMDH
Asymmetric	Yes	Yes	Yes
Energy efficient	Yes	Yes	Yes
Confidentiality	No	Yes	Yes
MIM attack	Yes	No	No

protocols as a building block. To be resilient to man-in-the-middle attacks, the protocol usually relies on a trusted server that authenticates the users. In our proposal, to achieve confidentiality, all parameters calculated in the DH protocol are distributed encrypted, as mentioned previously. Recall that based on the security of a network key that only legitimate users have, one can encrypt all the messages. This prevents MIM attacks, which will be explained in more detail in due course. In Table III, it is possible to see the main differences in the original DH, our version, and the method proposed in [36] called SMDH aforementioned in Section II.

B. Security of the Scheme

First, we analyze the security of the messages exchanged within the authenticated Nodes. Since the messages are encrypted with a symmetric key that is only shared between the Gateway and the Node and the cryptographic encryption underlying the scheme is assumed to be secure, then, without the knowledge of the encryption key, it is not possible to access the data. We also assume that only legitimate Nodes and the Gateway can access the collection of keys and the network key. Note that this also applies to the Nodes inside the network. Since we use a Diffie-Hellman protocol to establish a pair of shared keys between each Node and the Gateway, the Nodes inside the network cannot access the key used, and therefore the messages are also confidential inside the network.

Secondly, we discuss the security of the authentication process. We call the reader's attention to the fact that since the values that the Node needs to reply to the challenge at the second stage require that he knows the collection of keys and the network key, as the challenges are generated uniformly at random. Again, only legitimate Nodes can comply with the correct answer to the challenge, as no Node outside the network has access to those keys.

Now, we discuss the security of key session renewal. As mentioned in [37], in Diffie-Hellman, the difficulty of breaking it, i.e., finding out the shared key that was established through the exchange process, scales, classically, exponentially with the size of the primes used. However, one additional bit on the prime numbers is not equal to one extra bit of "key strength" in AES keys, for example. The reason behind it is that not every number in that interval is a prime. Looking at AES-256, one could expect the key to be a string of 256 random bits, meaning that the key could be any of the 2^{256} possibilities. Yet, if we use 256-bit primes in DH, fewer than those 2^{256} possibilities exist. Therefore, if one aims for the same key strength as in any AES version, we need to use larger prime numbers to have a sufficiently large number of possible primes.

To maintain the lightweight aspect of our solution and use NIST encryption algorithms, we use small prime numbers with 256 bits to obtain a 256-bit key. Notice that, in these conditions, even without considering the encryption part of Algorithms 4 and 3, the scheme has the same security as the standard Diffie-Hellman protocol for 256-bits. However, as shown in the calculations below, the scheme is secure for a reasonable amount of time, even with this size of keys. If the key is calculated with a 256-bit prime, that means we are working with values in the "realm" of 2^{256} or approximately 1.2×10^{77} possible numbers. Among all these values, we can estimate the number of primes using the Prime Number Theorem [38] that states that the distribution of prime is of the form $\pi(x) = \frac{x}{\log_2 x}$. There are approximately 1.5×10^{75} or 2^{250} possible prime numbers for choosing the public value. If we estimate the time it would take to brute force such a value with a rate of 2^{30} trials per second, it will take approximately 2^{220} seconds or 5.3×10^{58} years to discover the secret.

Brute force is not the best attack for methods with large numbers, and many effective attacks could reveal the secrets way faster. We refer the reader to the best results known so far [39] that still be of exponential time for classical computers. However, this bound method provides a basic understanding of the keys' security and compares them to other known schemes. According to these results and the conditions where the real scenario is implemented, one may estimate the minimum time that Light-SAE will require for the key refresh mechanism to be applied.

Finally, we discuss the security of the digital signature process. Note that the mechanism to establish a pair of private/public keys used in our solution for digital signatures is similar to the Diffie-Hellman key exchange. The Node signs with its private key, and the Gateway verifies with the public key it has established with that Node. Note that only the Node knows its private key and therefore is the only one able to sign its own messages. Since the Gateway has the public key can verify the legitimacy of the signature. Hence the security of the signature process is equivalent to the minimum security of the Diffie-Hellman size key applied and the SchnorrQ with similar parameters.

C. Practical Attacks Analysis and Countermeasures

For the sake of the presentation, we give a detailed discussion about several practical attacks and how our solution prevents them.

1) Brute Force Attack: As mentioned previously, although brute force is not the best possible attack, this is one of the most well-known attacks [40]. The best-known attacks are of a similar order of magnitude as the brute-force [39] and therefore they provide a practical perspective of the security of the scheme. In this type of attack, the adversary tries every possible key combination until the correct one is found. Depending on the size of the key, the time needed to discover the key successfully is exponential.

