SKAFS: Symmetric Key Authentication Protocol with Forward Secrecy for Edge Computing

Article *in* IEEE Internet of Things Journal · January 2023

DOI: 10.1109/JIOT.2023.3285513

3 authors, including:

Mohamed Mahdy Seifelnasr
Concordia University Montreal
7 PUBLICATIONS **71** CITATIONS

SEE PROFILE

Riham Altawy
University of Victoria
40 PUBLICATIONS **848** CITATIONS

SEE PROFILE

SKAFS: Symmetric Key Authentication Protocol with Forward Secrecy for Edge Computing

Mohamed Seifelnasr, *Student Member, IEEE*, Riham AlTawy, *Senior Member, IEEE*, and Amr Youssef, *Senior Member, IEEE*

***Abstract*—The IoT-edge-cloud paradigm enables resource-constrained IoT devices to offload their computation, thereby meeting the required quality-of-service for real-time applications. However, the deployment of IoT devices in public places such as smart cities exposes them to various security threats, including physical attacks. To address these security concerns, we propose a Physical Unclonable Function (PUF) based IoT-edge-cloud Symmetric Key Authentication protocol with Forward Secrecy (SKAFS), which ensures the anonymity of transacting IoT devices, resilience to desynchronization-based denial-of-service attacks, and PUF modeling attacks. To evaluate the security of our protocol, we conduct a formal security analysis using the automated AVISPA tool. In addition, based on the indistinguishability property of the PUF, we formally prove that SKAFS is secure under the CK-adversary model. Moreover, we implement the protocol using socket programming between a Raspberry Pi 1 as an IoT device, a Raspberry Pi 4 as an IoT gateway, and an 11th Gen Intel Core i7-11800H laptop as the CA to simulate the message flow between the protocol entities in a real-time experiment and calculate its end-to-end latency. Finally, we compare SKAFS with other PUF-based protocols in terms of computation time, communication cost, and storage requirements.**

***Index Terms*—PUF, Mutual authentication, Privacy-preserving, Forward secrecy, IoT, Edge computing, Desynchronization attacks.**

I. Introduction

CLOUD computing has been proposed to provide computing resources, such as servers, networks, storage, applications, and services to resource constrained IoT devices [1]. However, the traditional IoT-Cloud paradigm has a single point of failure, which has led to the adoption of the IoT-edge-cloud architecture. In this architecture, IoT devices offload heavy computation and storage to edge servers in their proximity [2]. The IoT-edge-cloud architecture is not a simple extension of the IoT-Cloud one. It requires revisiting the layers in the implementation stack and inspecting which logical and/or physical changes may bring new security implications [3]. Therefore, authentication and key agreement protocols designed for the IoT-Cloud architecture may not be suitable for the IoT-edge-cloud architecture, as pointed out by [4]. For instance, in the IoT-edge-cloud framework, the IoT device registers with the Cloud Admin (CA). Later on, registered IoT devices need to send authentication requests to geographically dispersed edge nodes without having pre-established keys with them. Such an authentication scenario is different from that of the IoT-Cloud architecture.

Several authentication protocols have been proposed to enable mutual authentication between communicating entities in the IoT-edge-cloud paradigm, such as those presented in [5]–[8]. These protocols achieve various security goals, including mutual authentication between IoT devices and edge nodes, as well as anonymity of the communicating entities and forward secrecy. However, these protocols do not account for the physical vulnerability of the widespread IoT devices, which according to Gartner statistics for the IoT market segment in 2020 [9], are expected to reach 2.38 billion devices deployed in unprotected public sectors, such as transportation, retail and wholesale trade, natural resources, and utilities, making them an attractive target for physical attacks [10].

Physical Uncloneable Functions (PUFs) have been explored as a way to prevent physical attacks. They have been used in PUF-based protocols for various applications, including RFIDs [11], electric vehicles [12], and IoT sensor nodes [13]. Aman *et al.* [14] proposed a PUF-based protocol that is resilient to physical attacks on IoT devices, in the IoT-Cloud architecture. However, this protocol is susceptible to desynchronization-based Denial of Service (DoS) attacks. Such attacks involve an adversary dropping update messages between communicating parties during the key agreement phase. Gope *et al.* [15] addressed such an issue by adding a synchronization set of extra pseudo-identities. They also proposed a mutual authentication protocol [16] that uses a reconfigurable PUF to prevent modeling attacks, while mitigating desynchronization-based DoS attacks using extra pseudo-identities stored at the IoT device and the CA. To maintain unlinkability, the requesting IoT device generates a new pseudo-identity with each request. However, with the increasing number of registered IoT devices and the ongoing service request, this imposes a pseudo-identity decoding problem and a storage overhead on the CA which scales linearly with the number of IoT devices and the size of the synchronization set.

Motivation. The existing protocols face various challenges, such as scalability issues with the increasing number of IoT devices, frequent Computation Offloading (CO) service requests, and vulnerability to physical and desynchronization-

Mohamed Seifelnasr is with the Concordia Institute for Information Systems Engineering (CIISE), Concordia University, Montreal, QC, Canada (e-mail: m.seifelnasr@h-eng.helwan.edu.eg and m_seifel@encs.concordia.ca). He is on leave from the Department of Computer Engineering, Helwan University, Cairo, Egypt.

Amr Youssef is with the Concordia Institute for Information Systems Engineering (CIISE), Concordia University, Montreal, QC, Canada (e-mail: youssef@ciise.concordia.ca)

Riham AlTawy is with the Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, Canada (e-mail: raltawy@uvic.ca)

based DoS attacks. In order to realize an efficient scheme addressing these challenges, we need to design a mutual authentication protocol for computation offloading that is resistant to physical compromise attacks, and desynchronization-based DoS attacks without the need for storing extra redundant pseudo-identities at the CA. The protocol should have a low complexity identification algorithm to enhance the efficiency of the CA in decoding IoT pseudo-identities, especially, with the large number of registered IoT devices and frequent IoT requests. Additionally, the proposed protocol has to maintain unlinkability between requests without updating the CA database with each authentication request. Finally, we need to ensure the security of the protocol, which relies on a single long-term key, in case of an IoT device compromise, so that other uncorrupted IoT devices can maintain secure operation.

Contributions. This paper has the following contributions:

- We propose SKAFS, a **S**ymmetric **K**ey **A**uthentication protocol with **F**orward **S**ecrecy for the IoT-edge-cloud paradigm. The security of the proposed protocol is based on the security of the utilized PUF and hash function. The protocol ensures the anonymity of IoT devices, provides forward secrecy, and is also resistant to hardware compromise attacks, machine learning-based modeling attacks, and desynchronization-based DoS attacks.
- We utilize the automated SPAN+AVISPA tool (Security Protocol ANimator for Automated Validation of Internet Security Protocols and Applications) for validating the security of our proposed scheme. Moreover, we provide a formal security analysis of SKAFS under the CK-adversary model.
- We estimate the cryptographic overhead time of the utilized cryptographic primitives on a Raspberry Pi 1 Model B+ with 512 MB of RAM and 0.7 GHz. Additionally, we implement SKAFS using socket programming between a Raspberry Pi 1 acting as an IoT device, a Raspberry Pi 4, equipped with a 64-bit Quad-core ARM Cortex-A72 processor clocked at 1.5 GHz, acting as an IoT gateway, and an 11th Gen Intel Core i7-11800H laptop acting as the CA, to simulate the message flow between the protocol entities in a real-time experiment, and calculate the end-to-end latency of the proposed protocol. Our source code for the Python implementation of the conducted experiments is available in the GitHub repository at https://github.com/IoT-SKAFS/SKAFS.
- we perform a comparative analysis of SKAFS with other recent PUF-based protocols, with a focus on assessing their security properties, computation complexity, communication cost, and storage overhead.

The advantages of SKAFS are summarized as follows: (i) our protocol adopts one shared long-term key for decoding the pseudo-identities of the IoT device. However, the protocol still maintains secure operation even with an IoT compromise. This is achieved by keeping the long-term key known only to the CA and the IoT device keeps another private key. Such an IoT private key is then used in the obfuscation operations done by the IoT which are equivalent to obfuscation using the CA's long-term key, (ii) unlike other proposed protocols which, for n IoT devices, require $\mathcal{O}(n)$ search complexity at the cloud server-side for identifying the IoT device from its pseudo-identity, SKAFS requires $\mathcal{O}(1)$ search complexity for identifying the IoT device. This is accomplished by obfuscating the identity of any IoT device with the same long-term key known only to the CA, and (iii) our scheme calls for synchronized states at both IoT devices and cloud admin sides without storing extra redundant pseudo-identities. As such, we embed a synchronization scheme in SKAFS to ensure that the state-update process takes place at the beginning of each session, thus, providing robustness with respect to DoS attacks.

In order to evaluate our protocol, we compare it with other closely related protocols in terms of communication cost, computation time, and storage overhead. Our performance evaluation shows that SKAFS requires constant storage of 64 bytes on the IoT device, a communication cost of 192 bytes, and a computation time of 16.95 msec.

The paper is structured as follows. In Section II, we provide a brief review of related work. Section III presents background information on the protocol primitives, while Section IV introduces the system model. In Section V, we describe the different phases of our protocol. The AVISPA validation, the formal security analysis of SKAFS, and the soundness of the synchronization scheme are discussed in Section VI. The comparative analysis of SKAFS with other related schemes is presented in Section VII. Finally, we conclude our work in Section VIII.

II. Literature Review

The IoT-Cloud paradigm has seen several proposed mutual authentication protocols [17]–[19]. Kalra *et al.* proposed an Elliptic Curve Cryptography (ECC)-based mutual authentication protocol for authenticating IoT devices to the cloud server [17]. Later on, Kumari *et al.* [18] proposed a new mutual authentication protocol to address the issues in Kalra's scheme such as lack of device anonymity, failure in session key agreement, and vulnerability to offline password guessing and insider attacks. Nevertheless, Kumari's protocol does not account for desynchronization-based DoS attacks, which can be caused by an attacker who disrupts the synchronization between the IoT device and the cloud server. To tackle this issue, Tewari and Gupta [20] proposed an ECC-based protocol that is resilient to DoS attacks. The aforementioned protocols proposed for the IoT-Cloud paradigm are not applicable to the IoT-edge-cloud paradigm, as they do not consider the new middle edge layer. This layer, comprising geographically-distributed decentralized edge nodes, introduces new security threats that were not present in the IoT-Cloud paradigm. These threats include but are not limited to, location exposure, insecure management, traffic analysis, intrusion of a malicious edge node, and trust issues between IoT devices and edge nodes from different vendors [4], [21]. Therefore, new mutual authentication protocols are required that take into account the unique security challenges posed by the IoT-edge-cloud paradigm.

Several mutual authentication protocols have been proposed in the IoT-edge-cloud paradigm, taking into account the middle

edge layer [5], [8], [22]. Nakkar *et al.* proposed a lightweight broadcast authentication protocol that uses hash chains to derive the session key and achieve forward secrecy [8]. However, it is a one-way authentication protocol for authenticating the cloud server, the sender of alert messages in emergency systems. In [22], a lightweight mutual authentication protocol has been proposed that allows IoT devices to register and subscribe with one CA for CO services. Cloud admins share the roots of a Merkle tree and a tree of secrets to authenticate and charge other cloud admins' IoT devices. However, this protocol does not provide full anonymity of IoT devices against insider attackers who share a partial path in the tree of secrets. Furthermore, the protocols do not consider the vulnerability of IoT devices to physical attacks. To address this, Wang *et al.* proposed LAMANCO [5], a PKI-free mutual authentication protocol that allows IoT devices to subscribe to a cloud server and get CO service from deployed edge nodes. However, the protocol requires tamper-proof devices to be installed in both IoT devices and the edge node, limiting its practical deployment.

