ASSURE: A Hardware-Based Security Protocol for Resource-Constrained IoT Systems

Yildiran Yilmaz^{1,2} • Leonardo Aniello² • Basel Halak²

Received: 9 September 2019 / Accepted: 20 August 2020 / Published online: 4 September 2020
© The Author(s) 2020

Abstract

The Internet of Things technology is expected to generate tremendous economic benefits; this promise is undermined by major security threats. This is mainly due to the ubiquitous nature of this technology, which makes it easy for potential adversaries to have access to IoT devices and carry well-established attacks. The development of defence mechanisms, in this case, is a challenging task; this is due to the fact that most IoT devices have limited computing and energy resources, which makes it hard to implement classic cryptographic algorithms. This paper addresses this challenge by proposing a lightweight mutual authentication and key agreement protocol named ASSURE based on Rivest Cipher (RC5) and physically unclonable functions (PUFs). To understand the effectiveness of this protocol, a rigorous security analysis under various cyber-attack scenarios is performed. In order to evaluate the overheads of the proposed solution, a wireless sensor network using typical IoT devices called Zolertia Zoul re-mote is constructed. The functionality of the proposed scheme is verified using a server-client configuration. Then, energy consumption and memory utilization are estimated and compared with the existing solutions, namely, the DTLS (datagram transport layer security) handshake protocol in pre-shared secret (key) mode and UDP (user datagram protocol). Experimental analysis results indicate that the proposed protocol can save up to 39.5% energy and uses 14% less memory compared with the DTLS handshake protocol.

Keywords Mutual authentication · Internet of Things · Wireless sensor · Physically unclonable functions · Rivest Cipher

1 Introduction

With recent developments in Internet of Things systems, it has become possible to design and develop multifunctional sensor nodes that are small in size, come at a low cost, require little power and communicate wirelessly in short distances [1]. The aim of the wireless sensor network is to detect physical and environmental changes such as temperature, sound, pressure and humidity. The sensor should transmit the sensed data to a central hub in cooperation over the established network. These networks have a wide application area in many sectors,

including military, environmental, health, industrial and smart homes. It is worth considering patient monitoring systems as a concrete example of this technology in use [1]. With this system, patient pulse rate, blood oxygen level, electrical activity in the heart, muscle activation and general kinesthetic movements may be measured. This makes it possible to monitor the patient's health remotely and intervene in case of an emergency. Taking this kind of IoT system into consideration, authentication and secure communication are crucial for ensuring that the health readings gathered are trustworthy so as to prevent potential attackers from falsifying that information for the purpose of jeopardizing the health of the patient. However, the traditional cryptographic solutions [2] designed for resource-rich devices to the security are often impractical for the resource-constrained devices used in these kinds of IoT systems. This is because the wireless sensor devices are limited in memory, computation ability and energy resource. These devices make the task of establishing security protocols difficult as they dedicate most of the available resources to perform their basic tasks. Several approaches have been taken into consideration in order to solve the security puzzle for such constrained devices. Some of these approaches [1, 3, 4] have

✉ Yildiran Yilmaz
yy6e14@soton.ac.uk; yildiran.yilmaz@erdogan.edu.tr

Basel Halak
bh9@soton.ac.uk

¹ Computer Engineering, University of Recep Tayyip Erdogan, Rize, Turkey

² Electronics and Computer Science, University of Southampton, Southampton, UK

built further levels of security on top of those devices' constrained application protocol (CoAP) by using DTLS (datagram transport layer security). Some [5–8] have even taken a completely different route by using physically unclonable functions (PUFs) because PUFs provide physical security against invasive attacks [9]. Furthermore, PUFs are increasingly becoming a vital security tool in the Internet of Things environments seeing as they have many uses in the field of security protocols and key generation [6, 10–15].

Existing PUF-based protocols introduced in [10–12, 14] still have three main limitations. The first limitation is the need for storing a large number of challenge/response pairs for each PUF at a central verifier for subsequent verification. This requirement makes this technology hard to develop in large networks. Secondly, there is a clear lack in the literature such as [13, 16] of proper evaluation of the cost associated with the PUF protocols compared with the existing non-PUF solutions (e.g. DTLS). The lack of such analysis is a major obstacle that prevents wider adoption of this technology. The protocols analysed in [17, 18] and the classical PUF-based authentications mentioned in [12, 19] have tended to focus on physical security rather than security analyses against protocol attacks. Another limitation is that they do not provide complete privacy against common protocol attacks, e.g. man in the middle [10].

The contributions of this paper are summarized below:

- 1.) A lightweight mutual authentication along with peer-to-peer key agreement protocol, called ASSURE, based on RC5 and PUF cryptographic primitives is developed, which requires less computing resources and consumes less energy.
- 2.) A systematic security analysis is carried out to validate the proposed solution.
- 3.) The proposed solution is implemented in a constrained IoT, i.e. Zoul device, and rigorous cost estimation is performed, and the estimated results reveal that the ASSURE protocol can save up to 39.5% energy and uses 14% less memory compared with the DTLS handshake protocol which serves a pre-shared secret (key)-based authentication.

The rest of this paper is organized as follows. Section 2 gives an overview of related work and relevant literature. Section 3 describes the proposed solution. In Section 4, a systematic security analysis is presented, and the proposed solution is compared with the previous PUF protocols in terms of their security properties. Section 5 then describes the implementation process of the hardware demonstrator. A detailed cost analysis and discussion are presented in Section 6. Lastly, conclusions are drawn in Section 7.

2 Background

The purpose of this section is to provide an overview of the related works on security protocols for resource-constrained environments and provide background information about the subject of this paper. Thus, this section begins with related work and then discusses the PUF technology and classical PUF-based authentication protocols. Subsequently, this section discusses the pre-shared key-based DTLS protocol used in this paper, considering that one of the implementations is based on the PSK-DTLS handshake protocol.

2.1 Related work

In light of the recent IoT networks containing resource-constrained devices, there is a considerable concern about how these devices will be connected to each other and how a secure authentication protocol will be established in an energy-efficient way [20]. The reason for this concern is that security solutions for traditional resource-rich networks will not work in the current and next-generation IoT networks due to limited available energy, processing ability and storage capacity of the constrained devices.

Conventional cryptographic protocols reported in [21–24] which require an encryption key to be stored on the device memory aim to solve the security issue of constrained devices [9]. The authors in [21] proposed a new secure and energy-efficient communication protocol for CoAP (constrained application) devices by employing a strong public-key cryptography. The protocol proposed in [23] uses the DTLS approach to provide security with TLS-RSA-AES-128-CBC-SHA1 cipher suite. The SPIN (Sensor Protocol for Information via Negotiation) reported in [25] is the first protocol used to provide data confidentiality and authentication. It is the first security protocol created for resource-constrained environment, e.g. wireless sensor network. It comprises of two secure building blocks termed as μ TESLA and SNEP. Currently, the SNEP has not been completely detailed and implemented [26]. Among the authentication protocols mentioned, SPIN is slightly subpar/dated for today's system requirements, as suggested in [26]. The rest of the protocols can be employed to provide the application dependent security, although not all of them are cost-effective for resource-constrained networks, especially the protocols relying on the use of RSA public-key cryptography such as [23].

As an alternative to conventional approaches, PUF-based protocols have been gaining much attention in IoT domains due to the security properties discussed in [9]. The PUF-based authentication methods proposed in [12, 17–19] consist of two processes: enrolment, which is performed before the authentication by a verifier, and verification, which ensures the authentication. As discussed in [10], though, during the verification process, the transparent exchange of challenges and

responses between the verifier and the prover reveals serious privacy concerns with relation to non-invasive attacks. Elsewhere, the PUF-based authentication proposed in [27] requires CRP storage for each PUF-enabled node. The work in [28] achieves authentication without storing explicit challenge-response pairs in the verifier database. However, this protocol uses a public-key cryptography which requires heavy computations than a symmetric key cryptography. Therefore, it is implemented on an Intel Edison device with a Nexys 4 FPGA board equipped with 16 MB CellularRAM which is a resource-rich device. There are also two additional facilities called the security association provider and security credential generator in [28]. This makes the network much more complex and increases the cost. Consequently, there is no energy consumption evaluation in both [27, 28]; this prevents a wider adoption of PUF-based protocols in energy-constrained environments. Our protocol does not require CRP storage and additional facilities and uses a light symmetric key cryptography. This paper also makes resource consumption evaluation for the proposed protocol.

2.2 Physically unclonable functions

Physically unclonable functions (PUFs) are physical random functions that provide specific outputs for the physical objects they work in [9], as illustrated in Fig. 1. Therefore, authentication mechanisms can be developed by taking advantage of PUF outputs [5, 6, 11].

PUFs are named into two groups as the so-called strong and weak, according to the bit length of challenge and responses [9]. In particular, practising a strong-PUF is a difficult task. The reason is that when PUFs generate responses, the noises make it very difficult to balance response bits. In some studies [12, 14], error-correction codes (ECC) with the helper data have been used for resolving this problem [29]. Latterly, the authors in [29] have proposed the structure of the PUF-FSM, which generates reliable outputs. They found a solution for removing the need for ECC and helper data by using error-free responses.