Let us consider that a machine or group of machines could run through 2^{30} (1.07 billion) keys per second (this number varies depending on how powerful the devices are and the

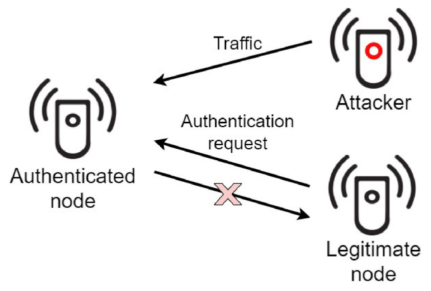

Fig. 4. Denial-of-service attack.

number of cores being used together to find the right key). To see how many seconds it would take to brute force a key depending on its size, we can use the following equation:

$$BruteForce_{Time}(s) = \frac{2^{KeySize(bits)}}{2^{30}} \quad (1)$$

If we calculate the time needed for a 256-bit key, which is the key size used by AES256, it will take 2^{256-30} seconds or approximately 3.42×10^{60} years to discover the key. This value, however, is obtained by calculating every possible key, representing the maximum time it would take for a key of such size. The average would be half of the time since the key could be either on the lower or higher half of the key space. This means that it would take approximately 1.71×10^{60} years since each additional bit in the key size doubles the key space (1.24×10^{50} times the Universe's lifetime $\approx$13.7 billion years).

2) Denial of Service (DOS) Attack: In DOS attacks, the adversary floods the target with traffic and information to cause a crash. In both scenarios, the main goal is to deprive and refuse service from the target to legitimate users. In the case of IoT devices, like the Nodes, for example, it is even more critical since they might have a limited power supply.

In Figure 4, it is depicted a scenario of this attack in our implementation. Because of the traffic that the attacker is directed to an authenticated Node, it can not respond to the authentication request from a legitimate Node trying to join the network. However, as previously mentioned in our scheme, the blacklist may minimize the problem as the Node will ignore the messages sent by this attacker. This message verification takes very little computational power, so it will not affect the standard functionality of a Node or the Gateway.

3) Nodes Being Curious: In the case of Light-SAE, where some Nodes have important roles, such as authenticating Nodes, it is important to observe that one authenticated Node might deviate from his standard behavior and try to access data in the network. It is important to note that even if an authenticated Node can authenticate others, the session keys are always established with the Gateway and not with other Nodes. This prevents also an authenticated Node from changing messages in the network as they are encrypting with the session key with other Nodes.

VI. Results

Light-SAE and its algorithms were tested in a Raspberry Pi 3 Model B+ with a 1.4GHz 64-bit quad-core processor. In this section, we present and discuss the results obtained from a set of experiments which allowed us to measure the execution time of different operations (authentication request, challenge generation, challenge-response, challenge verification, calculation of DH public values and session keys and the digital signature) involved in the protocol. Energy consumption of the scheme in single-hop and multi-hop communication is also presented. Finally, a detailed comparison of our solution is presented with several other similar approaches.

In our evaluation, we focused on several key performance indicators (KPIs) to measure the effectiveness and efficiency of our proposed solution based on [41]. The most important KPI we considered was the resulting size of the payload. We also consider the time required to encrypt and decrypt the information during the authentication process and the overall time needed to complete the process. By analyzing these KPIs, we aimed to quantify the overhead of our solution compared to a scenario where our proposal is not used. We considered these KPIs mainly because they allow quantifying the impact of Light-SAE on the network's performance and resource usage. The size of the payload, in particular, is a crucial factor as it affects the amount of data transmitted and the network bandwidth consumption. The time required for encryption and decryption is another important KPI, as it directly affects the latency of the authentication process. Optimizing the encryption and decryption algorithms can reduce the resources needed, processing time, and latency.

Overall, based on these KPIs, we can evaluate Light-SAE's effectiveness and efficiency, which allows us to identify the strengths and weaknesses of the solution.

A. Encryption Algorithms

The first subset of experiments we did to evaluate Light-SAE is related to the encryption algorithm we can use. We selected the TinyJAMBU [16] and ELEPHANT [9] algorithms from the related work analysis as candidates for our protocol. Considering both algorithms, they have a list of variables that can be configured. Some of these variables can be tuned to fit personal usage:

- CRYPTO_KEYBYTES: Allows for keys of 32 bytes in both algorithms. TinyJAMBU does not allow higher values. However, increasing this value on ELEPHANT works fine, changing the ciphertext (size stays the same) and increasing execution time.
- CRYPTO_NPUBBYTES: Allows for nonces up to 24 bytes in both algorithms. While on TinyJAMBU, the increase of this value does not change the resulting ciphertext, on ELEPHANT, this value can be increased to a maximum of 20, allowing the usage of nonces 40 bytes long. The authors of ELEPHANT recommend not changing the variable since it could lead to errors.
- CRYPTO_ABYTES: Allows up to 8 bytes of associated data in both algorithms. The default value of this variable can be increased from 8 to 128 bytes, but the algorithms will increase the execution time. If associated data increases above the default value, the cipher size increases when the ELEPHANT algorithm is used.