PUFs have been proposed as a means of achieving feasible physical attack-resilient protocols [11], [13]–[16], [23]–[25]. For instance, Akgun *et al.* [23] proposed a scalable authentication protocol with $\mathcal{O}(1)$ search complexity, but it did not account for the compromise of the long-term key. Li *et al.* [13] proposed a PUF-based protocol that achieves forward secrecy for session keys, but does not achieve unlinkability of the sessions. Aman *et al.* [14] proposed two PUF-based protocols but they do not ensure the anonymity of IoT devices and are vulnerable to desynchronization-based DoS attacks. To address these issues, Gope *et al.* [15] proposed a PUF-based privacy-preserving protocol, but modeling attacks on PUFs were not considered. Furthermore, the works in [26] and [27], offer PUF-based multifactor authentication schemes for IoT devices that exploit the PUF circuit installed on the device and physical layer security to profile and authenticate the IoT device. In these schemes, the communicating entities rely on a PUF-derived secret and channel randomness to authenticate each other and establish a secret session key. However, the adoption of physical layer security schemes still has some hurdles [28]. The communicating entities need to assess that they are in a rich scattering environment before establishing the physical layer key, as the security of such a key relies on the uniqueness of the channel. Simple scattering environments are considered a disadvantage in this domain. Gope *et al.* [11], [29] studied using PUF as a second factor of authentication for IoT and smart grid systems. However, their proposed protocols require extra pseudo-identities to be stored at both the IoT device and the CA to mitigate desynchronization-based DoS attacks. This approach may not be effective if an attacker repeatedly performs desynchronization attacks on the IoT device, as it may run out of synchronization identities, requiring re-registration with the Cloud Admin. To mitigate the modeling attacks on PUFs, Gope *et al.* [16], [25] proposed two modeling-attack resilient reconfigurable PUF-based schemes.

Approaches in [11], [16], [25], [29] impose storage overhead on the cloud server due to the need to store a set of extra pseudo-identities and their corresponding PUF responses for each IoT device to thwart desynchronization-based attacks. Additionally, they require a $\mathcal{O}(n)$ searching algorithm for identifying the IoT device and updating the IoT pseudo-identity with each authentication request to ensure unlinkability. In contrast, our proposed protocol, SKAFS, is a modeling-attack resilient PUF-based mutual authentication protocol that achieves $\mathcal{O}(1)$ search complexity for identifying the pseudo-identity of the transacting IoT device and does not require updating the cloud server database with each authentication request. Additionally, SKAFS preserves the privacy and anonymity of IoT devices and is robust against desynchronization-based DoS attacks, as shown in Table I, which provides a comparison of SKAFS with other existing PUF-based protocols.

III. Preliminary

We use the notation $x \xleftarrow{r} \{0,1\}^n$ to denote the process of uniformly sampling an n-bit value x at random. We also use the notation $Y = Z$ to represent the assignment of the right-hand value Z to the left-hand value Y, while $Y \stackrel{?}{=} Z$ indicates a check of whether the right-hand side Z is equal to the left-hand side Y. In addition, we refer to our security parameter as λ. For a function $\epsilon(\lambda) : \mathbb{Z} \rightarrow \mathbb{R}$, we say that it is a negligible function in λ if, for all $d > 0$, there exists $\lambda_d \in \mathbb{Z}$ such that $\epsilon(\lambda) \leq 1/\lambda^d$ for all $\lambda > \lambda_d$ [30].

Physical Uncloneable Function. PUF can be considered a one-way function that generates a response R for a given input challenge C. In this paper, we refer to PUF calls as $\mathcal{P}(\ )$. PUFs were first introduced by Gassend *et al.* [31] and can be physically embedded into devices. The output response $R = \mathcal{P}(C)$ is determined by the specific PUF circuit and the challenge input itself. Such challenge-response behavior is unique and determined by the manufacturing process, making it impossible to clone or replicate. Consequently, the challenge-response behavior of the PUF can be used as a biometric identifier for the device. Since most PUFs are noisy, a fuzzy extractor [32] is typically employed to compensate for the variations in the output response. Ideal PUFs are characterized by [33]:

1) *Robustness:* A particular PUF circuit produces the same response R with a very high probability for the same challenge C, even if the experiment is repeated many times.
2) *Uniqueness:* The responses of two different PUFs for the same challenge C are distinguishable with a very high probability.
3) *Unpredictability:* Given the responses of a particular PUF for a set of challenges, it is infeasible to predict the response of this PUF to a new challenge.
4) *Tamper-Evidence:* Any external unauthorized access to the PUF to know the response of the PUF circuit for a certain challenge will alter its physical behavior and accordingly its challenge-response behavior.
5) *Indistinguishability:* The PUF response is computationally indistinguishable from a random sequence of the same length.

TABLE I: Security/ Functionality properties compared to other IoT-edge-cloud schemes

Security/ Functionality Properties	[14]	[15]	[11]	[13]	[8]	[22]	[24]	[16]	[25]	SKAFS
Mutual Authentication	✓	✓	✓	✓	×	✓	✓	✓	✓	✓
Confidentiality	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Unlinkability	×	✓	✓	×	×	×	✓	✓	✓	✓
Forward Secrecy	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Resilience to DoS	×	✓	✓	✓	×	×	×	✓	✓	✓
Resilience to Physical Attack	✓	✓	✓	✓	✓	×	×	✓	✓	✓
Resilience to Privileged Insider Attack	✓	✓	✓	✓	×	×	✓	✓	✓	✓
Resilience to Modeling Attack	×	×	×	×	×	×	×	✓	✓	✓
Dynamic IoT addition	✓	✓	✓	×	×	×	×	✓	✓	✓
Free of CA updating overhead [1]	×	×	×	NA	NA	✓	✓	×	×	✓
IoT-edge-cloud Compatible	×	×	×	×	✓	✓	✓	×	✓	✓
Identification complexity [2]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	NA	NA	$\mathcal{O}(log(n))$	$\mathcal{O}(log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$

[1]: There is no need to update the CA with a new IoT pseudo-identity with each authentication request
[2]: The complexity of the protocol in decoding an incoming IoT pseudo-identity.
✓: The security/functionality feature is achieved.
×: The security/functionality feature is not achieved.
NA: The protocol does not maintain unlinkability through pseudo-identity. Therefore, the functionality feature of updating the CA with a new pseudo-identity and decoding the pseudo-identity is not applicable in that context.

Given a PUF $\mathcal{P} : \{0,1\}^{l_1} \rightarrow \{0,1\}^{l_2}$ that maps an l_1-bit input challenge to an l_2-bit output response based on its physical properties, the PUF response should be computationally indistinguishable from a random sequence of the same length. Such a security notion is formally defined as:

Definition 1. *A physical uncloneable function $\mathcal{P} : \{0,1\}^{l_1} \rightarrow \{0,1\}^{l_2}$ is said to be indistinguishably secure if for any PPT adversary $\mathcal{D}$, $adv_{PUF}^{IND}(\mathcal{D})$ is a negligible function in l_1.*

We can define the adversary advantage against PUF indistinguishability, denoted as $adv_{\text{PUF}}^{IND}(\mathcal{D})$, as the probability of an adversary $\mathcal{D}$ winning a game against a challenger $\mathcal{C}$ that is running the PUF $\mathcal{P}$. The game involves the following steps [34]:

1) During the training phase, $\mathcal{D}$ repeatedly queries the PUF circuit run by $\mathcal{C}$ with challenges C_i and obtains responses $R_i = \mathcal{P}(C_i)$, where $i = 1, 2, \cdots, n$ and n is the number of queries.
2) After the training phase, $\mathcal{D}$ queries the PUF with a challenge C that is different from the ones used in the training phase, $C \notin \{C_1, C_2, \ldots, C_n\}$.
3) The challenger $\mathcal{C}$ generates a random bit $b \xleftarrow{r} \{0,1\}$ and responds with R_b to $\mathcal{D}$, where $R_0 \xleftarrow{r} \{0,1\}^{l_2}$ and $R_1 = \mathcal{P}(C)$.
4) $\mathcal{D}$ outputs its guess $b' \in \{0,1\}$ for b.

The adversary $\mathcal{D}$ wins the game if $b' = b$. The adversary advantage against PUF indistinguishability is defined as $adv_{\text{PUF}}^{IND}(\mathcal{D}) = |\mathsf{Pr}(b' = b) - 1/2|$.

Modeling Attacks and Reconfigurable PUFs. Practical PUFs have been vulnerable to machine learning-based modeling attacks. In such attacks, an adversary, denoted by $\mathcal{D}$, collects a large set of Challenge-Response Pairs (CRPs) $(c_1, r_1), (c_2, r_2), \cdots, (c_m, r_m)$ from the PUF. Using this set, the adversary builds a mathematical model, denoted by $\hat{\mathcal{P}}$ to predict the PUF's response R_{m+1} for a new challenge C_{m+1} [35]. Only strong PUFs are susceptible to modeling attacks, as weak PUFs have a limited set of CRPs, which makes it difficult to train a reliable model using a small amount of data [36].

Reconfigurable PUFs have been proposed as a way to thwart modeling attacks. In particular, dynamic reconfigurable PUFs can change their challenge-response behavior by resetting their configuration and transitioning between states of different challenge-response behavior [37]. Since resetting the PUF configuration renders any previous model useless. An adversary attempting a modeling attack must collect a new set of CRPs to model the new PUF behavior. It is important to note that reconfiguring the PUF does not affect its security properties, such as tamper-evidence and robustness.

DRAM-based PUFs (DPUF) have a very large addressing space and high density for providing a large set of CRPs which makes them suitable for realizing reconfigurable PUFs. In DRAM modules, data is stored in memory cells which are composed of a capacitor and an access transistor. Based on the charge of the capacitor a stored value of 0 or 1 is stored in the memory cell. However, DRAM modules require periodic refreshes of the capacitors to maintain data integrity. In DRAM-based PUFs, the cell power-up and refresh time are used to generate entropy between the stored challenge and the output response bits of the PUF. A challenge bit string may be stored in an arbitrary memory block of the DRAM module then a refresh-time interval is applied. Then, the stored bit in the memory can be read as the response bit string. For resetting the PUF configuration to realize a new challenge-response behavior, the setting parameters such as the memory block in the DRAM module and the cell refresh time are used. Note that Sutar *et al.* [25] showed that a refresh-time interval of 40-60 ms generates an entropy of $200 - 500$ bit flips across a 32KB memory block. This new resulting bit-flip behavior is extremely difficult to predict which implies that an adversary with a prior-built model of the PUF has a low probability of predicting the CRPs of its new configuration. Hence, reconfigurable PUFs are secure against modeling attacks. Also, [37] showed that setting a match

threshold equal to 27 makes such a design achieve a 100% true-positive rate (i.e., robustness rate) while also ensuring 0% false-positive rate (uniqueness rate).