Fig. 1 PUF cryptographic primitive

In fact, PUF design is not the focus of this paper; the benefits and importance of contributions proffered in this paper lie at the protocol level. Therefore, the use of the PUF-FSM within ASSURE is assumed. Thus, consistent response corresponding with a challenge can be generated in a controlled and reliable manner.

2.3 DTLS protocol in resource-limited environments

The DTLS design was originally intended to provide security in traditional IoT networks containing both resource-rich and resource-constrained devices [23]. Hence, using this protocol in its original form on resource-limited devices means a heavy-duty solution because of the computation and communication requirements [1, 3]. DTLS in default mode uses RSA-based certificates, which requires heavy computation, and six communication flights between the client and server [3]. Due to the necessity of minimizing resource use in the constrained IoT, Raza et al. [1, 30] suggested adopting 6LoWPAN header compression for the DTLS. For the same reason, Kothmayr et al. [23] proposed an RSA-based security protocol called “certificate based DTLS”. However, the measured overhead of this DTLS handshake indicates high energy usage because of asymmetric cryptosystem usage. In other works [31, 32], the authors estimated the performance of the DTLS handshake, for resource-constrained networks using ECC-based cryptography, which was proposed for the CoAP [33]. The measurements, however, display the fact that energy consumption is still quite high. In this paper, PSK (pre-shared key) mode [34], a low-cost application of DTLS, was implemented to minimize the computation and communication costs.

3 Proposed protocol

This section begins with the system model and attacker model for understanding the server/device roles and rationale behind the proposed protocol and provides a brief introduction of the proposed solution. It then describes in detail the security protocol, designed for wireless sensor devices, which uses PUF technology and the light cryptographic algorithm.

3.1 System model

In the system model depicted in Fig. 2, a number of parties execute a generic authentication and key agreement protocol by exchanging messages with the aim of proving each other’s identity. The protocol specification defines what roles are involved in each execution of the protocol itself. For example, an authentication protocol usually includes a prover role for the party to be authenticated and a verifier role for the party verifying the prover’s identity. The specification also defines

Fig. 2 System model for resource-constrained IoT devices

what messages are sent and received in which order, the expected content of messages, and the state of each involved role, i.e. what the variables are in the local memory of each role and how they change upon receiving new messages.

There can be more instances of the same protocol being executed at the same time, and the same party can participate in more instances simultaneously. With reference to the previously mentioned ASSURE authentication protocol, a party can run the verifier role into two distinct instances with two different parties; each party playing the prover role as shown in Fig. 2. During peer-to-peer key agreement, the two provers, which have been authenticated by the verifier, exchange a secret key for secure communication. Finally, all the above statements about the system model lead to define the attacker models the system might face, as described in the following section.

3.2 Attacker model

In the attack scenario, an attacker is a party potentially playing several roles in different instances of the same authentication and key agreement protocol. This party attempts to deceive honest parties by behaving in compliance with the protocol with the aim of breaking security properties. Relevant examples of attack strategies are described below.

In a man in the middle (MitM) attack, the attacker operates in between two honest parties that are running an instance of the protocol. In this type of attack, the attacker can intercept, alter and relay the messages exchanged by the honest parties without them being aware of the intrusion. By operating in the middle, secretly and with the possibility of modifying to some extent the content of intercepted messages, the attacker can break the protocol.

In an eavesdropping attack, the data transmitted over a network or channel is listened without permission by malicious third parties to gain access to private information. It is possible that the sensitive data, e.g. key, can be obtained and exploited for malicious purposes.

In a replay attack, the attacker first picks valid messages exchanged by honest parties in some instance of the protocol

and then reuses these messages in a different instance to bypass identity controls and pose as a legitimate party.

In a model building-attack, the attacker first tries to imitate the PUF behaviour with some machine learning techniques. He/she exploits machine learning algorithms to predict the responses to the new challenges using a certain number of known challenge and response pairs (CRPs), obtained from the communication channel during the process of running the protocol. Based on the CRPs obtained, the attacker trains the ML algorithms to provide responses to be involved in the authentication. Finally, the attacker attempts to make his/her device authentic by modelling the protocol.

In a device impersonation attack, the adversary attempts to deceive an authenticator server into accepting a fake device as valid. In a cloning attack, an attacker tampers with a device to retrieve secrets in the device. He/she then stores the collected secrets into a fraudulent device. The same design and secrets make it possible to be involved in the system as a valid device.

In a server impersonation attack, a powerful adversary who aims to obtain the internal state of a device tries to emulate the current server and portray itself as a valid server.

In a de-synchronization attack, which is a sort of denial of service attack, an adversary intentionally disrupts the synchronization of the server with the device to block subsequent verifications. The authentication protocols in some IoT devices, such as RFID, carry out an update process for the secrets and identity of the device during successful protocol operation. Since this update is processed on both back-end server and processing device, the synchronization of confidential data between the device and the verifier database is particularly critical for future authentication attempts.

3.3 Proposed solution

The proposed solution combines PUF technology with a symmetric light cipher named RC5 to produce a new protocol that has the same security properties of DTSL against protocol attacks while at the same time consuming less energy. In addition to this, it provides security against physical attacks thanks to PUF cryptographic primitive [35].

This work is a preliminary attempt seeing as it compares the costs associated with the PUF protocol (ASSURE) and the existing non-PUF solutions (e.g. DTLS). The ASSURE protocol proposed in this work also satisfies the following requirements:

- **Mutual authentication:** Both parties (client and server) involved in the network must authenticate each other in an energy-efficient way.
- **Peer-to-peer key agreement:** Two IoT devices authenticated by the same server must be able to agree on the key to ensuring secure communication with each other.

- Client unclonability: When all the design features of the IoT device are obtained by the attacker, the security of the system should not be able to be compromised by means of cloning the device.
- PUF model on the verifier: The trained PUF model is used at the server side to reduce the cost of storing a large number of challenge/response pairs.

3.4 Protocol description

The properties of the authentication and key agreement model are the system manager, represented by S . The involved objects in the model are the verifier, represented by V , and devices A and B (i.e. provers), represented by D_A and D_B . In implementing this scheme, three devices are connected to each other using the UDP connection, one as a server (verifier) and the other two as provers. This model allows mutual authentication between two entities (V, D) and peer-to-peer key agreement between D_A and D_B . Party V includes PUF models based on a neural network algorithm for each device enrolled in the system by the system manager S .

The proposed protocol (ASSURE): The protocol consists of three phases, namely, the registration phase, the verification phase and the peer-to-peer key agreement phase. For these phases, the protocol is explained using the symbols in Table 1.

The registration and verification phases of the protocol consist of the following steps.

- a) *Registration phase*: As a one-time effort, the verifier (V) creates PUF models, i.e. $PUFM_A$ and $PUFM_B$, using the

machine learning algorithm (neural network) and stores them with the MAC address and IDs of devices (A and B) into the database, e.g. $DB = (ID'_A, MAC_A, PUFM_A)$. It should be noted that the PUF will be unique by associating it with the ID and MAC address of the client. The authorized MAC addresses of the clients corresponding to the model should be kept in the verifier database to make sure that the client is a device belonging to the network.

- b) *Authentication phase*: The first three flights in Fig. 3 are for device-server mutual authentication, and the fourth and last flight is for device-device key agreement. The authentication processes of devices A and B are carried out with similar actions, with the exception of the third flight only. This section describes the authentication steps of device A in the following manner:

Step 1. The verifier generates a random number r and $nonce_{v_1}$ $0 \leq r \leq 2^k$, $r \in R$, normalizes the challenge $C_i = r$, $0 < i \leq 2^n$, $i \in Z^+$ and sends the challenge (C_i) and $nonce_{v_1}$ to device A .

Step 2. Device A receives the challenge (C_i) and generates the response $R_i^A = PUF_A(C_i)$. Afterwards, device A computes $\alpha_d = RC5_{R_i^A}(M)$ where $M = ((ID_B), MAC, nonce_d, nonce_{v_1})$ and sends it along with its ID to the verifier.

Step 3. Device A is checked by looking at its ID in the database of the verifier. The verifier computes the response $R_i^A = PUFM_A(C_i)$. Then the verifier decrypts α_d and obtains $(ID_B), MAC, nonce_{v_1}$ and $nonce_d$. If ID_B exists in message (α_d), that means that device A wants to communicate with device B . The verifier checks the authenticity of the MAC from the database and verifies the $nonce_{v_1}$. The freshness control is based on $nonce_{v_1}$. This implies that the received information is recent. If those checks are successful and the data is fresh, the device is authenticated; otherwise, the device is rejected and unauthorized.