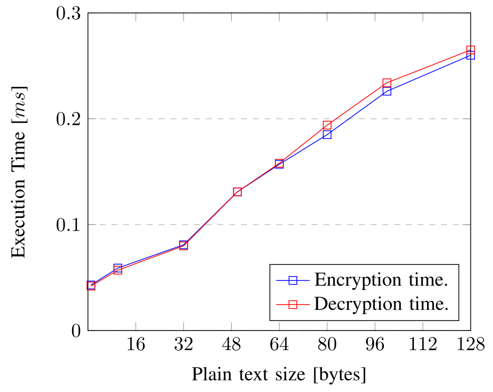

Fig. 5. TinyJAMBU average execution time versus plain text size.

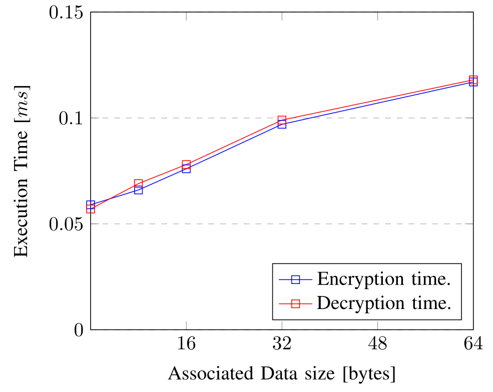

Fig. 6. TinyJAMBU average execution time versus associated data.

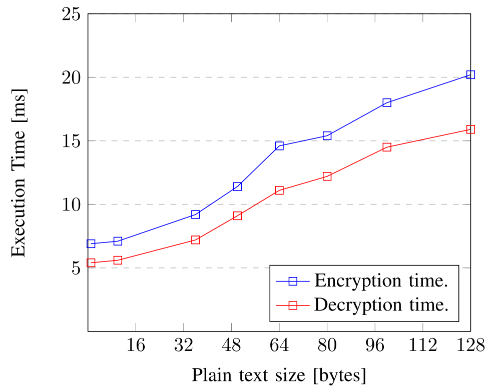

Fig. 7. ELEPHANT average execution time versus plain text size.

In Figures 5 and 6, it is possible to see how the plain text and associated data size affect the execution time on TinyJAMBU in both encryption and decryption processes. The same tests can be seen in Figures 7 and 8 for ELEPHANT. The results were achieved by calculating the average of ten runs in each configuration. Both algorithms come with a default maximum plain text size of 64 bytes. However, it can be changed, and as shown in the graphics, it was tested from 1 to 128 bytes. Both algorithms end up with the same cipher text size, always adding 8 bytes to the original plain text size. TinyJAMBU presents much better results, taking around 0.15 ms to encrypt a message of 64 bytes, while ELEPHANT takes 12 ms. In this case, TinyJAMBU is about 80 times faster than ELEPHANT. This could be due to TinyJAMBU being explicitly targeted

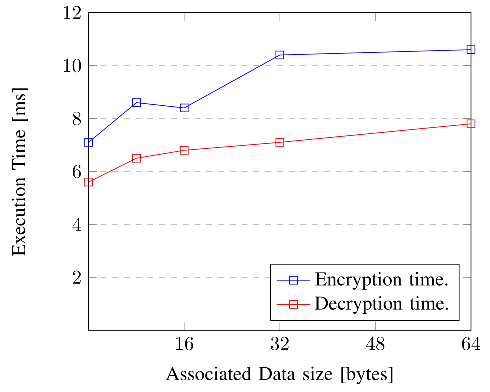

Fig. 8. ELEPHANT average execution time versus associated data.

to hardware, while ELEPHANT is targeted to software and hardware. The estimated execution time on a Raspberry 3 of TinyJAMBU can be calculated depending on the plain text on Equation (2) or the associative data size on Equation (3). Their coefficient of determination (R-Squared) is also presented:

$$\begin{aligned} Time &= 1.6 \times PlainText_{Size} + 0.050\ ms, \\ R^2 &= 0.9909 \end{aligned} \tag{2}$$

$$\begin{aligned} Time &= 0.9281 \times AD + 0.061\ ms, \\ R^2 &= 0.9732 \end{aligned} \tag{3}$$