Fractional Hamming Distance (FHD). Practical PUFs are known to be noisy, meaning that the output response of a PUF will differ slightly when queried multiple times with the same challenge. To quantify this difference, the Hamming Distance (HD) metric is commonly used. HD measures the difference between two strings by counting the number of substitutions required to change one string into another. Two identical strings will have an HD of zero. FHD is a variant of HD that is normalized by the length of the original string. Specifically, FHD between the strings r and $\hat{r}$ is defined as $\text{FHD}(r, \hat{r}) = \frac{HW(r \oplus \hat{r})}{L(r)}$, where $HW(r \oplus \hat{r})$ is the number of non-zero digits (i.e., 1's) in the XoR of r and $\hat{r}$, and $L(r)$ is the length of the original string r.

IV. System Model and Threat Model

This section introduces the system model and key entities in our system, followed by the threat model. Additionally, we provide a list of abbreviations used throughout the paper in Table II.

A. System Model

We have adopted the IoT-edge-cloud paradigm, which is comprised of three layers: the IoT layer, the edge layer, and the cloud layer. The cloud layer is operated by the CA, as shown in Fig. 1. The edge layer is securely connected to the cloud layer through the core network. In order to meet the Quality of Service (QoS) requirements of real-time applications, low-end devices in the IoT layer outsource their heavy computations and storage to the edge server, such as wireless access points and base transceiver stations, in the edge layer via a wireless link. The computational-capable edge server in the proximity of the IoT device performs the computation for the resource-constrained IoT device.

We assume that every IoT device is outfitted with a PUF circuit. Similar to previous works such as [11], [38], [39], we consider the PUF circuit and the IoT microcontroller as a single system on a chip, where the connection between them is regarded as secure. Any tampering with the PUF would change its behavior and render it ineffective. To prevent code modification attacks, it is assumed that the IoT microcontroller is equipped with tamper-proof memory, which guarantees code integrity and protects against malicious code modification attacks through the PUF, as described in [40].

The entities in our proposed protocol are:

1) *Cloud Admin*: The cloud service provider is responsible for deploying the CA, which manages the registration of IoT devices and stores their identities and the corresponding responses from the PUF circuit. To ensure the security of the CA, we assume, as in [25], that the server has access to secure storage and is protected from external interference.
2) *IoT Gateway*: The cloud service provider deploys IoT gateways as access points for IoT devices to access CO services.

TABLE II: List of abbreviations.

Abbreviation	Description
CA	Cloud Admin
IoT_j	the j-th IoT device
N_{IoT}	number of subscribed IoT devices
G_l	the l-th IoT Gateway
N_G	number of deployed gateways
j	index of the IoT device, where $j = 1, 2, \ldots, N_{IoT}$
l	index of the IoT gateway, where $l = 1, 2, \ldots, N_G$
PUF	Physical Unclonable Function
$\mathcal{P}(\,)$	invoking a PUF call
$\mathcal{P}_F(\,)$	fixed PUF call
$\mathcal{P}_D(\,)$	dynamic PUF call
$\mathcal{P}_D^i(\,)$	configuration state i for the dynamic PUF
$\mathcal{K}_{CA-G}$	the session key shared between the IoT gateway and the CA
$\mathcal{K}_{CA-G}^{Next}$	the updated session key between the IoT gateway and the CA
MK_{CA-G}	the master key between the IoT gateway and the CA
ID_G	identity of the IoT gateway
$\texttt{ID}$	identity of the IoT device
M	authentication message
S_{IoT}	the IoT device internal state
S_G	the IoT gateway internal state
C_{F0}, C_{F1}	fixed challenges
r	random nonce
K	the fixed long-term key
$\oplus$	bitwise "exclusive-OR" operator
T_j	a bit string stored on IoT device j
K_s	session key for the communication between the IoT device and the IoT gateway
i	session index
M_{K_i}	symmetric encryption of M under session key K_i
D_{IG}	synchronization difference between IoT device and the CA
D_{CA-G}	synchronization difference between the IoT gateway and the CA
K_p^G	the previous session key stored at the IoT gateway
K_c^G	the current session key at the IoT gateway
K_n^{IoT}	the next session key at the IoT device
K_c^{IoT}	the current session key at the IoT device
K_a	the latest common authentication key between the IoT device and the CA

3) *IoT Device*: Low-end devices are limited in their resources and have internet accessibility. They are deployed in various applications such as healthcare and virtual reality. To meet the required QoS, IoT devices offload some of their computation and storage to the IoT gateway via a wireless link.

B. Threat Model

In this paper, we adopt the Canetti-Krawczyk (CK) adversary model [41], in which the stored information in the memory of any communicating entity is vulnerable to memory leakage attacks. An adversary $\mathcal{A}$ in the CK adversary model can eavesdrop on, drop, insert, or modify messages between the communicating entities, as in the conventional Dolev-Yao threat model [42]. In addition, the adversary has access to stored information such as ephemeral secrets, session keys, and long-term private keys through explicit attacks.

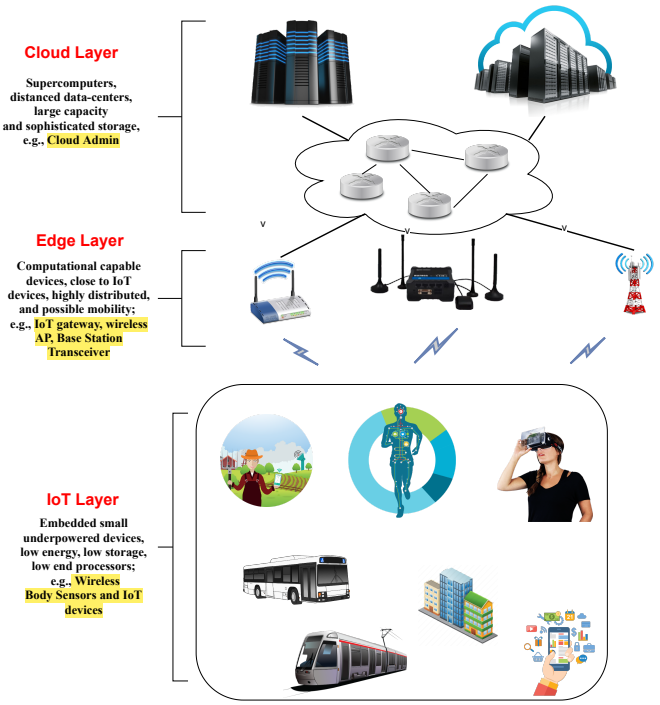

Fig. 1: The IoT-edge-cloud paradigm.

V. Protocol Design

In this section, we present the different phases of our proposed protocol. We assume that the registration phase where IoT devices and IoT gateways register with the CA runs in a secure environment. Moreover, throughout the protocol, we assume our keys to be 128 bits. In our protocol, we consider two PUFs: weak PUF $\mathcal{P}_F$ and re-configurable PUF $\mathcal{P}_D$. We consider the weak PUF (e.g., SRAM) and non-volatile memory to be embedded in system-on-Chip. The weak PUF is used for storing the long-term key on the IoT device. On the other hand, re-configurable PUF $\mathcal{P}_D$ is used in deriving the device's one-time key and it is attached to the device's external memory.

A. Registration Phase

The CA generates its long term key $K \xleftarrow{r} \{0,1\}^{128}$ and sets its identity to ID_{CA}. The registration phase between IoT_j and CA is illustrated in Fig. 2. Note that $j = 1, 2, \cdots, N_{IoT}$ denotes the IoT device index, where N_{IoT} is the total number of registered IoT devices. Initially, IoT_j randomly generates the challenge $C_1 \xleftarrow{r} \{0,1\}^{128}$. Subsequent challenges are computed by applying the hash function over the previous challenge using $C_{i+1} = H(C_i)$ where i is the session index and it starts from 1. On the other hand, C_{F0} and C_{F1} are fixed hardwired challenges for the PUF circuit. Then, the IoT device sends the PUF response for the re-configurable DRAM-based PUF $\mathcal{P}_D^1(C_1)$ and the fixed PUFs $\mathcal{P}_F(C_{F0}), \mathcal{P}_F(C_{F1})$ along with its identity ID to the CA. Afterwards, CA computes the bit string $T_j = \mathcal{P}_F(C_{F0}) \oplus \mathcal{P}_F(C_{F1}) \oplus K$ where K is the cloud long-term key, and C_{F0} and C_{F1} are fixed for all IoT devices. The resulting T_j is unique to each IoT device and is stored in its memory. It is different from one device to another based on their PUF responses to C_{F0} and C_{F1}. The CA sends T_j to the IoT device, stores the device's identity ID, and initializes the current session key between the IoT device and the IoT gateway, K_c by $\mathcal{P}_D^1(C_1)$. The previous session key, K_p, is initialized with $K_0 \xleftarrow{r} \{0,1\}^{128}$, and the two-step previous K_{p-1} is initialized with $K_{-1} \xleftarrow{r} \{0,1\}^{128}$. The registration of the IoT gateway l is illustrated in Fig. 3 where the IoT gateway sends its identity to the CA. Note that $l = 1, 2, \cdots, N_G$ denotes the IoT gateway index where N_G is the total number of registered IoT gateways. In turn, the CA generates the long-term key MK_{CA-G} and initializes the session key between the CA and the gateway $\mathcal{K}_{CA-G}$ with $H(\mathcal{K}_{CA-G}^p || r_1^p)$ where $\mathcal{K}_{CA-G}^p$ is the previous session key and it is initialized with $\mathcal{K}_0 \xleftarrow{r} \{0,1\}^{128}$ and the random r_1^p is initialized with $r_0 \xleftarrow{r} \{0,1\}^{128}$. Finally, the CA shares both the long-term key and the initial values with the IoT gateway.

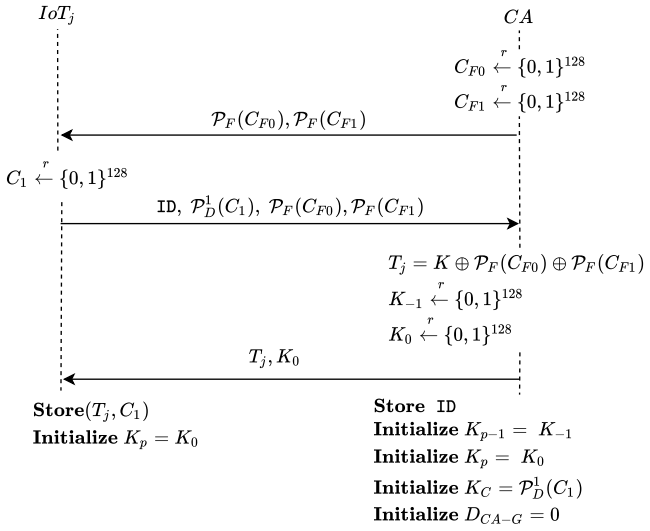

Fig. 2: IoT device registration.

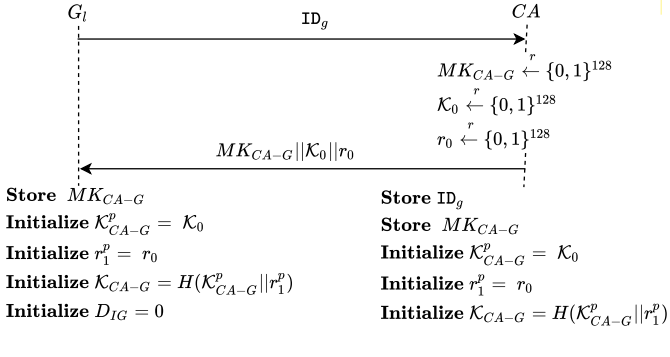

Fig. 3: IoT gateway registration.