Step 4. After the authentication of device A has been successfully completed, the verifier computes and sends $\alpha_v = RC5_{R_i^A}(MAC_A, nonce_d)$ to be authenticated by A . The token (Tkn_i) is generated by the verifier such that $Tkn_i = RC5_{R_i^B}(ID_A, MAC_B, T_p, S_{key})$. V then generates a random $nonce_{v_2}$ and sends it with Tkn_i in an encrypted form to A such that $M_{SA} = RC5_{R_i^A}(MAC_A, S_{key}, Tkn_i, nonce_{v_2})$.

Transmitting M_{SA}^* containing Tkn_i is optional and occurs when device A wants to communicate with device B .

Step 5. The device decrypts α_v and checks MAC_A . If the check is successful and $nonce_d$ is fresh, the server (verifier) is authenticated.

Table 1 List of symbols

Parameter	Description
ID'	The device's identifier
V	The verifier to which the device wants to connect (server)
k	The bit length of the PUF. N.B.: arbiter and TCO-PUF [36] are 32 bits
r	A random number generator defined in the set Z_n^+
$nonce$	Random number
C, R	Challenge and response
$RC5$	Rivest Cipher
$RC5_{key}$	Subscript of RC5 indicates the key used (128 bits)
DB	Database
T_p	The time period in which the token (Tkn) may live
S_{key}	Symmetric key to be agreed and utilized for creating a session between D_A and D_B
Tkn	Token for sharing S_{key} between D_A and D_B
M	Message containing confidential data

Fig. 3 The steps of the proposed authentication and key agreement protocol

Heretofore, the mutual authentication between device A and the verifier V has been described. In fact, the mutual authentication steps between B and V are processed in a similar way as indicated in authentication between A and V. Thus, the authentication steps of device B are described briefly in the following manner:

Step 6. The verifier generates and sends a random number $nonce'_{v1}$ and normalized challenge C'_i to device B.

Step 7. Device B generates the response $R_i^B = PUF_B(C'_i)$ and then computes α'_d and sends it along with its ID' to the verifier.

Step 8. Device B is checked by looking at its ID' in the database of V. Afterwards, V computes the response $R_i^B = PUF_B(C'_i)$ and then decrypts α'_d and obtains $MAC, nonce'_{v1}$ and $nonce'_d$. V checks the authenticity of the MAC from the database and verifies the $nonce'_{v1}$.

Step 9. If those checks are successful, V computes and sends $\alpha'_v = RC5_{R_i^B}(MAC_B, nonce'_d)$ to B.

Step 10. Device B decrypts α'_v and checks MAC_B and $nonce'_d$ and then authenticates the server.

Peer-to-Peer key agreement phase At the key agreement stage, device A communicates with device B in the following manner to exchange the key. It should be noted that depending on the applications and network requirements, the peer-to-peer key agreement is optional and happens when device A wants to communicate securely with device B because B would be a service provider or resource provider.

Step 11. Device A generates a request ($rqst$) and a random nonce (n_A) and sends them with ID_A, ID_B in an encrypted

form in a message such that $M_{AB} = RC5_{S_{key}}(ID_A, ID_B, n_A, rqst)$, along with Tkn_i to device B.

Step 12. Device B obtains R_i^B , which is already used in the authentication phase, using its PUF_B . B decrypts the Tkn_i using R_i^B and obtains the secret key S_{key} and MAC_B and then verifies MAC_B . It then decrypts the M_{AB} using the secret symmetric key S_{key} and obtains the ID_A, ID_B, n_A and $rqst$. If obtaining S_{key} and decrypting M_{AB} are successful, then device B verifies whether ID_B, MAC_B are correct or not. If those verifications fail, B terminates the process. Otherwise, the key agreement is considered complete, and device B can communicate and share resources securely with device A using S_{key} . Note that in the case of reconnection, device B will receive the same challenge from the server and generate the same response during the current authentication and key agreement session. Consequently, B knows which response to use to decrypt the token received from A.

Thus far, the details on the proposed solution and the ASSURE protocol description are provided in this section. The following section analyses in detail the security of the protocol mentioned above.

4 Security analysis

This section describes the security analysis of ASSURE for the purpose of validating the proposed solution. First, a system model and an attacker model are characterized to define the security properties provided by ASSURE, namely, mutual authentication, resistance to man in the middle and eavesdropping attacks, resistance to replay attacks, resistance to device impersonation and cloning, server impersonation

and de-synchronization attacks. Then, the way by which these properties are proved is explained by using a well-known automatic tool for the analysis of security protocols named Scyther. This section moreover analyses the security of ASSURE against model building attacks (ML attacks) based on SVM and ANN implementations.

4.1 Scyther tool

Scyther is a publicly available tool responsible for formally analysing security protocols. It is assumed that all cryptographic algorithms are fault-free, i.e. an attacker cannot derive any information from encrypted messages without having the encryption key. In order for protocols to be verified, they must be defined using a custom C-like programming language, as advised in Section 4.3. Each protocol defines the involved roles, the local variables of each role (i.e. its private variables), the content and order of exchanged messages and what specific security properties must be formally verified.

Those security properties are specified through the so-called claim events. A secrecy claim of a variable holds if each time an honest party communicates with honest parties, the variable value will not be known to the adversary. A non-injective synchronization claim—also known as Nisynch in Scyther jargon—states that everything intended to happen in the protocol definition also happens in its execution, i.e. all received messages were sent by the communication partner, and all sent messages have been received by the communication partner. Furthermore, Commit claims are used to require that parties agree on the values of a set of variables.

4.2 Verification of security properties

In order to validate the proposed solution, this paper aims to prove that ASSURE is secure against all of the attacks mentioned in Section 3.2 while at the same time providing resistance to cloning. Moreover, in this section, the security properties that are intended to show ASSURE provides are defined as follows:

- **Mutual authentication:** In every instance of the authentication protocol, if both participating parties are honest, then each party correctly verifies the identity of the other. Otherwise, if any participating party behaves maliciously, then the authentication process fails. The ASSURE protocol satisfies mutual authentication in the following manner. The server first verifies the client's authenticity by comparing the *ID* and *MAC* in the encrypted data α_d . Only the client can generate the correct encrypted data of the *MAC* and $nonce_{v_1}$ because of the PUF. The server only sends the encrypted α_v and M_{SA} to the trusted client. The client then verifies the server's authenticity by

checking if the content (MAC_A and $nonce_d$) of the α_v are accurate. Only the server which has the PUF response can generate the correct encrypted data α_v .

- **Resistance to man in the middle attack:** An attacker cannot break the mutual authentication property and peer-to-peer key agreement by running an MitM attack. As shown in Fig. 4, upon running the Scyther tool on ASSURE, which was itself reported in Section 4.3, the output shows that all the itemized claims were verified. This means that if the device and the verifier are honest, they mutually agree on what messages are exchanged, as well as their order and content (Nisynch and Commit claims), regardless of the potential actions of the attacker. As proven by Scyther that Commit, Nisynch and SKR (secret) claims are satisfied, this guarantees mutual authentication and resistance to the MitM attack in the ASSURE protocol. Consequently, all secret and sensitive information (PUF response, *MAC*, S_{key}) are transmitted securely because they are always encrypted before transmitted.
- **Resistance to eavesdropping attack:** In ASSURE protocol, an attacker cannot compromise the security of the transmitted sensitive data during the protocol execution by eavesdropping on the communication channel between client and server as proven by Scyther tool. This is because all sensitive data (PUF response, *MAC*, S_{key}) is encrypted during communication between the client and server.
- **Resistance to replay attack:** In the ASSURE protocol, an attacker cannot break the mutual authentication property and key agreement by running a replay attack. As the encrypted data α_d and α_v are both randomized by nonces ($nonce_d, nonce_{v_1}$) for each client and each authentication cycle, the resistance to the replay attacks defined in Section 3.2 is ensured. No communication or a part of the communication can be reused.
- **Resistance to de-synchronization:** The Nisynch (non-injective synchronization) part of the protocol definition

Claim	Status	Comments
proposedProtocol, A1 SKR Ri	Ok	No attacks within bounds.
proposedProtocol, A2 SKR sKey	Ok	No attacks within bounds.
proposedProtocol, A3 Nisynch	Ok	No attacks within bounds.
B proposedProtocol, B1 SKR Ri	Ok	No attacks within bounds.
proposedProtocol, B2 Nisynch	Ok	No attacks within bounds.
V proposedProtocol, V1 SKR Ri	Ok	No attacks within bounds.
proposedProtocol, V2 SKR tokeni	Ok	No attacks within bounds.
proposedProtocol, V3 SKR sKey	Ok	No attacks within bounds.
proposedProtocol, V4 Nisynch	Ok	No attacks within bounds.

Fig. 4 Scyther verification results

in Scyther verified that the authentication and key agreement protocol are not vulnerable to a de-synchronization attack. Considering that the response R_i is kept secret between the device and the verifier guarantees that the content of two exchanged messages α_d and α_v cannot be decrypted. The Nisynch claim of the protocol was verified by Scyther, i.e. that the device never loses synchronization with the server because its secrets and public data are freshly generated for each protocol cycle. The de-synchronization attack cannot be performed because listening or stopping any link in the protocol cannot break the synchronization of data between the communication parties.