Both algorithms allow up to 8 bytes of associated data as default. If this variable is kept that way, it does not influence execution time on ELEPHANT as in TinyJAMBU. TinyJAMBU allows up to 128 bytes of associated data, but ELEPHANT only allows up to 64 bytes. Interestingly, especially on Raspberry Pi, the execution time increases proportionally to the associated data in TinyJAMBU. However, it increases logarithmically on ELEPHANT. The estimated execution time on a Raspberry 3 of ELEPHANT can be calculated depending on the plaintext in Equations (4) and (5) or depending on associated data (AD) in Equations (6) and (7):

$$\begin{aligned} Time_{Encrypt} &= 0.1133 \times PlainText_{Size} + 6.2095ms \\ R^2 &= 0.9794 \end{aligned} \tag{4}$$

$$\begin{aligned} Time_{Decryption} &= 0.0902 \times PlainText_{Size} + 4.8346\ ms, \\ R^2 &= 0.9819 \end{aligned} \tag{5}$$

$$\begin{aligned} Time_{Encryption} &= -0.0012 \times AD^2 + 0.1326 \times AD \\ &\quad + 7.1738\ ms, \\ R^2 &= 0.9307 \end{aligned} \tag{6}$$

$$\begin{aligned} Time_{Decryption} &= -0.0005 \times AD^2 + 0.0628 \times AD \\ &\quad + 5.7985\ ms, \\ R^2 &= 0.9435 \end{aligned} \tag{7}$$

Besides the two chosen algorithms, we also evaluated other NIST finalists, such as ASCON [8], GIFT-COFB [10], Grain-128AEAD [11], ISAP [12], Romulus [14] and SPARKLE [15]. We ran the algorithms ten times to obtain the average of those runs, using a plaintext of 32 digits and a 128-bit key, measuring the execution time of the encryption and decryption process. Table IV presents the results in milliseconds.

TABLE IV
NIST LIGHTWEIGHT CRYPTOGRAPHIC ALGORITHMS EXECUTION TIMES FOR RASPBERRY 3B+

Algorithm	Encryption $[ms]$	Decryption $[ms]$
TinyJAMBU [16]	0.01	0.01
ELEPHANT [9]	13.28	12.94
ASCON [8]	1.46	1.14
Grain-128AEAD [11]	9.47	8.81
ISAP [12]	2.37	1.55
Romulus [14]	4.38	4.10
SPARKLE [15]	0.09	0.08

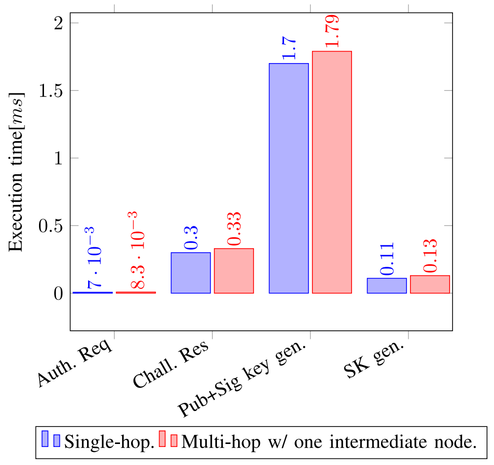

Fig. 9. Node execution time per protocol phase.

B. Light-SAE Evaluation

Light-SAE was evaluated in 2 different scenarios: communication between Nodes and the Gateway directly and communication between Nodes and the Gateway with the help of an intermediate Node. All the results presented were obtained by making ten Nodes join the network until session keys were established to calculate the average execution time of all processes. Encryption, decryption, and times until the devices successfully receive the messages were ignored for these experiments. During the implementation of Light-SAE, all communication was made using TCP/IP communications. Functions to read and write are used to send and receive messages over the network.

To measure the total time needed to establish communication using Light-SAE, we measured each process separately: Authentication request, challenge generation, challenge-response, challenge verification, and public and session key generation in single-hop and multi-hop scenarios.

Figure 9 shows the execution times for all protocol phases when executed in a Node. The most demanding process is calculating the recently joined Node's public value and signature key pair generation. The reason for this is that the signature keys take a long time to generate. The total execution time is about 2.6 ms. From Figure 9, we can conclude that there are no significant differences for the same protocol phase in the single-hop versus multi-hop communications.

Figure 10 shows the Gateway execution time for each phase of the protocol, taking into consideration direct authentication (single-hop authentication) and intermediate Node authentication (multi-hop authentication).