B. Mutual Authentication Phase

Consider the i^{th} session between an IoT device IoT_j and an IoT gateway G_l. Initially, IoT_j sends Hello message to the

G_l. G_l then replies with a random nonce $r_1 \xleftarrow{r} \{0,1\}^{128}$. Afterwards, IoT$_j$ computes its obfuscated identity $\texttt{ID}^* = H(r_1, r_2) \oplus \texttt{ID}$, obfuscated key $K_i^* = K_p \oplus K_i$ and the authentication message $M_1 = H(K_i, r_1, r_3)$ where r_2 and r_3 are two randoms chosen by the IoT device $r_2,\ r_3 \xleftarrow{r} \{0,1\}^{128}$. Additionally, IoT$_j$ obfuscates r_2 with the long-term key as shown in Fig. 4 where r_2 is XoRed with the PUF output $\mathcal{P}_F(C_{F0})$ then with the other PUF output $\mathcal{P}_F(C_{F1})$ and finally with T_j. The deletion of both r_2 and the intermediate data (i.e., $\mathcal{P}_F(C_{F0})$, $\mathcal{P}_F(C_{F1})$) thwarts the disclosure of the long-term K in case of a physical attack on IoT$_j$. Following that, IoT$_j$ sends M_1, $\texttt{ID}^*$, r_2^*, r_3^* and K_i^* to the IoT gateway which in turns forwards it to the CA encrypted under the session key between the CA and the IoT gateway, $\mathcal{K}_{CA-G}$. Note that, this shared session key is updated with $\mathcal{K}_{CA-G}^{Next} = H(\mathcal{K}_{CA-G}||r_1)$ for each subsequent session to ensure forward secrecy. Moreover, the IoT gateway sends the messages $\sigma_1,\ \sigma_2,\ \mathcal{E}_1$ to the CA. Then, the authentication token σ_1 is verified at the CA and σ_2 is used to determine the synchronization difference D_{CA-G} between the IoT gateway and the CA. The CA retrieves the random r_2' and the identity of the IoT device $\texttt{ID}'$. Then, CA picks the stored keys of IoT$_j$ (i.e., two-step previous key K_{p-1}, previous key K_p, and current key K_c). Afterwards, the CA sends $\sigma_2,\ \mathcal{E}_2,\ D_{CA-G}$ to the IoT gateway which updates its $\mathcal{K}_{CA-G}$ in case that $D_{CA-G} = -1$ and verifies the authentication token σ_3. Then, to determine the synchronization state of IoT$_j$, the IoT gateway verifies the fractional Hamming distance of K_i and checks M_1 against the current key K_c and the previous key K_p. In case of a successful verification with one of the keys, the IoT gateway sends the authentication message M_2 to IoT$_j$ along with the synchronization difference flag D_{IG}.

C. Key Agreement Phase

In case of a lagging IoT device where the value of $D_{IG} = -1$, IoT$_j$ updates its challenge $C_i = H(C_i)$ and also sets the configuration of the $\mathcal{P}_D$ to the next state. Then, it computes the synchronized key K_i as illustrated in Fig. 5. For the subsequent session, IoT$_j$ sets $\mathcal{P}_D$ to $(i+1)^{th}$ session and computes K_{i+1}. Then, it returns back to the i^{th} session configuration. Afterward, it forwards the obfuscated keys $\hat{K}_{i+1}$ to the IoT gateway which in turn sends it to the CA. The CA updates the synchronization keys $K_{p-1},\ K_p,\ K_c,\ r_1^p,\ \mathcal{K}_{CA-G}^P$ and $\mathcal{K}_{CA-G}$. After receiving the confirmation message M_3, the IoT gateway updates the session key $\mathcal{K}_{CA-G}$ and sends the message M_4 to IoT$_j$ to confirm the key agreement and synchronization. Finally, IoT$_j$ verifies M_4 and updates its challenge, and reconfigures the $\mathcal{P}_D$ to its new state.

D. Message Exchange Phase

After mutual authentication and session key agreement, the IoT device starts offloading its heavy computation to the IoT gateway. The IoT device exchanges messages with the IoT gateway encrypted under the session key K_s as shown in Fig. 6.

E. Dynamic IoT device Addition

The proposed protocol supports the dynamic addition of an IoT device during an up-running instance of the protocol. This can be achieved by the IoT device sending its fixed PUF responses $\mathcal{P}_F(C_{F0})$ and $\mathcal{P}_F(C_{F0})$ and the dynamic PUF response $\mathcal{P}_D^1(C_1)$ to the CA admin. After that, CA responds with the $T = K \oplus \mathcal{P}_F(C_{F0}) \oplus \mathcal{P}_F(C_{F1})$ that the IoT will use later in obfuscating its pseudo-identity.

VI. Soundness and Formal Security Analysis

In this section, we prove the soundness of the synchronization scheme of SKAFS and provide a formal security analysis of our proposed protocol.

A. Soundness of SKAFS

According to [43], soundness of a synchronization scheme requires that after a complete correct session between an IoT device and an IoT gateway, the respective internal states of the IoT device, S_{IoT}, and the IoT gateway, S_G, are updated and synchronized. Moreover, the IoT device and the IoT gateway agree on a new session key.

Theorem 1. *Let S_{IoT} and S_G denote the IoT device and gateway states, respectively. At the i-th session, let K_c^{IoT} and K_n^{IoT} denote the current and next session keys at the IoT device, respectively. Similarly, let K_p^G and K_c^G denote the previous and current session keys at the IoT gateway. SKAFS ensures that the following two conditions always hold.*

1) *Either ($S_{IoT},\ S_G$) = ($S_i,\ S_i$) indicating synchronized states, or ($S_{IoT},\ S_G$)= ($S_{i-1},\ S_i$) indicating a lagging IoT device state, can exist.*
2) *After the completion of a correct session, both the IoT device and the gateway must have updated their internal states at least once, and $K_c^{IoT} = K_c^G$.*

Proof. Considering the first case where the IoT device and the IoT gateway are synchronized (i.e., $D_{IG} = 0$), the IoT gateway verifies the authentication message M_1 using K_c^G. Then, the gateway sets $D_{IG} = 0$ and sends the authentication message M_2 to the IoT device for computing the session key. Therefore, starting with $D_{IG} = 0$ and after receiving each of $M_1,\ M_2,\ K_{i+1}$ and M_4, one can obtain the following IoT and gateway states S_{IoT} and S_G for the session i

1) Initially,
($S_{IoT},\ S_G$) = ($S_i,\ S_i$), $D_{IG} = 0$.
2) After receiving M_1 by the IoT gateway,
($S_{IoT},\ S_G$) = ($S_i,\ S_i$), $D_{IG} = 0$.
3) After receiving M_2 by the IoT device,
($S_{IoT},\ S_G$) = ($S_i,\ S_i$), $D_{IG} = 0$.
4) After receiving K_{i+1} by the IoT gateway,
($S_{IoT},\ S_G$) = ($S_i,\ S_{i+1}$), $D_{IG} = -1$.
5) After receiving M_4 by the IoT device,
($S_{IoT},\ S_G$) = ($S_{i+1},\ S_{i+1}$), $D_{IG} = 0$.

The evolution of the respective internal states in the IoT device and the IoT gateway are listed in Table III. We can see that upon receiving M_2, the IoT device and the gateway are synchronized (i.e., $D_{IG} = 0$) and they agree on the same

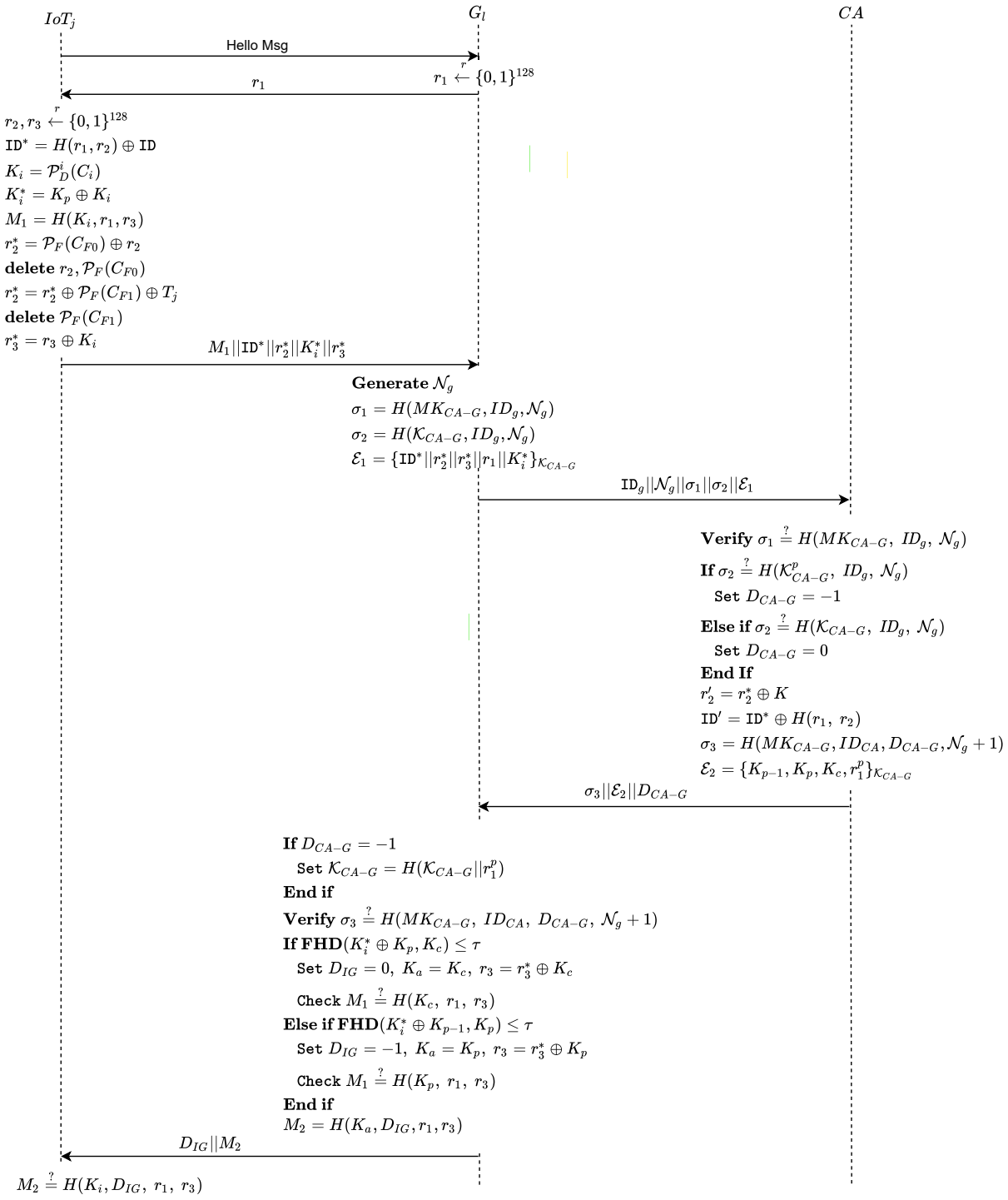

Fig. 4: Mutual authentication between the IoT device and the IoT gateway for the i^{th} session.