- Resistance to server impersonation: An adversary cannot reveal the internal state of the device due to PUF and, thus, cannot emulate the proper server in the ASSURE protocol. Regarding server impersonation, it is not possible for a fake server to correctly generate the encrypted message α_v because it depends on the PUF response, the MAC and the random nonce $nonce_d$ value. Furthermore, the fake server cannot identify a device from its MAC with relation to its corresponding PUF. The MAC address is never exposed since it is always sent in an encrypted way on the channel.
- Resistance to device impersonation and cloning: The proposed ASSURE protocol has resistance to device impersonation and cloning attack due to the security properties of the PUF described in [9]. As stated in [9], the CRP behaviour of the PUF collapses severely if an invasive attack is executed against the device. This PUF property is evidence of tamper resistance [9]. In the ASSURE protocol, an encrypted output that covered the R_i (response) to C_i (challenge) does not give any information about the next R_{i+1} to C_{i+1} . It is impractical to predict the R_i to C_i without accessing the corresponding PUF. The device responds with an encrypted text to the server query using the newly generated nonce ($nonce_d$) and response (R_i). Thus, it is impossible for the attacker to generate the correct encrypted text using the challenge and nonce. Consequently, PUF protects sensitive secrets in the device, and the encryption prevents the collection of PUF responses from the communication channel. Hence, it is impossible to impersonate and clone the device.
- Resistance to model building attacks: In the classic PUF-based authentication, if an attacker has an ability to eavesdrop on challenge and response pairs sent in clear, he/she can imitate the PUF behaviour with the machine learning algorithms as described in [5]. As will be proved in Section 4.4, in the proposed protocol, an attacker cannot build a modelling attack because the output of the PUF is never exposed during the protocol process. The ASSURE protocol is resilient to model building attacks, which is achieved by hiding the responses sent in the

communication channel. The response is masked by using the RC5 algorithm, i.e. $\alpha_v = RC5_{R_i}(MAC_A, nonce_d)$. In this way, the real response is protected to prevent an adversary from collecting all CRPs and building a model of the PUF.

4.3 Defining the proposed protocol (ASSURE) in Scyther

The listing below shows the definition of the proposed protocol using the programming language of Scyther. Having the listing also allows the reader to check the consistency between this protocol and the protocol described in the previous section.

Listing 1 Model of the proposed protocol in Scyther jargon

```
//ASSURE protocol description
const PUF: Function;
protocol ASSUREProtocol(A,B,V)
{
  role A
  {
    fresh nonceD: Nonce; var nonceV: Nonce; var Ci: Nonce;
    var Ri: Nonce; var tokenI: var sKey; var nonceV2: Nonce;
    fresh nA: Nonce; fresh MACd: fresh rqt;
    recv_1(V, A, Ci, nonceV);
    match(Ri, PUF(k(A, V), Ci));
    send_2(A, V, A, (MACd, nonceD, nonceV(k(Ri)));
    recv_3(V, A, (MACd, nonceD)(k(Ri)), (MACd, sKey, tokenI,
    nonceV2)(k(Ri)));
    //key agreement
    send_7(A, B, tokenI, (A, B, nonceA, rqt)(k(sKey)));
    claim_A1(A, SKR, Ri);
    claim_A2(A, SKR, sKey);
    claim_A3(A, Nisynch);
  }
  role B
  {
    fresh nonceD: Nonce; var nonceV: Nonce; var Ci: Nonce;
    var Ri: Nonce; var tokenI: var sKey; var nonceV2: Nonce;
    fresh nonceB: Nonce; fresh MACb: var rqt;
    recv_4(V, B, Ci, nonceV);
    match(Ri, PUF(k(B, V), Ci));
    send_5(B, V, B, (MACb, nonceD, nonceV(k(Ri)));
    recv_6(V, B, (MACb, nonceD)(k(Ri)));
    //key agreement
    recv_7(A, B, tokenI, (A, B, nA, rqt)(k(sKey)));
    claim_B1(B, SKR, Ri);
    claim_B2(B, Nisynch);
  }
  role V
  {
    fresh nonceV: Nonce; fresh Ci: Nonce; var nonceD: Nonce;
    var Ri: Nonce; fresh tokenI; fresh timeP: Nonce;
    fresh sKey: Nonce; fresh nonceV2: Nonce; var MACd, MACb;
    //authentication of A
    send_1(V, A, Ci, nonceV);
    match(Ri, PUF(k(A, V), Ci));
    recv_2(A, V, A, (MACd, nonceD, nonceV(k(Ri)));
    match(tokenI, (A, MACb, timeP, sKey)(k(Ri)));
    send_3(V, A, (MACd, nonceD)(k(Ri)), (MACd, sKey, tokenI,
    nonceV2)(k(Ri)));
    //authentication of B
    send_4(V, B, Ci, nonceV);
    match(Ri, PUF(k(B, V), Ci));
    recv_5(B, V, B, (MACb, nonceD, nonceV(k(Ri)));
    send_6(V, B, (MACb, nonceD)(k(Ri)));
    claim_V1(V, SKR, Ri);
    claim_V2(V, SKR, tokenI);
    claim_V3(V, SKR, sKey);
    claim_V4(V, Nisynch);
  }
}
```

Three roles are defined: the devices, i.e. role A and B, and the verifier, i.e. role V. The local status of each part is defined as a set of variables: the nonces `nonceD`, `nonceV`, `nonceV2` and `nA`; the challenge `Ci`; and the response `Ri`. The device and the verifier generate one nonce each, `nonceD` and `nonceV`, respectively. The challenge `Ci` is generated by the verifier as it were a nonce. The response `Ri` derives from the PUF function declared at the beginning, known to both the device and the verifier. Each of these variables is either declared as `fresh`, meaning that it is randomly generated by the party, or as `var`, meaning that it gets a value which is assigned to it upon the reception of a message. Messages are sent using the `send` event and received through the `recv` event. The first parameter of the `send` sets the identity of the sender; the second specifies the identity of the destination. The same may be applied to the `recv` event. The other

parameters define the content of the message, i.e. what variables are included in the message. For the `send`, all variables must already have an assigned value. For the `recv`, the local variables included in the content are assigned the values provided in the corresponding `send` event. The `match` event assigns the value specified as the second parameter to the variable placed as the first parameter. In this case, the output of the PUF functions is computed on the secret key shared between the device and the verifier, i.e. $k(A, V)$, and on the challenge C_i . The `claim` events at the end of each role define the first six security properties for verification, as was already explained in Section 4.2.

All the above definitions aim to validate the security of ASSURE against protocol attacks. The following section will validate the security of the proposed protocol against the model building attack in particular.

4.4 Model building resistance

As previously discussed in [5], exposing the large number of challenge and corresponding response pairs of the PUF in the communication channel makes classical PUF-based authentication protocols vulnerable to model building attacks. PUF modelling can be implemented using machine learning methods such as the support vector machine and the neural network algorithm. The behaviour of most electronic delay-based PUFs, such as arbiter, can be modelled by taking advantage of challenge-response pairs [5]. Different PUF designs are proposed using an Arbiter PUF to generate non-linear CRPs, such as XOR Arbiter PUF [19] and feed-forward Arbiter PUF [37]. It is worth to mention, though, that these designs are still vulnerable to machine learning attacks [9, 38]. The possible way to prevent this attack is to break the relationship between challenge and response pairs by hiding the challenge or response using cryptographic primitives [16]. Che et al. [39] used a cryptographic hash function which is applied to the challenge. Its goal was to make it difficult for modelling attacks which attempt to deliberately apply a set of CRPs to be performed. For the purpose of obfuscating the responses, the XOR function is also included alternatively to make it more difficult for modelling attacks to be completed while at the same time not completely overcoming the risk [38].

Barbareschi et al. [16] aimed to hide the real responses involved in the authentication protocol. The authors presented the PUF model masking scheme using AES encryption. In their scheme, the R_i is generated stimulating the Anderson PUF with the C_i ; then, the C_i is also encrypted via AES encryption using the response (R_i) as a key. They claim that both the encrypted output $E_{R_i}(C_i) = R'_i$ and the C_i pairs (C_i, R'_i) can be used in an authentication protocol. The encryption algorithm guarantees that the operation cannot be deciphered

for the purpose of obtaining the response from the encrypted outputs [10]. This makes the authentication machine learning resilient. Using some strong cryptographic methods, for example, AES can completely overcome the risk of modelling attacks given the confusion and diffusion properties of encryption algorithms [10].

However, AES encryption is a costly approach for resource-constrained devices [25]. In this work, RC5 encryption is used as a more practical solution for hiding the PUF response because it is a more lightweight cipher for resource-limited devices.

As displayed in Fig. 5, the PUF response is hidden in the following manner. The response (R_i) generated by the PUF is used as a key. Then, the output (α_i) is generated by encrypting plaintext M , including the *mac* and the *nonces*, by the formula $\alpha_i = RC5_{R_i}(M)$ using RC5 encryption. The *nonces* were generated by a 40-bit pseudo-random number generator (PRNG). The resiliency of the classic PUF-based authentication process and the proposed authentication protocol against ML attacks (support vector machine and artificial neural network) are evaluated. For the evaluation, the resiliency of the Arbiter and TCO-PUF against ML attacks are tested.