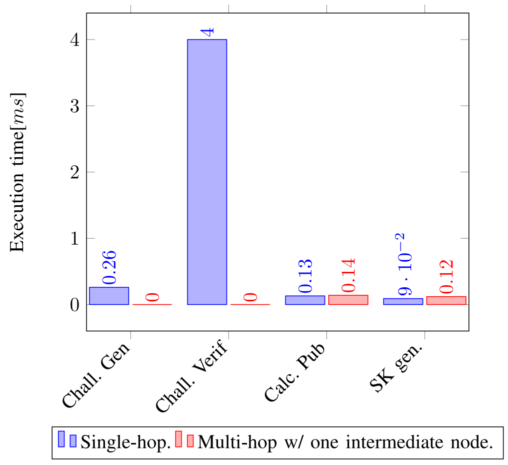

Fig. 10. Gateway execution time per protocol phase.

An intermediate Node is used to allow devices further away from the Gateway to communicate with it and alleviate some computational overhead on the Gateway. As shown in Figure 10, when a multi-hop scenario is considered, the Gateway only has to worry about two protocol phases (calculating public value and session key generation), reducing the possible overhead in networks with a high number of Nodes. This means that the load the Gateway needs to handle is shifted to intermediate Nodes, improving the protocol's scalability. The total execution time is about 0.26 ms. On the other hand, if the Gateway needs to deal with the entire authentication process for all Nodes, the resulting time is about 4.5 ms.

Concerning the session key renewal, it consists of the Node generating a new Diffie-Hellman public value with a new random private key to calculate the new session key. This means that the Node will have to do those two operations while the Gateway has to calculate the session key after receiving the public value from the Node.

VII. Comparison of Light-SAE With Other Existing Approaches

In Table V, we compared Light-SAE and other authentication approaches. Similar to what the authors of [24] did, we have scored with *high*, *medium* or *low* parameters such as resource cost, confidentiality, integrity, scalability, key freshness, capture resistance and forward and backward secrecy. The papers [24], [25], [22], [27] and [28] were already presented in the related work section as well as the reasoning behind their evaluation. As for the schemes on [42], [43], [44] and [45], they are presented and evaluated in more detail in [24].

From the set of features used to make the comparison, we can see that there is no single solution. Each solution must be designed/selected according to the trade-off between resource usage and the desired level of security. The reasons we scored Light-SAE as presented are the following: For Light-SAE, Nodes only have to store at most two keys, the session key established to exchange data with the Gateway and a K_i used for authentication. Light-SAE uses Diffie-Hellman, consisting

TABLE V
SCHEMES COMPARISON OF MAIN FEATURES OF LIGHT-SAE AGAINST OTHER PROPOSALS EXISTING IN THE LITERATURE

Scheme/Reference	Resource cost	Confidentiality	Integrity	Scalability	Key freshness	Capture resistance	Forward and backward secrecy
Light-SAE	Medium	High	Yes	High	Medium	Medium	High
Hao S. [24]	Medium	High	Yes	High	Medium	High	High
Vipin K. [25]	Medium	High	Yes	High	Medium	High	Medium
Mohsen S. [22]	Low	Medium	No	High	High	Medium	No
Kadri B. [27]	Low	Medium	Yes	Medium	Medium	Low	Medium
Chakavarika [28]	Low	Medium	No	High	Medium	Medium	No
Morshed A. [42]	High	Low	Yes	Medium	Low	Medium	Low
Yinghong L. [43]	Medium	Low	Yes	High	Low	Medium	Low
Zhou J. [44]	Low	Medium	Yes	Medium	Medium	Low	High
Bowen S. [45]	Low	Medium	Yes	High	High	Low	High

of extensive operations. These take a bit more resources than schemes that distribute the keys. However, since the generated keys are 256 bits, it brings neither much time nor resources compared to other methods, with the benefit of not passing session keys through the network.

As for confidentiality, the Gateway authenticates the sensors by comparing several parameters known only to valid Nodes. Authentication parameters are calculated randomly by the Gateway. Eavesdroppers or man-in-the-middle attacks are prevented by encrypting the Diffie-Hellman parameters that go through the network. Furthermore, if a key somehow got captured, it would only affect a single Node since each has a unique session key with the Gateway.

The solution ensures the integrity of the messages with the usage of the digital signature algorithm. If a message gets captured during transmission, an attacker can not modify or tamper with its contents without the Gateway noticing since the verification will fail.

Nodes can join the network as long as the authentication process runs smoothly. Authentication may happen directly with the Gateway (if near it) or using an intermediate authenticated Node (if out of range of the Gateway), ensuring a reasonable degree of scalability.

The session keys get refreshed from time to time. The Node should initialize the process. However, since the Gateway stores a timestamp when the session key was established if a Node does not initiate the process, the Gateway itself will request a session key renewal. If the Node gives no response, it gets removed from the network.