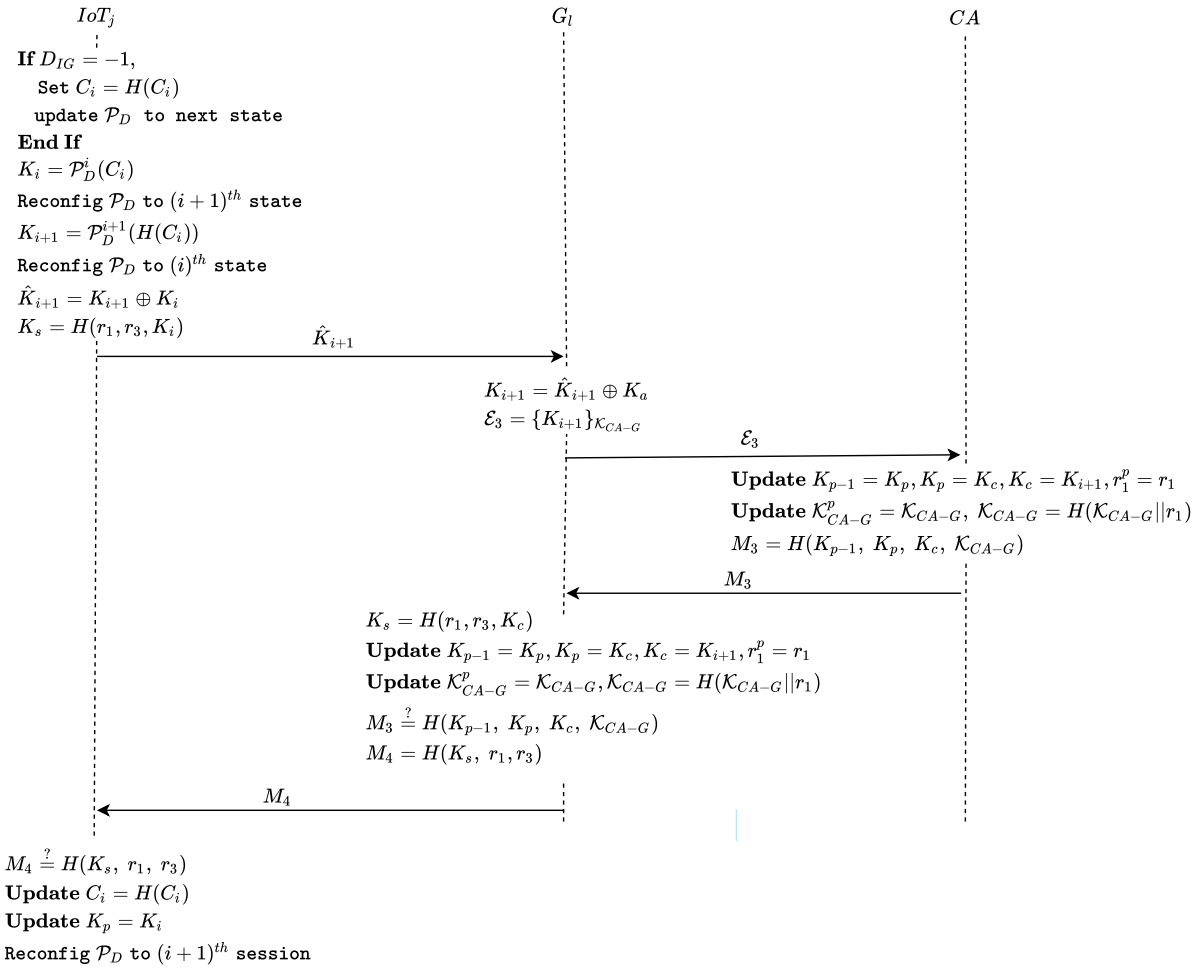

Fig. 5: Key agreement between the IoT device and the IoT gateway for the i^{th} session.

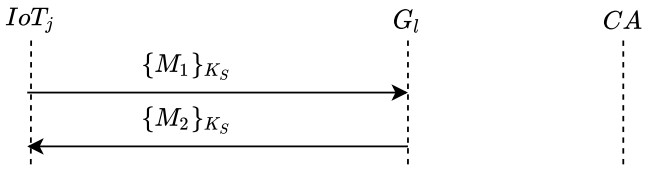

Fig. 6: Obtaining the CO service offered by the IoT gateway.

session key (i.e., $K_c^{IoT} = K_i$ and $K_c^G = K_i$). Moreover, after a complete run of SKAFS, both the IoT device and the gateway update their internal states at least once (i.e., $(S_{IoT},\ S_G) = (S_{i+1},\ S_{i+1})$). Note that the possible resulting D_{IG} values are either 0 or -1, and the case where $D_{IG} = 1$ does not occur since the IoT device updates its state only upon receiving the confirmation message M_4.

Investigating the other case where the IoT device is lagging, i.e., $D_{IG} = -1$, the IoT gateway has K_p^G, and K_c^G from the previous session, the IoT gateway verifies the authentication message M_1 using its K_p^G. Then, the IoT gateway sets $D_{IG} = -1$ and sends the authentication message M_2 to the IoT device for synchronization and session key computation. After M_2 verification, the IoT device updates its internal state and computes the session key. Therefore, starting with $D_{IG} = -1$, after the authentication messages M_1, M_2, K_{i+1} and M_4, one can obtain the following IoT and gateway states S_{IoT} and S_G for the session i

1) Initially,
   $(S_{IoT},\ S_G) = (S_{i-1},\ S_i)$, $D_{IG} = -1$.
2) After receiving M_1 by the IoT gateway,
   $(S_{IoT},\ S_G) = (S_{i-1},\ S_i)$, $D_{IG} = -1$.
3) After receiving M_2 by the IoT device,
   $(S_{IoT},\ S_G) = (S_i,\ S_i)$, $D_{IG} = 0$.
4) After receiving K_{i+1} by the IoT gateway,
   $(S_{IoT},\ S_G) = (S_i,\ S_{i+1})$, $D_{IG} = -1$.
5) After receiving M_4 by the IoT device,
   $(S_{IoT},\ S_G) = (S_{i+1},\ S_{i+1})$, $D_{IG} = 0$.

The evolution of the IoT device and the IoT gateway internal states are listed in Table IV. We can see that after the reception and verification of the authentication message M_2, the IoT device updates its internal state and becomes synchronized with the IoT gateway (i.e., $D_{IG} = 0$); meanwhile, the IoT device and the IoT gateway agree on the same session key (i.e., $K_c^{IoT} = K_i$ and $K_c^G = K_i$). Moreover, during a complete run of SKAFS, the IoT device updates its internal states twice while the IoT gateway updates its internal state once (i.e.,

initially, (S_{IoT}, S_G) = (S_{i-1}, S_i) and at the end (S_{IoT}, S_G) = (S_{i+1}, S_{i+1})). Note that the possible resulting D_{IG} values in the lagging case $\in \{0, -1\}$. □

TABLE III: The evolution of the internal states in case $D_{IG} = 0$

Epoch	state	D_{IG}	K_c^{IoT}	K_n^{IoT}	K_p^G	K_c^G
Initial	(S_i, S_i)	0	K_i	K_{i+1}	K_{i-1}	K_i
After M_1	(S_i, S_i)	0	K_i	K_{i+1}	K_{i-1}	K_i
After M_2	(S_i, S_i)	0	K_i	K_{i+1}	K_{i-1}	K_i
After $\hat{K}_{i+1}$	(S_i, S_{i+1})	-1	K_i	K_{i+1}	K_i	K_{i+1}
After M_4	(S_{i+1}, S_{i+1})	0	K_{i+1}	K_{i+2}	K_i	K_{i+1}

TABLE IV: The evolution of the internal states in case D_{IG} = -1

Epoch	state	D_{IG}	K_c^{IoT}	K_n^{IoT}	K_p^G	K_c^G
Initial	(S_{i-1}, S_i)	-1	K_{i-1}	K_i	K_{i-1}	K_i
After M_1	(S_{i-1}, S_i)	-1	K_{i-1}	K_i	K_{i-1}	K_i
After M_2	(S_i, S_i)	0	K_i	K_{i+1}	K_{i-1}	K_i
After $\hat{K}_{i+1}$	(S_i, S_{i+1})	-1	K_i	K_{i+1}	K_i	K_{i+1}
After M_4	(S_{i+1}, S_{i+1})	0	K_{i+1}	K_{i+2}	K_i	K_{i+1}

Using the same approach as in Theorem 1, we can prove that the states of the IoT gateway and the CA in the link between the IoT gateway and the CA are updated and synchronized after a complete run of the protocol and the two communicating entities agree on a new session key $\mathcal{K}_{CA-G}$.

B. Security Analysis of SKAFS using AVISPA

In what follows, we provide our security analysis of the proposed protocol using SPAN+AVISPA (Security Protocol ANimator for Automated Validation of Internet Security Protocols and Applications) framework [44]. AVISPA is used for modeling and formally analyzing security protocols. AVISPA ensures that the security protocol is free from active and passive attacks. Furthermore, it determines whether the protocol is vulnerable to replay and Man-in-the-Middle security attacks.

High-Level Protocol Specification Language (HLPSL) is used to describe security protocols in AVISPA. Afterward, the HLPSL script is translated into an Intermediate Format (IF) specification using the HLPSL2IF translator. Such an IF is analyzed by back-end tools to generate the Output Format (OF). AVISPA comprises four back-end tools namely: On-the-fly Model-Checker (OFMC), Constraint Logic based Attack Searcher (CL-AtSe), SAT-based Model-Checker (SATMC), and Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP). Only OFMC and CL-AtSe tools support exclusive-OR operations.

HLPSL describes each protocol entity in a module called a basic role which specifies what information the entity possesses, its initial state, and the transition between the states. An intruder is modeled using the channel (dy) which stands for the Dolev-Yao intruder model [42].

In our analysis of SKAFS we used the CL-AtSe back-end tool as it supports the exclusive-OR operations. The generated OF shown in Fig. 7 illustrates that the protocol is safe under the CL-AtSe back-end. This means that SKAFS is secure against Man-in-the-middle attacks and satisfies the secrecy of the session key, the secrecy of IoT ID, and the mutual authentication between the IoT device, IoT gateway, and the CA as specified in the environment role.

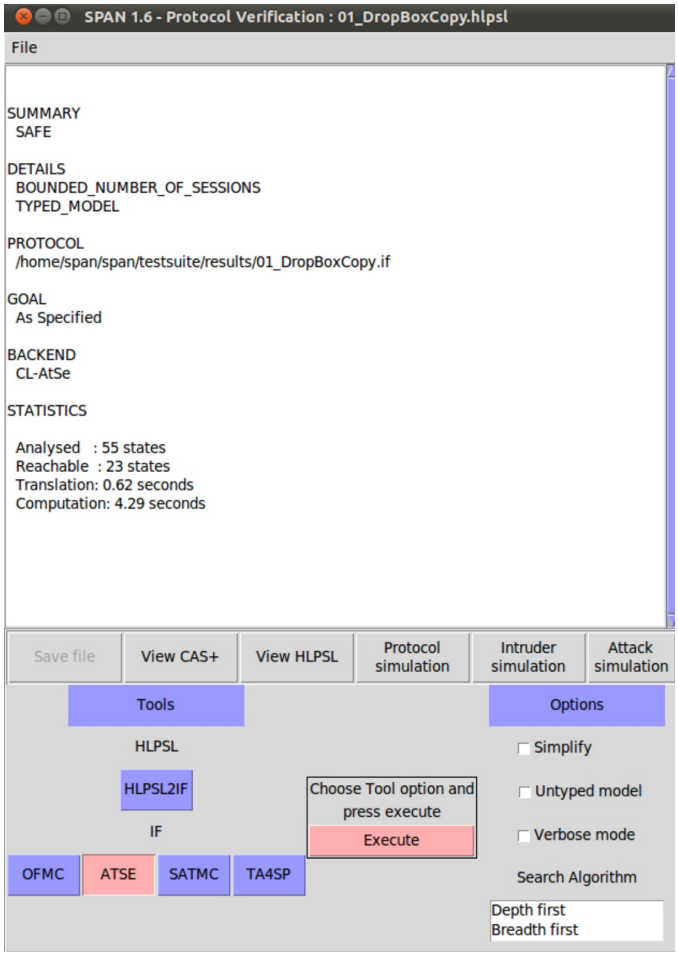

Fig. 7: AVISPA results for the specified goals (i.e., the secrecy of the session key, the secrecy of IoT ID, and authentication of the IoT device, the IoT gateway, and the CA).