4.4.1 Test vector generation and machine learning

ML attacks exploit the already used challenge and response pairs involved in the authentication protocol. In the basic PUF-based authentication protocol, the challenge and response pairs are sent in clear between the client and server, as was explained in [10]. Therefore, the model building attack is an open threat to this protocol. As reported in Section 3.2, an attacker can collect a sufficient number of CRPs and create a model with machine learning algorithms. For the machine learning analysis of the basic PUF-based authentication protocol, 32,000 test vectors, namely, challenge-response pairs (C_i, R_i) , are obtained from the 32-bit Arbiter PUF and TCO-PUF. For machine learning analysis of the proposed protocol, 32,000 test vectors (C_i, α_i) , namely, challenge-encrypted output pairs, are generated. Then, ANN and SVM are used to examine whether those test vectors can be modelled or not. In the next section, the test results of SVM and ANN over the test vectors mentioned above are evaluated.

Fig. 5 PUF response-hiding scheme

4.4.2 Model building attack results

In order to analyse the resiliency of the arbiter, TCO-PUF and the response-hiding scheme described in Fig. 5 against machine learning attacks, SVM and ANN were used. The Arbiter and TCO-PUF were introduced in [36, 40]. Both take the 32-bit challenge as an input and generate the 1-bit response as an output. These are modelled in the current study. Throughout the modelling analysis, the value b_i refers to bits of the challenge (C), i.e. $C = b_1 \cdots b_k$, and the output of the PUF is interpreted as a response R . Then the term o_i is used to refer to bits of the encrypted output (α_i), i.e. $\alpha = o_1 \cdots o_m$. The value of the k is often regarded as the bit length of each PUF. As shown in Fig. 5, the value of the encrypted output (128 bits) is generated by RC5 encryption according to the formula $\alpha_d = RC5_{R_i^A}(M)$ using the response as a key and the 80-bit nonces and 48-bit mac as the plain text. Thus, the value for m is the number of output bits generated by the response-hiding scheme.

For the purpose of comparison, the resiliency of the 32-bit Arbiter and TCO-PUFs against ML attacks can be derived from Fig. 6 and Fig. 7. The two figures show the prediction results of ML attacks against the Arbiter and TCO-PUF. It is obvious that the resulting prediction accuracies are very high, viz. 99.5% and 98.4%, respectively.

Following the above, the resiliency of the response-hiding scheme for the 32-bit Arbiter and TCO-PUFs against ML attacks is evaluated. Figure 6 and Fig. 7 also depict the prediction results of the response-hiding scheme using both types of ML attacks (SVM and NN). These results are based on the challenge (C_i) and the first 5 bits (o_1, o_2, o_3, o_4, o_5) of the encrypted output o_i . For each challenge (C_i) and o_i bit pairs, the prediction accuracies in average for the Arbiter response-hiding scheme are 52.6% in NN attack and 51.9% in SVM attack; conversely, on average, the prediction accuracies for the TCO response-hiding scheme are 51.6% in NN attack and 51.2% in SVM attack. As can be seen in the figures above, collecting more challenge (C_i) and encrypted output (o_i) pairs

Fig. 6 ML attacks (SVM, NN) on Arbiter PUF and its response-hiding scheme

Fig. 7 ML attacks (SVM, NN) on TCO-PUF and its response-hiding scheme

does not increase the accuracy of the machine learning processes. In order to predict the whole output of the response-hiding scheme, both ML techniques can be applied as in Arbiter and TCO-PUF analysis to predict its single output bits o_i . Based on the results gleaned from the figures above, the probability of predicting 1 bit of the encrypted output is $1/2$ in one attempt. As a result, the probability of predicting whole bits of the output is $1/2^m$ ($1/2^{128}$) in one attempt. From the context of a security system, an adversary who is aiming to decode the ciphered output encrypted with a 128-bit length key has to complete approximately 2^{128} computations in order to obtain the plaintext. The 128- and 256-bit security levels are the most common and carefully analysed cryptographic properties that have been tested in terms of sufficiency of security for the Internet of Things devices [41].

The above analysis concluded that the probability to predict the whole output of response-hiding schemes which were shown in Fig. 5 exponentially, and with certainty, reduced the misprediction rate of a single bit o_i . This appears to be reasonable for the ML attack analysis in what has been described. Hereby, the results show that response-hiding scheme ensures the bit security of the ASSURE protocol. These results have further strengthened the confidence of this work in terms of the resiliency of the response-hiding scheme in the ASSURE protocol against ML attacks. All of the above findings of the hiding scheme have also validated the usefulness of encryption algorithms' confusion and diffusion features which were described in [42].

4.5 Comparison

This section begins by describing a list of protocol requirements mentioned in the survey [10]. It then compares the proposed ASSURE protocol with the PUF-based protocols from the survey as well as lockdown protocols I and II from [18], considering the capabilities and claims of those protocols.

4.5.1 Protocol requirements based on survey [10]

In the survey [10], the protocol requirements are listed and described as follows:

- 1) Complete Specification: A protocol should be clearly stated and explained in a complete manner with its graphical representation.
- 2) Leakage Resilience: It should provide security against highly advanced physical attacks.
- 3) Able to Handle Noisiness: The reliability of the PUF circuit may not be stable because of the noises of their responses. To exploit a stable response, there are some approaches such as using error correction codes or error-free responses and giving fault tolerance [10].
- 4) Counteracting Modelling Attacks: The protocol should be resistant against machine learning attacks.
- 5) PUF Response Space Expansion: In order to prevent brute force and random guessing attacks, the PUF-based protocols require that the length of output generated by the response should be at least 128 bits large.
- 6) Low Cost in Resource Consumption: The PUF-based protocol should be lightweight in the resource-limited environments.
- 7) Easy to Instantiate: Impractical assumptions should not be made of the PUF in order to successfully and easily implement the protocol.
- 8) Resistance Against Protocol Attacks: The protocol should provide security against server impersonation, eavesdropping, MitM and replay attacks.
- 9) Mutual Authentication: The protocol should support mutual authentication between both the client and the server.

4.5.2 Comparison with other PUF-based protocols

In this section, Table 2 considers some of the PUF-based authentication protocols found in [13, 14, 17, 18, 43] and compares them with ASSURE. The table lists the protocol requirements from [10, 18]. This indicates the capabilities and claims of the protocols. All PUF-based protocols in this table utilize two stages. The first stage is the enrolment stage in a secured environment. At this stage, the verifier stores the PUF CRPs in a database or generates the shared secret with some information obtained from the PUF. The second stage is the verification stage. The prover and verifier then perform a verification over the communication channel. Most of the PUF-based protocols use cryptographic algorithms and error correction codes; moreover, they store CRPs on the verifier's side in order to maintain the authentication [10]. One of the most serious limitations of strong PUF authentications, which is the storage of secret CRPs, requires that the server interacts securely with each PUF device in enrolment stage and then used CRPs are deleted so that they are not used again. Accordingly, the server must collect and store several secrets or CRPs before deployment and update them over time. For large IoT networks, e.g. a WSN with thousands of devices, this requires not only a large amount of storage but also an intensive update workload. Unfortunately, this is not achievable due to the ubiquitous and resource-constrained nature of WSNs. In the proposed ASSURE protocol, though, neither the server nor the client needs to record and update the CRPs in the database.

Van et al. [43] proposed a PUF-based authentication protocol that provides mutual authentication using reverse fuzzy extractors. As mentioned in [18], the security in the first protocol of [43] relies on PRNG; however, it is cracked because

Table 2 Comparing the ASSURE protocol with other work finalists from [10, 18]

Protocols		Rostami et al. [13] (slender)	Sadeghi et al. [14] (Converse)	Bolotnyy et al. [17]	Lockdown I [18]	Lockdown II [18]	Van et al. [43] (reverse FE)	ASSURE
Protocol capabilities	Server auth.	○	○	○	○	●	●	●
	Client auth.	●	●	●	●	●	●	●
	Leakage resilience	-	●	○	○	○	-	●
	Error correction	○	●	■	○	○	■	○
	CRP storage	○	●	●	●	○	●	○
	Number of auth.	∞	∞	■	■	■	∞	∞
	Modelling robust	■	●	○	■	●	●	●
	Server impersonation	○	○	○	○	■	■	●
	Client impersonation	●	●	●	■	■	●	●
	MitM	■	●	○	○	○	●	●

● Satisfied, ■ partly satisfied, ○ no, – not applicable, ∞ limitless

of LFSR circularity. The slender protocols in [13] were also cracked because of LFSR linearity. A simplified solution to this problem is to redesign the PRNG carefully. As the authors of [44] have highlighted, an attacker can lead to an authentication error when the man in the middle substitution attack is carried out against the slender protocol.