As for capture resistance, if a Node shuts down, restarts updates, or gets replaced, it has to go through the authentication process. Only Nodes with all the correct parameters can pass this stage. This ensures that no outsider can capture a Node and take possession of it while remaining in the network.

Finally, the proposed protocol ensures forward and backward secrecy. Once a Node rejoins the network, it must be authenticated and establish a new key.

We have comprehensively compared Light-SAE and the most important solutions already published in the literature (Table V). This analysis highlights the main features of Light-SAE, considering resource cost, confidentiality, integrity, scalability, key freshness, capture resistance, and forward and backward secrecy. It is clear that Light-SAE has advantages compared with others. In detail, our solution is modular oriented to be adjusted to fit resources available at the Nodes. Concerning confidentiality and integrity, Light-SAE is as good as the encryption and signature schema used. Regarding scalability, the solution was designed to be distributed, e.g., an existing leaf Node (already authenticated) can authenticate new Nodes avoiding overprocessing at the Gateway. Upon establishing a session key, the time for renewal is an adjustable parameter of the solution, which also provides ephemeral keys that ensure capture resistance and forward and backward secrecy.

VIII. CONCLUSION

Most devices in IoT applications do not use secure encryption or authentication algorithms in their communication. Failure to use these security methods can lead to severe consequences, such as unauthorized access and manipulation of data. The work focuses on proposing and developing a lightweight key distribution solution for WSN called Light-SAE. It supports multi-hop authentication and communication between resource-constrained devices and a Gateway. Nodes can authenticate and generate a common key with a Gateway, even if out of its reach. We use the latest NIST lightweight cryptographic algorithms to encrypt all the data and parameters that go through the network, in particular for the payload used to exchange the information needed to generate the common key. However, any encryption algorithm can be used with Light-SAE as long as the key size matches the ones defined, which in our case, was 256 bits. Encrypted Diffie-Hellman is used to generate the 256-bit session keys unique to every Node and a renewal mechanism to update them after a necessary amount of time. A modified version of Schnorr's algorithm was used to generate the signatures and verify them to ensure no modification. With all these mechanisms, we have achieved an excellent key distribution solution that provides authenticity, confidentiality and integrity. We provide experimental results that prove how fast Light-SAE processes are. A detailed comparison of the protocol was also made with similar protocols in a diverse set of parameters.

Since the Light-SAE solution presents promising outcomes regarding key distribution, authentication, data encryption and integrity, further research is necessary to evaluate its scalability and performance in more extensive and complex networks with heterogeneous devices and sub-systems. Light-SAE operates only at the application layer, manipulating only the payload. For the cases where the payload must be divided into multiple packets, a possible application of Light-SAE as a communication protocol requires further investigation to ensure correct content recovery. Light-SAE was designed to be scalable. However, it assumes a star network topology where every Node intends to exchange secure information with the Gateway. In order to enhance its scalability, more research is needed to adapt it to work in hierarchical networks based on cluster architecture. In this particular scenario, each authenticated Node can act similarly to the Gateway, providing functionalities, such as defining a new session key. In addition, with the evolving threat landscape, it is crucial to keep Light-SAE up-to-date with the latest security standards to ensure it can effectively protect against emerging security threats. Since Light-SAE follows a modular approach, ongoing evaluation of the Light-SAE solution will help to enhance its robustness and effectiveness, allowing the development and replacement of specific modules, targeting specific levels of security and privacy in IoT ecosystems.

References

[1] P. Asghari, A. Rahmani, and H. Javadi, "Internet of Things applications: A systematic review," *Comput. Netw.*, vol. 148, pp. 241–261, Jan. 2019.

[2] Upasana. "Real world IoT applications in different domains." 2022. Accessed: May 14, 2022. [Online]. Available: https://www.edureka.co/blog/iot-applications/

[3] Y. Pinar, A. Zuhair, A. Hamad, A. Resit, K. Shiva, and A. Omar, "Wireless sensor networks (WSNs)," in *Proc. IEEE Long Island Syst. Appl. Technol. Conf. (LISAT)*, 2016, pp. 1–8.

[4] Techjury. "How many IoT devices are there in 2021?" 2021. Accessed: Oct. 13, 2021. [Online]. Available: https://techjury.net/blog/how-many-iot-devices-are-there/

[5] A. Fotovvat, G. Rahman, S. Vedaei, and K. Wahid, "Comparative performance analysis of lightweight cryptography algorithms for IoT sensor nodes," *IEEE Internet Things J.*, vol. 8, no. 10, pp. 8279–8290, May 2021.