C. Formal Security Analysis

In this section, we start by modeling the adversarial capabilities. Then, we provide the formal security proofs for our protocol. For simplicity, we proceed with our proof by assuming that a practical PUF on the IoT device followed by fractional hamming distance measurement on the IoT gateway side can be modeled as an ideal PUF as in [16], [25].

1) Adversarial Oracles: Under the CK-threat model discussed in Section IV-B, an adversary $\mathcal{A}$ has control over the wireless communication channel between the IoT device and the IoT gateway. Furthermore, $\mathcal{A}$ has access to several oracles:

- **Launch**(1^λ)**:** Creates a fresh protocol instance with identifier π.
- **Send IoT**(m, IoT)**:** Models the ability of the adversary to act as a legitimate IoT gateway. In this context, $\mathcal{A}$ sends a message m to IoT in the protocol instance π, and IoT responds with message m'.

- **Send Gateway**(m, G) **:** Models the adversary's capacity to act as a legitimate IoT device. In this context, $\mathcal{A}$ sends message m to the IoT gateway G in the protocol instance π, and G responds with message m'.
- **Execute**(IoT, G)**:** Models the adversary's ability to observe the radio communication channel and intercept a protocol instance π between an IoT device and an IoT gateway. It generates the transcript $\mathcal{T}$ of the transmitted messages of the protocol instance π during its execution between IoT and G.
- **SSReveal**(U)**:** Allows the adversary $\mathcal{A}$ to learn the session state information inside the entity U.
- **SKReveal**(U)**:** Allows the adversary $\mathcal{A}$ to learn the session key of the entity U.
- **Corrupt**(U)**:** Allows $\mathcal{A}$ to learn the long-term private key of the entity U.

  It is worth noting that according to [45], a physical attack on memory is always invasive and ultimately destroys its on-chip neighboring PUF. Therefore, the **SSReveal, SKReveal**, and **Corrupt** oracles can be invoked only once on an IoT device. $\mathcal{A}$ cannot access the PUF output afterward and the IoT is rendered useless. This assumption is consistent with [11].

In what follows, we proceed with our security proofs for mutual authentication, IoT device anonymity and hardware compromise resilience.

Theorem 2. *Our proposed SKAFS protocol achieves mutual authentication between the IoT device and the IoT gateway.*

Proof. An adversary $\mathcal{A}$ may want to impersonate a legitimate IoT device or a legitimate IoT gateway. This can be modeled by the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1) In the training phase, the challenger $\mathcal{C}$ chooses a legitimate IoT device IoT and gateway G.
2) $\mathcal{A}$ interacts with SKAFS (i.e., IoT and G) using the adversary oracles.
3) After the training phase, $\mathcal{A}$ notifies the challenger $\mathcal{C}$.
4) $\mathcal{A}$ calls **Send Gateway** oracle to impersonate IoT.

$\mathcal{A}$ wins the game if it can send a valid authentication message M_1 to impersonate a legitimate IoT device IoT to the IoT gateway G. The adversarial advantage against the mutual authentication is denoted by $adv^{auth}_{SKAFS}(\mathcal{A})$.

Assuming a PPT adversary $\mathcal{A}$ who can break the mutual authentication of SKAFS, this adversary can function as a subroutine in a deterministic algorithm of an adversary $\mathcal{D}$ to break the indistinguishability property of the PUF. For impersonating an IoT, $\mathcal{D}$ intercepts r_1, randomly generates r_2 and uses $\mathcal{A}$ to invoke the **SSReveal** oracle on IoT to know the current challenge C_i. $\mathcal{D}$ passes r_1 and r_2 to $\mathcal{A}$ where $\mathcal{A}$ generates the authentication message M_1. Then, $\mathcal{D}$ challenges PUF (i.e., in the PUF indistinguishability game) with C_i and gets R_b as the response. $\mathcal{D}$ outputs $b = 1$ when $H(R_b, r_1, r_2) = M_1$ and $b = 0$ otherwise (i.e., $adv^{IND}_{PUF}(\mathcal{D}) = adv^{auth}_{SKAFS}(\mathcal{A})$). Since we assume an ideal and secure PUF, therefore, the advantage of $\mathcal{A}$ in forging M_1 message and impersonating an IoT device is $adv^{auth}_{SKAFS}(\mathcal{A}) = adv^{IND}_{PUF}(\mathcal{D}) \leq \epsilon$.

For impersonating an IoT gateway to the IoT device, $\mathcal{D}$ intercepts r_1 and D_{IG} and uses $\mathcal{A}$ to invoke **SSReveal** oracle on IoT to know the current challenge C_i and r_3. $\mathcal{D}$ passes r_1 and D_{IG} to $\mathcal{A}$ where $\mathcal{A}$ generates the authentication message M_2. Then, $\mathcal{D}$ challenges the PUF (i.e., in the PUF indistinguishability game) with C_i and gets R_b as the response. $\mathcal{D}$ outputs $b = 1$ when $H(R_b, D_{IG}, r_1, r_3) = M_2$ and $b = 0$ otherwise (i.e., $adv^{IND}_{PUF}(\mathcal{D}) = adv^{auth}_{SKAFS}(\mathcal{A})$). Since we assume an ideal and secure PUF, the advantage of $\mathcal{A}$ in forging M_2 message and impersonating an IoT gateway is $adv^{auth}_{SKAFS}(\mathcal{A}) = adv^{IND}_{PUF}(\mathcal{D}) \leq \epsilon$. Therefore, SKAFS achieves mutual authentication. □

Theorem 3. *Our proposed SKAFS protocol is privacy-preserving.*

Proof. The anonymity of the sender means that an attacker cannot sufficiently identify the sender within the sender's anonymity set, while unlinkability of two or more items of interest (e.g., messages, and actions) means that an attacker cannot sufficiently distinguish if these two items are related or not [46]. Unlinkability is a sufficient condition of anonymity [47]. Therefore, when SKAFS achieve unlinkability between the sessions, it also achieves the anonymity of the IoT device.

SKAFS is privacy-preserving (i.e., unlinkable) if a PPT adversary $\mathcal{A}$ cannot correlate two successful authentication requests and responses of an IoT device with a gateway. This can be modeled by the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1) In the training phase, the challenger $\mathcal{C}$ chooses a gateway G and two legitimate IoT devices IoT_0 and IoT_1.
2) $\mathcal{A}$ interacts with SKAFS (i.e., IoT_0, IoT_1 and G) using the adversary oracles.
3) After the training phase, $\mathcal{A}$ notifies $\mathcal{C}$.
4) $\mathcal{C}$ samples a bit $b \xleftarrow{r} \{0, 1\}$ and selects IoT device IoT_b
5) $\mathcal{A}$ interacts with IoT_b and G.
6) $\mathcal{A}$ outputs her guess $b' \in \{0, 1\}$ for b. $\mathcal{A}$ wins the game if $b' = b$.

The advantage of $\mathcal{A}$ against the anonymity of SKAFS is defined as $adv^{unlink}_{SKAFS}(\mathcal{A}) = |\mathsf{Pr}(b' = b) - 1/2|$. SKAFS is privacy-preserving if a PPT $\mathcal{A}$ cannot distinguish between the obfuscated identities of two IoT devices (i.e., the adversary advantage $adv^{unlink}_{SKAFS}(\mathcal{A}) \leq \epsilon$).

Let us assume a PPT adversary $\mathcal{A}$ who breaks the anonymity of SKAFS and can distinguish between the obfuscated identities of two IoT devices. This implies that $\mathcal{A}$ can distinguish between obfuscated identities of an IoT device and a random sequence. In what follows, we show that $\mathcal{A}$ can be used by adversary $\mathcal{D}$ to break the PUF indistinguishability property. $\mathcal{D}$ uses $\mathcal{A}$ to invoke **SSReveal** and **Corrupt** oracles on an IoT device with identity $\mathtt{ID}$ to get the fixed challenges C_{F0}, C_{F1} and T. Then, in the PUF indistinguishability game, in the training phase, $\mathcal{D}$ sends C_{F0} to get $\mathcal{P}(C_{F0})$ and in the challenge phase, $\mathcal{D}$ sends C_{F1} and get R_b. $\mathcal{D}$ passes r_1, r_2^*, and $\mathtt{ID}^* = H(r_1, r_2^* \oplus T \oplus \mathcal{P}(C_{F0}) \oplus R_b) \oplus \mathtt{ID}$ to $\mathcal{A}$. If $\mathtt{ID}^*$ is a valid obfuscated identity, $\mathcal{A}$ outputs $b = 1$ as it can distinguish the obfuscated identity of an IoT device with identity $\mathtt{ID}$ from a random sequence. Otherwise $\mathcal{A}$ outputs $b = 0$. In the

PUF indistinguishability game, $\mathcal{D}$ outputs $\mathcal{A}$'s guess b (i.e., $adv_{PUF}^{IND}(\mathcal{D}) = adv_{SKAFS}^{unlink}(\mathcal{A})$). Since we assume an ideal and secure PUF, then $adv_{SKAFS}^{unlink}(\mathcal{A}) = adv_{PUF}^{IND}(\mathcal{D}) \leq \epsilon$, therefore, SKAFS is privacy-preserving. □

Theorem 4. *Launching a physical attack on the IoT device and having access to the stored information does not leak any additional information regarding the CA long-term Key K.*

Proof. Hereby, we prove that the long-term key K does not reside on the IoT device memory at any time. An adversary $\mathcal{A}$ with access to the **SSReveal, SKReveal, Corrupt** oracles does not get any additional information on the long-term key K more than what it gets by performing a passive attack on the communication channel (i.e., invoking the **Execute** oracle). Specifically, an adversary can compromise the IoT device before the $\mathcal{P}_F(CF_0)$ to get the random r_2 or after the $\mathcal{P}_F(CF_0)$ to get r_2 and $r_2 \oplus \mathcal{P}_F(C_{F0})$ or after the 2^{nd} PUF call $\mathcal{P}_F(C_{F1})$ to get $r_2 \oplus \mathcal{P}_F(C_{F0}) \oplus \mathcal{P}_F(C_{F1})$. Subsequently, compromising an IoT device enable $\mathcal{A}$ to gain access to one of these sets.