Therefore, the PRNG security is highly important for this protocol because it exchanges the responses clearly and generates the challenge via the PRNG. It also clearly sends half of the challenge from the device (C_d) and half of the challenge from the server (C_s). An adversary can easily collect the exchange messages (i.e. two parts of the challenge) and the response. If the adversary manages to recognize the PRNG function and imitate it, then the whole challenge can be obtained. In the end, when there is a sufficient amount of the challenge and response pairs collected, the PUF model can be built very easily using machine learning techniques. On the contrary, the authors claim that they enhanced the security of the PRNG by restricting adversary on mounting the challenge manipulation [18]. Consequently, the lockdown II protocol in the table is assigned as a modelling robust.

The PRNG in the ASSURE protocol was based on a cryptographically secured generator of the Contiki OS [45]. Nevertheless, it does not need to be highly secured since it is not the cryptographic primitive directly used for security. Moreover, the response is always hidden. Subsequently, an adversary cannot collect responses from the communication channel and cannot build the model. This form of security comes from the PUF response and encryption algorithm. Since the response is sent as an encrypted message, the response R_i is kept secret between the device and the verifier, thereby guaranteeing that the content of the last two transmitted pieces of data cannot be decrypted as was proven by the Scyther security verification tool.

Sadeghi et al. [14] proposed an authentication method called the converse PUF-based authentication. In contrast to the traditional approach to PUF authentication, the prover contains a PUF's CRPs, and the verifier contains the PUF circuit. They provided a broad security analysis on that protocol. Regarding the authentication, however, they have to store CRPs in the prover database. This protocol only provides server authenticity. The main weakness of their study is that they make no attempt to support mutual authentication. The proposed ASSURE protocol, on the other hand, allows mutual authentication between the client and server, as evidenced in the security analysis section.

In the protocol of Bolotnyy et al. [17], there is an assumption that PUFs are secure against modelling attack. However, it is not realistic to accept that the PUF-based protocols are resilient to the machine learning attacks, except when using cryptographic primitives. Using an encryption method, for example, the RC5 used in this work can completely overcome the risk of modelling attacks due to the confusion and diffusion properties of encryption, as shown in Section 4.4.2.

Finally, there is still a considerable uncertainty about the energy consumptions of the aforementioned protocols because they do not have real case implementations and energy analyses on the IoT devices. The ASSURE protocol is the only exception. They mainly focus on the importance of physical security rather than resistance against protocol attacks. This paper sets out with the aim of assessing the importance of resource constraints and has offered as a solution the ASSURE security protocol. The next two sections describe the implementations and experimental analysis methodology which was adopted in this paper and evaluate the energy and memory consumption of the ASSURE protocol.

5 Experimental analysis method

This section outlines the experimental setup of this paper and explains the metrics used to evaluate the energy and memory-related costs of the proposed ASSURE protocol.

5.1 The purpose of the experiment

The purpose of these experiments is twofold. The first is to verify the functionality of the proposed solution; this is achieved by constructing a wireless network which has a server and two clients, as depicted in Fig. 2. The second goal of the experiments is to evaluate the energy consumption and memory utilization of the ASSURE scheme compared with the existing solutions, namely, the PSK-DTLS and UDP protocols.

5.2 Experimental setup

To conduct this experiment, the Zolertia Zoul re-mote, a computer and the Contiki operating system were utilized. The Contiki operating system running on VMware with a 10 GB hard disk and 2 GB of RAM was used for the experimental setup. In the case of hardware implementation, three Zolertia Zoul devices, equipped with 512 KB flash memory (ROM) and 32 KB RAM, were used, with the one serving as the server and the other two as the clients. Three Zoul devices were programmed with Contiki, and energy measurements were estimated by using the energest module [46] in Contiki. Table 3 presents the current consumption details of the CPU and radio in active and sleep modes derived from the

Table 3 Power breakdown of the Zolertia Zoul device

		Current	Min (sleep)
Computation	CPU	0.6 mA	1.3microA
	LPM	150 nA	–
Radio	Tx (transfer)	24 mA	–
	Rx (listen)	20 mA	–

Zoul node datasheet [47]. The operating voltage was 3.4 V. Those details are used to calculate the energy measurement of the specified protocols in Section 6.4.

The Zolertia Zoul device is a popular choice among researchers. It was chosen for this work in order to have a standard base for comparison. This device has found application in a host of real-world scenarios, such as powering smart homes [47].

The Contiki OS is a well-known open-source operating system developed by Adam Dunkels under the C programming language [45]. It was designed for low power IoT (Internet of Things) devices having limited memory, e.g. wireless sensor nodes. A typical Contiki configuration is suitable for a microcontroller with 2 KB of RAM and 40 KB of ROM memory [21]. With this operating system running on connected resource-constrained devices, it is possible to develop many different applications.

5.3 How to build a PUF model

This subsection describes how a PUF model required for the proposed protocol can be built. One assumption is that the verifier has access to the PUF in the device once the PUF model has been established. Then, it securely stores the derived model of every PUF and fuses the programmable wires so that the challenge-response interface cannot be accessed anymore. In order to create a PUF model, the dataset that is a collection of 32-bit Arbiter and TCO-PUF challenges and their corresponding responses are obtained from the Arbiter PUF that is implemented using the BSIM4 (V4.5) transistor model in 25C with 1.2 V as well as from the TCO-PUF which is simulated in a 130-nm CMOS technology; then the neural network (NN) algorithm is trained using CRPs. For the neural network, a multi-layer perceptron feed-forward structure utilizing the neural net package was used for modelling. Regarding the parameters of the neural network model, the number of hidden neurons was set to 32, the number of input neurons was set to 32, the steepness of sigmoid curve (λ) was 2, and the learning rate was set to 0.03, as optimal features. Using the neural network algorithm, both PUF responses are imitated with 100% training accuracy using 32,000 CRPs. This is an effective solution, which eliminates the need for storing CRPs in a database on the verifier side for authentication. Therefore, the Arbiter and TCO-PUFs were modelled for devices A and B, respectively, with a neural network. The PUF-FSM introduced in [78] generates reliable outputs called responses in a controlled manner. The PUF-FSM structure, which is a controlled strong PUF [78], includes a basic PUF, which could be an arbiter or TCO-PUF, an FSM (finite state machine). One assumption when building PUF-FSM is that PUF responses are obtained by the server (trusted entity) in a secured environment once during the preparation of the PUF model to generate the PUF statistical model, and then direct access to the PUF is destroyed, for example,

by fusing a wire. The models were then kept on the server's database, as described in the protocol description in Section 3.4.

5.4 Functional verification

A wireless network, which has a server and two clients, was constructed. First, the server successfully verified the respective identities of the two clients, and then the two clients agreed on a secret key using the proposed ASSURE protocol. In each case, this process was carried out ten times.

5.5 Metrics of evaluation

The experiment runs mainly in two phases: memory utilization and energy estimation.

Memory utilization To measure the memory utilization of each implemented protocols, the ARM toolchain command “arm-none-eabi-size” was used [45]. This command gives a breakdown of the data used up both the RAM and flash memory. After the program code has been compiled and uploaded to the device, the ARM toolchain is then run on the terminal to determine the memory utilization.

Energy estimation To measure the energy consumption of each component in the devices, an application called the energest module on the Contiki operating system was used. The energest module measures time by taking the readings of clock ticks, while the device is in the receive state, transmit state, processing (CPU) mode and low power mode. The processing time of each component in milliseconds (ms) is calculated by Formula (1):

$$\text{Processing time [ms]} = \frac{\text{Energest_Value(ticks)} \times 1000}{\text{CLOCK_SECOND}} \quad (1)$$

In order to measure the energy consumption of these states, the following formula was used:

$$E = \frac{\text{Energest_Value} \times \text{Current} \times \text{Voltage}}{\text{CLOCK_SECOND}} \quad (2)$$

where the *Energest_Value* is read off the terminal directly while the program is running, where the *voltage* and *current* at different operating levels are obtained from Table 3. This experiment is conducted in order to measure how efficient the ASSURE protocol is on constrained devices. The results from the proposed ASSURE protocol are compared with the results of a DTLS-based protocol in the following section. For the PSK-DTLS protocol, the measurements are limited to the handshake phase, since it is the part which is concerned with device authentication.

Based on all of the above-mentioned metrics, the following section will evaluate the memory and energy-related cost of the implementation of the ASSURE protocol.

6 Evaluation and cost analysis

In this section, the PSK-DTLS, the ASSURE, and the UDP without security have been evaluated by measuring energy consumption and memory utilization in resource-constrained devices (i.e. the Zolertia Zoul re-mote). It must be noted that the experimental results presented for the client correspond to the measurements on device A.

6.1 Estimation of memory usage

From the experiment conducted in Section 5.2, the memory utilization of the ASSURE protocol was measured. The total RAM and ROM usage are presented in Fig. 8.

To ensure that there is a fair comparison with the PSK-DTLS protocol, the memory used by both protocols was measured on the same devices. The memory-related cost of the UDP protocol which did not provide security, on the same devices, was also measured to see the cost of the security layer provided by the ASSURE protocol. Table 4 represents the total ROM and RAM usage of the three protocols, along with the usage details of the memory sections, namely, *bss*, *data* and *text*.