[6] C. Fernandes, *Choosing the Future of Lightweight Encryption Algorithms*. IST, New Delhi, India, 2018.

[7] NIST. "Lightweight cryptography." 2017. Accessed: Sep. 24, 2021. [Online]. Available: https://csrc.nist.gov/projects/lightweight-cryptography

[8] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, *ASCON V1.2*, NIST, Gaithersburg, MD, USA, May 2021.

[9] T. Beyne, Y. Chen, C. Dobraunig, and B. Mennink, *ELEPHANT V2*, NIST, Gaithersburg, MD, USA, May 2021.

[10] S. Banik et al., *GIFT-COFB v1.1*, NIST, Gaithersburg, MD, USA, May 2021.

[11] M. Hell, T. Johansson, A. Maximov, W. Meier, J. Sonnerup, and H. Yoshida, *Grain-128AEADv2—A Lightweight AEAD Stream Cipher Cover Sheet*, NIST, Gaithersburg, MD, USA, 2019.

[12] C. Dobraunig et al., *ISAP v2.0*, NIST, Gaithersburg, MD, USA, 2019.

[13] Z. Bao et al., *PHOTON-Beetle Authenticated Encryption and Hash Family*, NIST, Gaithersburg, MD, USA, May 2021.

[14] C. Guo, T. Iwata, M. Khairallah, K. Minematsu, and T. Peyrin, *Romulus v1.3*, NIST, Gaithersburg, MD, USA, 2019.

[15] C. Beierle et al., *Schwaemm and Esch: Lightweight Authenticated Encryption and Hashing Using the Sparkle Permutation Family*, NIST, Gaithersburg, MD, USA, May 2021.

[16] H. Wu and T. Huang, *TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms*, NIST, Gaithersburg, MD, USA, Mar. 2019.

[17] J. Daemen, S. Hoffert, S. Mella, M. Peeters, G. Assche, and R. Keer, *Xoodyak, a Lightweight Cryptographic Scheme*, NIST, Gaithersburg, MD, USA, May 2021.

[18] C. Bormann, M. Ersue, and A. Keränen, "Terminology for constrained-node networks," IETF, RFC 7228, May 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7228

[19] J. Cecólio and A. Casimiro. "AQUAMON-dependable monitoring with wireless 1095 sensor networks in water environments." 2018. Accessed: Oct. 10, 2021. [Online]. Available: https://aquamon.di.fc.ul.pt/news-events/ News & Events - aquamon.di.fc.ul.pt

[20] M. Yousefpoor and H. Barati, "Dynamic key management algorithms in wireless sensor networks: A survey," *Comput. Commun.*, vol. 134, pp. 52–69, Jan. 2019.

[21] S. Szymoniak and S. Kesar, "Key agreement and authentication protocols in the Internet of Things: A survey," *Appl. Sci.*, vol. 13, no. 1, p. 404, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/1/404

[22] M. Sharifi, S. Ardakani, and S. Kashi, "SKEW: An efficient self key establishment protocol for wireless sensor networks," in *Proc. Int. Symp. Collaborative Technol. Syst.*, 2009, pp. 250–257.

[23] M. Messai, H. Seba, and M. Aliouat, "A lightweight key management scheme for wireless sensor networks," *J. Supercomput.*, vol. 71, pp. 4400–4422, Dec. 2015.

[24] H. Shi, M. Fan, Y. Zhang, M. Chen, X. Liao, and W. Hu, "An effective dynamic membership authentication and key management scheme in wireless sensor networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2021, pp. 1–6.

[25] V. Kumar, N. Malik, G. Dhiman, and T. K. Lohani, "Scalable and storage efficient dynamic key management scheme for wireless sensor network," *J. Ambient Intell. Humanized Comput.*, vol. 2021, Jul. 2021, Art. no. 5512879. [Online]. Available: https://doi.org/10.1155/2021/5512879.

[26] W. Bechkit, Y. Challal, and A. Bouabdallah, "A new class of hash-chain based key pre-distribution schemes for WSN," *Comput. Commun.*, vol. 36, no. 3, pp. 243–255, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366412003386

[27] B. Kadri, D. Moussaoui, M. Feham, and A. Mhammed, "An efficient key management scheme for hierarchical wireless sensor networks," *Wireless Sensor Netw.*, vol. 4, no. 6, pp. 155–161, 2012.

[28] T. Chakavarika, S. Gupta, and B. Chaurasia, "Energy efficient key distribution and management scheme in wireless sensor networks," *Wireless Pers. Commun.*, vol. 97, no. 1, pp. 1059–1070, Nov. 2017. [Online]. Available: https://doi.org/10.1007/s11277-017-4551-2

[29] R. Vinoth, L. J. Deborah, P. Vijayakumar, and B. B. Gupta, "An anonymous pre-authentication and post-authentication scheme assisted by cloud for medical IoT environments," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 5, pp. 3633–3642, Sep./Oct. 2022.