1) r_2, r_3, ID, T_j
2) r_2, r_3, ID, ID*, T_j
3) r_2, r_3, ID, ID*, K$_i$, T_j
4) r_2, r_3, ID, ID*, K$_i$, K$_i^*$, T_j
5) r_2, r_3, ID, ID*, K$_i$, K$_i^*$, $r_2 \oplus \mathcal{P}_F(C_{F0})$, T_j
6) r_3, ID, ID*, K$_i$, K$_i^*$, $r_2 \oplus \mathcal{P}_F(C_{F0})$, $r_2 \oplus K$, T_j
7) r_3, r_3^*, ID, ID*, K$_i$, K$_i^*$, $r_2 \oplus K$, T_j

Since the cloud long-term key $K = \mathcal{P}_F(C_{F0}) \oplus \mathcal{P}_F(C_{F1}) \oplus T_i$, getting r_2, r_3, ID, T_j leaks no information on the cloud key K. Getting r_2, r_3, ID, ID*, T_j is equivalent to getting r_2, r_3, ID, ID*, T_j , $H(r_1, r_2)$. Under the assumption of a preimage-resistant hash function, the additional information is of no value to the attacker. Getting r_2, r_3, ID, ID*, K$_i$, T_j is equivalent to getting r_2, r_3, ID, ID*, K$_i$, T_j, $H(r_1, r_2)$. Similar to the previous case, the additional information is of no value to the attacker. Getting r_2, r_3, ID, ID*, K$_i$, K$_i^*$, T_j (resp. r_2, r_3, ID, ID*, K$_i$, K$_i^*$, $r_2 \oplus \mathcal{P}_F(C_{F0})$, T_j) is equivalent to getting r_2, r_3, ID, ID*, K$_i$, K$_i^*$, T_j, K_p (resp. r_2, r_3, ID, ID*, K$_i$, K$_i^*$, $\mathcal{P}_F(C_{F0})$, T_j). Assume the length of the PUF response to be l_r and $\mathcal{P}_F(C_{F1})$ to be a random value of entropy l_r. Since $K = \mathcal{P}_F(C_{F0}) \oplus \mathcal{P}_F(C_{F1}) \oplus T_i$, therefore, the cloud long-term key K is still a random value of entropy l_r. Similarly, getting r_3, ID, ID*, K$_i$, K$_i^*$, $r_2 \oplus \mathcal{P}_F(C_{F0})$, $r_2 \oplus K$, T_j is equivalent to getting r_3, ID, ID*, K$_i$, K$_i^*$, $r_2 \oplus \mathcal{P}_F(C_{F0})$, $r_2 \oplus K$, $\mathcal{P}_F(C_{F1})$, T_j. Hence, K remains a random value with entropy l_r. Getting r_3, r_3^*, ID, ID*, K$_i$, K$_i^*$, $r_2 \oplus K$, T_j does not leak information on the long-term key K. □

2) Perfect Forward Secrecy: A protocol achieves perfect forward secrecy if compromising the long-term key or the current session key does not lead to the disclosure of the past session keys. In our protocol, the session key for the session i is computed as $K_S = H(r_1, r_3, K_i)$. The IoT gateway generates a random number r_1 and sends it to the IoT device in the clear. The IoT device generates a random number r_3 and obfuscates it using the instantaneous key K_i, which is randomly generated by the PUF circuit and obfuscated with the previous-session PUF key K_{i-1}. Therefore, this scheme ensures perfect forward secrecy because the past session keys are independent of the long-term key and cannot be derived from a compromised session key.

3) Backward Secrecy: A protocol is said to achieve the backward secrecy property if an adversary who has access to the protocol state values of the current session, cannot calculate the past session keys. We have listed in Theorem 4 all of the state information of the IoT device during the authentication phase. Since the fresh session key depends on randoms freshly generated by the IoT gateway, the IoT device, and the embedded PUF $K_s = H(r_1, r_2, K_i)$, the previous session keys are still secured and the protocol achieves the backward secrecy property.

4) Replay Attacks: After receiving a hello message from the IoT device, the IoT gateway generates the random r_1 and sends it to the IoT device. Subsequently, the IoT device generates the randoms r_2 and r_3. Afterwards, authentication messages M_1, M_2 and M_4 incurs the randoms r_1, r_3. Therefore, any replay attack of a previous authentication request of $r_1^{'}$ and $r_3^{'}$ values will not be accepted by the IoT device with fresh r_3 nor by the IoT gateway with fresh r_1.

5) Attacks Exploiting the IoT Gateway:

A) *Privilege Insider Attack.* According to [48], an insider attack is a malicious attack caused by an entity inside the organization. In our case, we consider the privileged insider attack originating from the IoT gateway. We exclude the CA and consider it as a trusted authority like in the IoT-Centric Cloud paradigm [49], [50]. If the IoT gateway can deviate from the design goals during the mutual authentication with the IoT device and gets access to extra information, we would say that SKAFS is prone to privileged insider attack.
The requesting IoT sends M_1, ID*, r_2^*, r_3^* and K_i^* to the IoT gateway which in turns forwards it to the CA. Note that, ID* is obfuscated with H(r_1, r_2), r_2^* is obfuscated with the long-term key K, r_3^* is obfuscated with the session key K_i and the current session key on the IoT device K_i^* is obfuscated with the previous session key K_p. The cloud admin identifies the requesting IoT device and responds with two-step previous key $K_{p-1} = K_{i-2}$, previous Key $K_p = K_{i-1}$, and current key $K_c = K_i$ to the IoT gateway for synchronization and session key establishment. Therefore, with this additional information, the IoT gateway gets access to the current session key and r_3 without any extra information on r_2 or the identity of the requesting IoT. Moreover, since the IoT gateway does not have access to the identity of the requesting IoT or the long-term key K, it cannot launch an impersonation attack on other IoT devices to other IoT gateways. Therefore, our proposed protocol is immune to privileged insider attacks.

B) *Stolen Verifier Attack.* According to [51], a stolen verifier attack is where an adversary who has stolen the verifier token for the user password can impersonate that user to the cloud server later. By employing a dictionary attack, the stolen verifier attack can be achieved on low-entropy passwords. Here, in our protocol, the IoT gateway does not store verifier data. Instead, it obtains it from the CA admin after receiving the IoT authentication request. Moreover,

the IoT device uses the DPUF response for generating the authentication keys which have more entropy and are updated after each session. Thus, stealing the verifier token from the IoT gateway is not possible unless for the running session only. Even if the adversary gets access to the current verifier, this verifier will be useless for the next session which requires an updated authentication key.

VII. Performance Analysis and Comparison

Generally, IoT devices are limited in their storage and computation resources. Thus, it is important to consider the efficiency of SKAFS by analyzing its storage and computation requirements. In our evaluation, the storage overhead is indicated by the size of the memory required by the IoT device and the CA to store the secrets needed during the mutual authentication and key establishment phases. The computation requirement is measured by the cryptographic operations on the IoT device and the IoT gateway in order to achieve mutual authentication and establish the session key before the actual exchange of the data. The communication overhead between the IoT device and the IoT gateway is the messages exchanged between them before the actual exchange of the data. The performance evaluation results are summarized in Table V.

TABLE V: Storage, Computation and Communication Cost

Description	Value
IoT storage	64 bytes
Cloud storage	$N_{IoT} \times 4 \times 16 + N_G \times 2 \times 16$ bytes
IoT computation	2 RNG, 7 hash, 2 $\mathcal{P}_D(\,)$, 2 $\mathcal{P}_F(\,)$
Gateway computation	2 RNG, 7 hash, 2 Enc, 1 Dec, 2 FHD
CA computation	4 hash, 1 Enc, 2 Dec
Communication IoT-G	192 bytes
Communication G-CA	320 bytes

A. Communication Cost

Throughout this section, we consider SHA-256 and AES-CBC encryption mode as in [29]. For the identity of the IoT device, we assume a 128-bit identity and 1-bit D_{IG}. For the PUF circuit parameters, we assume a challenge of 128 bits as stated in [14] and a PUF response of 128 bits, as stated in [29].

In the mutual authentication phase, the IoT gateway sends r_1 which is a 128-bit message and the IoT device replies with a 6×128 bits message of $\{M_1,\ \texttt{ID}^*,\ r_2^*,\ K_i^*,\ r_3^*\}$. Then, the IoT gateway sends a 1-bit D_{IG} and the authentication message M_2 of 256 bits. Later, in the synchronization phase, the IoT device sends the next session key K_{i+1} of 128 bits, and the IoT gateway confirms with a 256-bit M_4. Thus, in total, the communication cost between the IoT device and the IoT gateway is 192 bytes plus 1-bit D_{IG}. On the other side, the communication cost between the IoT gateway and CA is 320 bytes.

B. Storage Requirements

For each IoT device, the server needs storage of 4×128 bits for keeping the IoT's identity ID, K_{p-1}, K_p, and K_c. Also, the server requires 2×128 bits for storing the master key and the session key with each deployed IoT gateway. On the other side, the IoT device stores its identity ID, the bit string T, the session's challenge C_i, and the previous key K_p. Thus, in total, the IoT device requires a total storage of 4×128 bits.

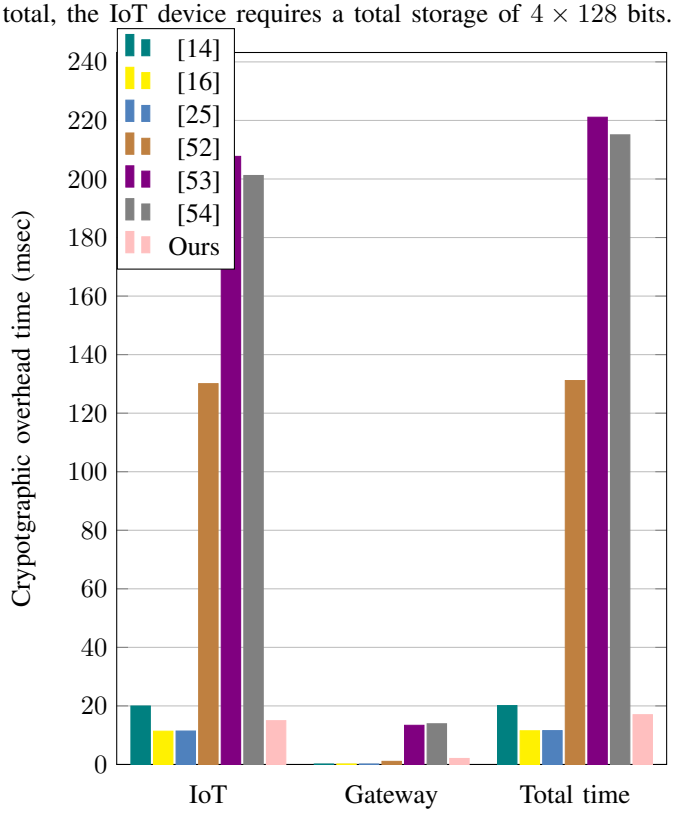

Fig. 8: The cryptographic overhead time for schemes [14], [16], [25], [52], [53], [54] and SKAFS. The time for the IoT device and the gateway are computed on Raspberry Pi 1 and Raspberry Pi 4, respectively.