As described in [45], the RAM usage in the system is the sum of the *bss* and *data* sections. The *bss* area includes dynamic variables, whereas the *data* area composes static variables. The ROM usage in the system is the sum of the *text* and *data* sections. The ROM includes other sections of the program, which are static during the run time of the program.

The results obtained from the ASSURE implementation reveals that on the client, 13,631 bytes and 56,002 bytes of RAM and ROM were used, respectively, which accounts for 42.3% of the total available RAM and

Fig. 8 Memory footprints for the three protocols

Table 4 Memory utilization of their protocols on Zoul

	In byte	.text	.data	.bss	ROM	RAM
ASSURE	Client (Zoul)	55,170	832	12,799	56,002	13,631
	Server (Zoul)	59,505	1664	12,993	61,169	14,657
PSK-DTLS	Client (Zoul)	64,904	750	13,623	65,654	14,373
	Server (Zoul)	67,364	1500	13,733	68,864	15,233
UDP	Client (Zoul)	50,022	479	12,764	50,501	13,243
	Server (Zoul)	51,234	958	12,878	52,192	13,836

11.01% of the total available ROM. When comparing the aforementioned protocol with the PSK-DTLS implementation on the Zoul remote, it can be seen that the ASSURE protocol performs slightly better than the PSK-DTLS implementation, which itself used 65,654 bytes of ROM and 14,373 bytes of RAM memory. When the program is implemented without security on the client, the memory usage stood at 13,243 bytes and 50,501 bytes, respectively, which accounts for 40.41% of the available RAM and 9.63% of the available ROM.

6.2 Discussion of memory utilization results

The above results for memory utilization indicate that the overhead resulting from the use of the proposed protocol would not result in a massive strain on the resource-constrained device.

As shown in Table 5, the difference in RAM utilization when using security as compared with using the protocol without security stands at about 2%. When ROM utilization is examined, the difference was about 2%, which is not a large overhead to incur for security. The ASSURE protocol uses 9652 bytes less flash memory (ROM) than the PSK-DTLS handshake protocol in the resource-constrained device (Zoul).

6.3 Estimation of completion time

Another evaluation metric for comparison is the completion time of the PSK-DTLS handshake and the proposed protocol process.

Table 5 Percentages of memory utilization of three implementations

		PSK-DTLS	ASSURE	UDP
	Available	Usage (%)		
ROM	512 KB	12.52	11.01	9.63
RAM	32 KB	43.86	42.3	40.41

The processing time of each component is estimated in milliseconds by Formula (1) for the implementation of both the DTLS handshake implementation and the ASSURE protocol. Then the total completion time is estimated by summing the processing time of each component shown in Fig. 9. A total of ten measurements for both protocols were run. Figure 9 shows the average completion time over these measurements. Measurement results confirm that the PSK-DTLS handshake completion time is between 1.2 and 1.4 s on average and the ASSURE protocol's completion time is less than 1 s on average. Another purpose of time measurements is to estimate the energy consumption of both protocols. As a result, the total energy consumption is estimated according to Formula (3), which is detailed in the following section.

6.4 Estimation of energy consumption

A linear method is used when measuring online energy consumption for the devices is involved in the experimental setup mentioned in Section 5.2. The total energy consumption of all components is expressed as E . The formula for E is

$$E = (I_m t_m + I_l t_l + I_t t_t + I_r t_r) \times V \quad (3)$$

This formula has the following variables: V stands for supply voltage; I_m for the current draw of the microprocessor during operation; t_m for the operation time of the microprocessor; I_l and t_l for the current draw and time in low power mode; I_t and t_t for current draw and communication time in transfer mode, i.e. Tx; and I_r and t_r for the current draw and communication time in receive mode, i.e. Rx.

In order to evaluate the energy consumptions of the DTLS and the proposed ASSURE, each protocol was carried out ten times on the devices. Energy measurements were taken from each protocol cycle based on the power breakdown of the Zolertia Zoul node detailed in Table 3. Then, the average of the energy consumption results was obtained from these measurements. The energy estimation results show that when the

Fig. 9 Average completion time of DTLS handshake and ASSURE for one authentication cycle

Fig. 10 The average energy consumption estimates of three protocols on Zolertia Zoul device

experiment was conducted using the ASSURE protocol, the average energy consumed was 14.3 mJ (millijoule). Figure 10, which provides a graphical illustration of the results, shows that without security, the average energy stood at 5.6 mJ. When measuring the energy consumption of the PSK-DTLS handshake implementation, which is the equivalent of the authentication protocol, the average energy consumption was 23.6 mJ. This outcome can be gleaned from Fig. 10.

From the energy consumption figure, it can be observed that the CPU mode and LPM mode of the device consumed very little energy, while the receive and transfer modes consumed the most energy. This is because the device is almost always on the listen and transfer modes while expecting connections. Optimization of the receive and transfer modes can also assist in saving energy. The LPM mode consumes negligible energy because such low power IoT devices considerably reduce the amount of energy consumption by using the power saving modes specified in [47].

As analysed in [48], the gate count and energy per CRP of the Arbiter PUF circuit are shown in Table 6. This table also shows the energy per CRP of the modelled PUF on the Zoul device. As can be seen from the table, the difference in energy consumption between Arbiter PUF and its model is a negligible quantity. In total energy consumption (14.320 mJ) per authentication cycle, the exact arbiter PUF circuit consumes

Table 6 Gate equivalent and energy consumption [48]

PUF	GE	Energy per CRP in pJ
Arbiter PUF	450	0.239
Modelled PUF	—	0.235

Table 7 Transaction time/energy consumption of ASSURE and PSK-DTLS (128-bit key)

		Current	ASSURE	DTLS handshake
CPU		0.6 mA	222 ms, 0.453 mJ	263.4 ms, 0.537 mJ
LPM		0.4 μ A	496.9 ms, 0.001 mJ	650.5 ms, 0.001 mJ
Radio	Transfer (Tx)	24 mA	96.1 ms, 7.8 mJ	170.9 ms, 13.9 mJ
	Receive (Rx)	20 mA	88.6 ms, 6.02mJ	135.3 ms, 9.2 mJ
Total			903.6 ms, 14.3 mJ	1220.1 ms, 23.684 mJ

$0.239 * 128 = 30.592$ pJ, while the modelled PUF consumes $0.235 * 128 = 30.08$ pJ. The additional cost of a PUF will reduce the energy gains over the DTLS protocol such as 0.512 pJ = 512×10^{-12} mJ which can be negligible. Consequently, the share of energy consumption of PUF in total energy consumption is a negligible quantity, since most of the energy is consumed when the device is in communication as can be seen in Fig. 10.

6.5 Discussion of energy consumption results

Based on the above results for energy consumption obtained from the experiment described in Section 5.2, it can be seen that the difference between the secure (ASSURE) and non-secure (UDP) implementation in terms of energy is 8.6 mJ and that the amount of energy used is considerably different when compared with that of the PSK-DTLS handshake. These results promise that the proposed ASSURE protocol can be an effective security mechanism for low powered devices (Table 7).

One of the main reasons for the considerable difference in energy consumption can be attributed to the fact that the proposed protocol consists of four flights, while the authentication process in the DTLS is made up of a six-way handshake. This six-way handshake has been identified in [23] as one of the major consumers of resources in that protocol's implementation. All the above evaluation results indicate that the PSK-based DTLS is expensive for energy-constrained devices, whereas the ASSURE protocol requires a less amount of memory than the PSK-DTLS and that it consumes much less energy as well.

7 Conclusion

This work has been carried out to design a lightweight authentication and peer-to-peer key agreement protocol that would consume both less memory and energy. One of the main points of this paper was to compare the ASSURE protocol with the pre-shared key-based DTLS (datagram transport layer security) handshake and UDP (user datagram protocol), with respect to memory utilization and energy consumption in the resource-constrained environment. In order to develop a

secure and cost-effective solution, both physically unclonable function and light cipher that constitutes the main ideas behind the ASSURE method were used.