[30] L. Zhang, J. Xu, M. Obaidat, X. Li, and P. Vijayakumar, "A PUF-based lightweight authentication and key agreement protocol for smart UAV networks," *IET Commun.*, vol. 16, no. 10, pp. 1142–1159, 2022. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cmu2.12295

[31] A. Yavuz and M. Ozmen. "Ultra lightweight multiple-time digital signature for the Internet of Things devices." 2019. [Online]. Available: https://arxiv.org/abs/1907.03911

[32] C. Costello and P. Longa. "SchnorrQ: Schnorr signatures on fourQ." Jul. 2016. [Online]. Available: https://www.microsoft.com/en-us/research/publication/schnorrq-schnorr-signatures-fourq/

[33] C. Costello and P. Longa, "FourQ: Four-dimensional decompositions on a Q-curve over the mersenne prime," in *Proc. 21st Int. Conf. Theory Appl. Cryptol. Inf. Security (ASIACRYPT)*, Dec. 2015, pp. 1–25.

[34] D. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang. "High-speed high-security signatures." 2011. [Online]. Available: https://eprint.iacr.org/2011/368

[35] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[36] S. Ali et al., "An efficient cryptographic technique using modified Diffie–Hellman in wireless sensor networks," *Int. J. Distrib. Sensor Netw.*, vol. 16, no. 6, p. 19, 2020. [Online]. Available: https://doi.org/10.1177/1550147720925772

[37] T. Kivinen and M. Kojo, "More modular exponential (MODP) Diffie–Hellman groups for Internet key exchange (IKE)," IETF, RFC 3526, 2003.

[38] D. Zagier, "Newman's short proof of the prime number theorem," *Amer. Math. Monthly*, vol. 104, no. 8, pp. 705–708, 1997. [Online]. Available: http://www.jstor.org/stable/2975232

[39] H. Wu and J. Zhuang, "Improving the Gaudry–Schost algorithm for multidimensional discrete logarithms," *Designs Codes Cryptography*, vol. 90, no. 1, pp. 107–119, Jan. 2022. [Online]. Available: https://doi.org/10.1007/s10623-021-00966-5

[40] L. Bošnjak, J. Sres, and B. Brumen, "Brute-force and dictionary attack on hashed real-world passwords," in *Proc. 41st Int. Conv. Inf. Commun. Technol. Electron. Microelectron. (MIPRO)*, 2018, pp. 1161–1166.

[41] M. Badawy, A. El-Aziz, A. Idress, H. Hefny, and S. Hossam, “A survey on exploring key performance indicators,” *Future Comput. Inf. J.*, vol. 1, no. 1, pp. 47–52, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2314728816300034
[42] A. Aski, H. Javadi, and G. Shirdel, “A full connectable and high scalable key pre-distribution scheme based on combinatorial designs for resource-constrained devices in IoT network,” *Wireless Pers. Commun.*, vol. 114, pp. 2079–2103, Oct. 2020.
[43] Y. Liu and Y. Wu, “A key pre-distribution scheme based on sub-regions for multi-hop wireless sensor networks,” *Wireless Pers. Commun.*, vol. 109, no. 2, pp. 1161–1180, Nov. 2019.
[44] Z. Jian, S. Liyan, D. Kaiyu, and W. Yue, “Research on self-adaptive group key management in deep space networks,” *Wireless Pers. Commun.*, vol. 114, pp. 3435–3456, Oct. 2020.
[45] B. Sun, Q. Li, and B. Tian, “Local dynamic key management scheme based on layer-cluster topology in WSN,” *Wireless Pers. Commun.*, vol. 103, pp. 699–714, Nov. 2018.

Pedro Rosa was born in Lisbon, Portugal, in 1999. He received the B.S. degree in computer engineering from Universidade Europeia in 2020. He is currently pursuing the M.S. degree in computer engineering with the Faculdade de Ciências, Universidade de Lisboa.

André Souto received the Ph.D. degree from the University of Porto in 2011. He is currently an Assistant Professor with the Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, where he is an Integrated Member of LASIGE.

José Cecílio received the Ph.D. degree from the University of Coimbra in 2013. He is currently an Assistant Professor with the Faculty of Sciences, University of Lisbon, where he is an Integrated Researcher with LASIGE. His research interests relate to the Internet of Things, adaptive and safety-critical CPS, and reliable and energy-efficient systems.