In the experimental setup for checking the functionality of the ASSURE protocol and estimating its memory utilization and energy consumption, a client and server application has been created and loaded into Zolertia Zoul re-mote devices. Through the client and server applications, the devices can interact with each other based on three protocols. Regarding their evaluation, the ARM GCC toolchain (arm-none-eabi) application has been used to determine memory usage, and the energest application has been used to monitor energy consumption. According to this evaluation, the ASSURE protocol has used less memory and consumed less energy for resource-constrained devices than the DTLS handshake implementation. This paper also describes the security analysis of the proposed protocol against a range of attacks and makes a comparison with other PUF-based protocols. The analysis showed that ASSURE provides mutual authentication and is resistant to MitM, eavesdropping, replay, server and client impersonation, desynchronisation, and model building attacks.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Raza S, Shafagh H, Hewage K, Hummen R, Voigt T (Oct. 2013) Lite: lightweight secure CoAP for the Internet of Things. *IEEE Sensors J* 13(10):3711–3720
2. Arfaoui G, Bultel X, Fouque P, Nedelcu A, Onete C (2019) “The privacy of the TLS 1.3 protocol.” in *Cryptology ePrint Archive*, vol. 2019, no. Report 2019/749, pp. 190–210

3. Caposelle A, Cervo V, De Cicco G, Petrioli C (2015) “Security as a CoAP resource: an optimized DTLS implementation for the IoT,” in IEEE International Conference on Communications, vol. 2015-Septe, pp. 549–554
4. Lessa dos Santos G, Guimaraes VT, da Cunha Rodrigues L, Granville LZ, Tarouco LMR (2015) “A DTLS-based security architecture for the Internet of Things,” in 2015 IEEE Symposium on Computers and Communication (ISCC), vol. 2016-Febru, pp. 809–815
5. Mukhopadhyay D (2016) PUFs as promising tools for security in Internet of Things. *IEEE Des Test* 33(3):103–115
6. Wallrabenstein JR (2016) “Practical and secure IoT device authentication using physical unclonable functions,” in 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 99–106
7. Maiti A, Kim I, Schaumont P (2012) A robust physical unclonable function with enhanced challenge-response set. *IEEE Trans. Inf. Forensics Secur.* 7(1):333–345
8. Gope P, Lee J, Quek TQS (2018) Lightweight and practical anonymous authentication protocol for RFID systems using physically unclonable functions. *IEEE Trans. Inf. Forensics Secur.* 13(11):2831–2843
9. Halak B (2018) *Physically unclonable functions from basic design principles to advanced hardware security applications*, 1st edn. Springer International Publishing, Cham
10. Delvaux J, Peeters R, Gu D, Verbaudhede I (Oct. 2015) A survey on lightweight entity authentication with strong PUFs. *ACM Comput Surv* 48(2):1–42
11. Resende ACD, Mochetti K, Aranha DF (2016) Lightweight cryptography for security and privacy, vol. 9542. Springer International Publishing, Cham
12. Frikken KB, Blanton M, Atallah MJ (2009) Robust authentication using physically unclonable functions. *Lecture Notes in Computer Science* 5735 LNCS:262–277
13. Majzoobi M, Rostami M, Koushanfar F, Wallach DS, Devadas S (2012) “Slender PUF protocol: a lightweight, robust, and secure authentication by substring matching,” in 2012 IEEE Symposium on Security and Privacy Workshops, pp. 33–44
14. Kocabaş Ü, Peter A, Katzenbeisser S, Sadeghi A-R (2012) Converse PUF-based authentication. *Lecture Notes in Computer Science* 7344 LNCS:142–158
15. Yilmaz Y, Do VH, Halak B (2020) ARMOR: an anti-counterefeit security mechanism for IOW cost radio frequency identification systems. *IEEE Trans Emerg Top Comput* 6750(c):1–15
16. Barbareschi M, Bagnasco P, Mazzeo A (2015) “Authenticating IoT devices with physically unclonable functions models,” in 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), pp. 563–567
17. Bolotnyy L, Robins G (2007) “Physically unclonable function-based security and privacy in RFID systems,” in Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom’07), pp. 211–220
18. Yu MD, Hiller M, Delvaux J, Sowell R, Devadas S, Verbaudhede I (2016) A lockdown technique to prevent machine learning on PUFs for lightweight authentication. *IEEE Trans Multi-Scale Comput Syst* 2(3):146–159
19. Suh GE, Devadas S (2007) “Physical unclonable functions for device authentication and secret key generation,” in 2007 44th ACM/IEEE design automation conference, vol. 129, pp. 9–14
20. Yang K, Blaauw D, Sylvester D (2017) Hardware designs for security in ultra-low-power IoT systems: an overview and survey. *IEEE Micro* 37(6):72–89
21. Sethi M, Arkko J, Keranen A (2012) “End-to-end security for sleepy smart object networks,” in Proceedings - Conference on Local Computer Networks, LCN., pp. 964–972
22. Sehgal A, Perelman V, Kürtyla S, Schönwälder J (Dec. 2012) Management of resource constrained devices in the internet of things. *IEEE Commun Mag* 50(12):144–149
23. Kothmayr T, Schmitt C, Hu W, Brünig M, Carle G (2013) DTLS based security and two-way authentication for the Internet of Things. *Ad Hoc Netw* 11(8):2710–2723
24. Yilmaz Y, Halak B (2019) A two-flights mutual authentication for energy-constrained iot devices. 2019 IEEE 4th Int. Verif Secur Work IVSW 2019:31–36
25. Perrig A, Szewczyk R, Wen V, Culler D, Tygar JD (2001) “SPINS: security protocols for sensor networks,” in Proceedings of the 7th annual international conference on Mobile computing and networking - MobiCom ‘01, pp. 189–199
26. Karlof C, Sastry N, Wagner D (2004) “TinySec: a link layer security architecture for wireless sensor networks,” in Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys ‘04, p. 162
27. Chatterjee U, Chakraborty RS, Mukhopadhyay D (2017) A PUF-based secure communication protocol for IoT. *ACM Trans Embed Comput Syst* 16(3):1–25
28. Chatterjee U et al (2018) Building PUF based authentication and key exchange protocol for IoT without explicit CRPs in verifier database. *IEEE Trans Dependable Secur Comput* 5971(c):1–14
29. Gao Y, Ma H, Al-Sarawi SF, Abbott D, Ranasinghe DC (2017) PUF-FSM: a controlled strong PUF. *IEEE Trans Comput Des Integr Circuits Syst* 37(5):1–1
30. Raza S, Trabalza D, Voigt T (2012) “6LoWPAN Compressed DTLS for CoAP,” in 2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems, pp. 287–289
31. Granjal J, Monteiro E, Silva JS (2012) “On the effectiveness of end-to-end security for internet-integrated sensing applications,” in 2012 IEEE International Conference on Green Computing and Communications, pp. 87–93
32. G. J., M. E., and S. J.S., “End-to-end transport-layer security for Internet-integrated sensing applications with mutual and delegated ECC public-key authentication,” in 2013 IFIP networking conference, IFIP Networking 2013, 2013, pp. 1–9
33. Shelby Z, Hartke K, Bormann C (2014) “The constrained application protocol (CoAP),” *RFC7252*, p. 3
34. “Tinydtls.” [Online]. Available: <https://sourceforge.net/projects/tinydtls/>. Accessed 1 Aug 2019
35. Maes R (2013) Physically unclonable functions constructions, properties and applications. Berlin, Springer Berlin Heidelberg
36. Mispan MS, Halak B, Chen Z, Zwolinski M (2015) “TCO-PUF: a subthreshold physical unclonable function,” in 2015 11th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), pp. 105–108
37. Daihyun L, Lee JW, Gassend B, Suh GE, van Dijk M, Devadas S (2005) Extracting secret keys from integrated circuits. *IEEE Trans Very Large Scale Integr Syst* 13(10):1200–1205
38. Ruhmair U, Solter J, Sehnke F, Xu X, Mahmoud A, Stoyanova V, Dror G, Schmidhuber J, Burleson W, Devadas S (2013) PUF modeling attacks on simulated and silicon data. *IEEE Trans Inf Forensics Secur* 8(11):1876–1891
39. Che W (2015) “PUF-based authentication invited paper,” in IEEE/ACM international conference on Computer-aided design, pp. 337–344
40. Mispan MS, Halak B, Zwolinski M (2017) “Lightweight obfuscation techniques for modeling attacks resistant PUFs,” in 2017 IEEE 2nd International Verification and Security Workshop (IVSW), pp. 19–24
41. Barker E (2016) “Recommendation for key management part 1: general,” NIST Spec Publ, pp. 51–54
42. El-din H, Ahmed H, Kalash HM, Allah O (2007) Implementation of RC5 block cipher algorithm for image cryptosystems. *Int J Inf Technol* 3(8):245–250

43. Van Herrewege A et al. (2012) “Reverse fuzzy extractors: enabling lightweight mutual authentication for PUF-enabled RFIDs,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7397 LNCS, pp. 374–389
44. Idriss T, Idriss H, Bayoumi M, Ieee F (2019) “Advanced PUF protocol : a machine learning resilient PUF-based protocol for lightweight mutual authentication,” <https://doi.org/10.13140/RG.2.2.35744.43529>, no
45. Velinov A, Mileva A (2016) “Running and testing applications for Contiki OS using Cooja simulator,” in *International Conference on Information Technology and Development of Education*, pp. 279–285
46. Dunkels A, Österlind F, Tsiftes N, He Z (2007) “Software-based sensor node energy estimation,” in *Proceedings of the 5th international conference on Embedded networked sensor systems - SenSys '07, 2007*, p. 409
47. “Zolertia RE-Mote Revision B,” 2016. [Online]. Available: <http://zolertia.io/product/hardware/re-mote>. Accessed 1 Aug 2019
48. Srivathsa SK (2012) “Secure and energy efficient physical unclonable functions,” Diss. University of Massachusetts Amherst

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.