C. Computation Cost

Let RNG denote the operation corresponding to invoking the random number generator. In the mutual authentication phase, the IoT device requires 2 RNG, 1 $\mathcal{P}_D(\,)$ call, 2 $\mathcal{P}_F(\,)$ call and 2 hash-based computations. Later, in the synchronization phase, the IoT device requires at most 2 $\mathcal{P}_D(\,)$ calls and 5 hash-based computations. In total, the IoT requires 2 RNG, 3 $\mathcal{P}_D(\,)$ call, 2 $\mathcal{P}_F(\,)$ call and 7 hash-based computations.

In the mutual authentication phase, the IoT gateway performs 2 RNG, 1 encryption operations, 1 decryption operations, 2 fractional hamming computation, and 7 hash-based computations. Alternatively, CA performs 1 decryption operations, and 1 encryption operations. In the synchronization phase, the IoT gateway performs 2 hash-based computation and 1 encryption operation. In total, the IoT gateway requires 2 RNG, 2 encryption operations, 1 decryption operations, 2 fractional hamming distance computation, and 9 hash-based computation.

D. Performance benchmarking and comparison

To estimate the cryptographic overhead time induced in our protocol, we run the utilized cryptographic primitives in our

protocol on a Raspberry Pi 4 Model B/8GB equipped with a 64−bit Quad-core ARM Cortex-A72 processor clocked at 1.5 GHz. Moreover, we consider a Raspberry Pi 1 Model B+ with 512 MB of RAM and a 0.7 GHz ARM11 processor for resource-constrained IoT devices. The average cryptographic time after 1,000,000 runs of the cryptographic primitives on the Raspberry Pi 1 is 0.03376 msec for the Hash, 0.1185 msec for the HMAC, 0.0874 msec for the random number generation, and 0.558 msec for the AES encryption. Table VI shows the average time (msec) on the Raspberry Pi 1 and the Raspberry Pi 4. Moreover, for PUF execution time, we based our calculation on the benchmark in [16] where the execution time of arbiter PUF, SRAM PUF, and DRAM PUF implementation on SASEBO-GII board, consisting of a Xilinx XC5VLX30 FPGA device with a system clock of 1.846 MHz and 16K byte of program memory, is reported ro be 3.945, 2.236, and 3.3334, respectively.

TABLE VI: Average cryptographic overhead time (msec) [3]

Description	Notation	Rasp. Pi 1	Rasp. Pi 4
Hash operation [4]	T_h	0.03376	0.001245
MAC operation [5]	T_{MAC}	0.1185	0.00464
RNG	T_{RNG}	0.0874	0.0353
Symmetric Enc/Dec [6]	T_s	0.558	0.03052
Fractional Hamming Distance	FHD	2.26	0.094
Bilinear Pairing operation [7]	T_b	196.29	12.552
Point Addition [8]	T_{PA}	1.199	0.073
Scalar Point Multiplication [8]	T_{PM}	3.20	0.297

[3]: The average cryptographic overhead time (msec) after 1,000,000 runs on Raspberry Pi 1 and Raspberry Pi 4.
[4]: We consider SHA-256 over 1024 bytes data
[5]: We consider SHA-256 as the underlying message-digest algorithm
[6]: We consider AES-CBC mode with a key size of 128 bits.
[7]: We use the Python library bplib, which implements a bilinear pairing over a Barreto-Naehrig curve [55].
[8]: Using the fastecdsa Python library.

Furthermore, we implement SKAFS using socket programming to measure its end-to-end latency and simulate the flow of our protocol messages between the IoT device, IoT gateway, and the CA in a real-time experiment. The Raspberry Pi 1 represents the IoT device while the Raspberry Pi 4 represents the IoT gateway, and an Intel laptop 11th Gen Core i7-11800H clocked at 2.3 GHz with 16 GB RAM, acts as the server. On average, the end-to-end authentication scheme incurs a latency overhead of 33.93 ms. These results illustrate the efficiency of the protocol in the IoT-edge-cloud paradigm. Our source code for the Python implementation of the socket programming experiment is available on the GitHub repository at https://github.com/IoT-SKAFS/SKAFS.

We compare SKAFS with schemes in [14]–[16], [25], [52]–[54] in Fig. 8, based on the average cryptographic overhead latency, as reported in Table VII. The comparison shows that SKAFS outperforms the protocols in [14], [15], [52]–[54], and is comparable to those in [16], [25] in terms of execution time. However, the protocols in [14], [16], [25], [54] have higher communication cost than SKAFS as shown in Fig. 9.

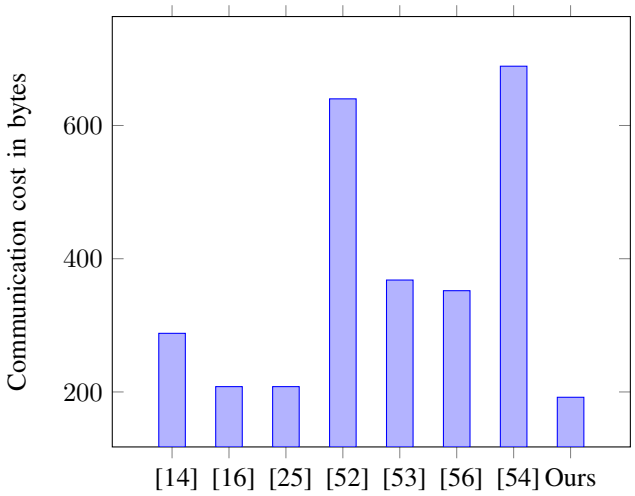

Fig. 9: Communication cost of [14], [16], [25], [52], [53] , [56], [54] and SKAFS.

In order to demonstrate the storage reduction in SKAFS, we assume the synchronization set required in other protocols to thwart desynchronization-based DoS attacks is a set of s pseudo-identities. In Fig. 10, we illustrate the scaling of storage requirements of the CA per IoT device with the size of the synchronization set for the protocols [11], [15], [16], [25] as well as SKAFS.

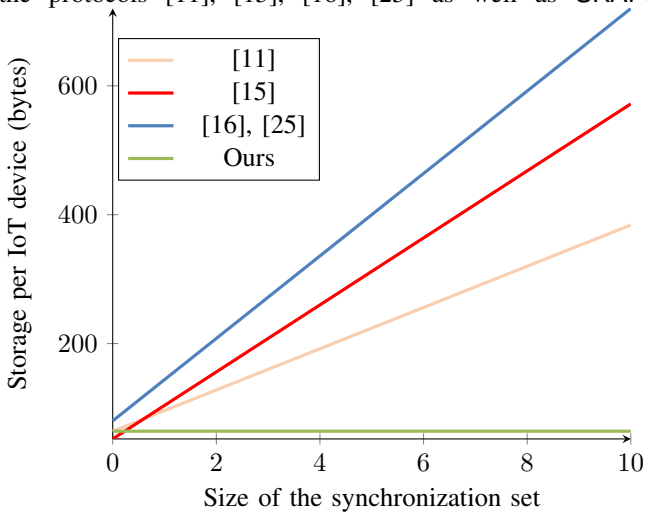

Fig. 10: The scaling of the storage required at CA per IoT device with the size of the synchronization set.

Protocols [25] and [16] require storage of $(80 + 64 \times s)$ bytes per IoT device to keep their identities, keys, and the corresponding challenge-response pairs. On the other hand, in [15], CA needs $(52 + 52 \times s)$ bytes per IoT device, and in [11], a storage of $(64 + 32 \times s)$ is required for every IoT device. Moreover, the protocols in [16] and [25] demand a storage of 48 bytes on the IoT device for keeping the identities, corresponding keys, and challenges. The work in [15] requires $48 + s \times 32$ bytes, while that in [11] requires $(182 + s \times 174)$ bytes for storing the instantaneous pseudo-identities, challenge, and helper data along with the synchronization identities set. Note that, in the aforementioned protocols, a repeated desynchronization attack on an IoT device until running out of synchronization identity, requires the IoT device to re-register

TABLE VII: Performance comparison based on the computation complexity

Protocol	IoT computation (msec) on Raspberry Pi 1	Gateway computation (msec) on Raspberry Pi 4
[14]	4 APUF + 5 T_{RNG} + 6 T_s + 3 T_{MAC} +2 T_h = 19.968	4 T_s + 2 T_{MAC} + 2 T_h = 0.13385
[16]	6 T_h + FHD + 2 DPUF + FPUF = 11.36536	T_{RNG} + FHD + 7 T_h = 0.138015
[25]	5 T_h + FHD + 2 DPUF + FPUF + T_{RNG} = 11.419	T_{RNG} + FHD + 6 T_h = 0.13677
[52]	1 FPUF + 2 T_{PA} + 12 T_h + 2 T_{PM} + 1 T_b = 207.72	9 T_h + 1 T_b + 2 T_{PM} + 3 T_{PA} = 13.376205
[53]	1 FPUF + 2 T_{PA} + 8 T_h + 1 T_b =201.194	8 T_h + 1 T_b + 4 T_{PM} + 2 T_{PA} = 13.89596
[54]	25 T_{RNG} + 48 FPUF + 163 T_h + 27 T_s = 130.08	35 T_h + 26 T_s + 48 T_{MAC} = 1.05882
Ours	2 T_{RNG} + 3 DPUF + 2 FPUF + 9 T_h= 14.95	2 T_{RNG} + 2 FHD + 3 T_s + 7 T_h = 2.045515

again with the CA. Nevertheless, SKAFS achieves mutual authentication beside other security properties (i.e., resilience to physical attacks, desynchronization-based DoS attacks, and PUF-modeling attacks) with a constant storage requirement of 64 bytes on the IoT device and storage requirement of 64 bytes/IoT device at CA.

VIII. Conclusion

In this paper, we introduced a mutual authentication protocol, named SKAFS, for secure computation offloading services between an IoT device and an IoT gateway in the IoT-edge-cloud paradigm. SKAFS achieve anonymity of the IoT device, forward secrecy, and is resilient to physical and desynchronization-based Dos attacks. SKAFS makes use of PUFs to thwart physical attacks that may lead to compromising the long-term authentication key. We have formally proved that under the assumption of the indistinguishability of secure PUFs, SKAFS is privacy-preserving, and achieves mutual authentication and forward secrecy. Moreover, the synchronization scheme ensures that both the IoT device and the gateway are updated and synchronized by the end of a complete run of the protocol. We have provided performance analysis to show that the protocol is efficient in terms of storage, communication cost, and computation complexity which makes SKAFS compatible with the IoT-edge-cloud paradigm.

References

[1] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, D. Leaf *et al.*, "NIST cloud computing reference architecture," *NIST special publication*, vol. 500, no. 2011, pp. 1–28, 2011.

[2] J. Pan and J. McElhannon, "Future edge cloud and edge computing for internet of things applications," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, 2017.

[3] E. Mingozzi, G. Tanganelli, C. Vallati, and V. Di Gregorio, "An open framework for accessing things as a service," in *2013 16th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. IEEE, 2013, pp. 1–5.

[4] S. N. Shirazi, A. Gouglidis, A. Farshad, and D. Hutchison, "The extended cloud: Review and analysis of mobile edge computing and fog from a security and resilience perspective," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2586–2595, 2017.

[5] F. Wang, Y. Xu, L. Zhu, X. Du, and M. Guizani, "LAMANCO: A Lightweight Anonymous Mutual Authentication Scheme for N-Times Computing Offloading in IoT," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4462–4471, 2018.

[6] H. Liu, Y. Zhang, and T. Yang, "Blockchain-enabled security in electric vehicles cloud and edge computing," *IEEE Network*, vol. 32, no. 3, pp. 78–83, 2018.

[7] M. Wazid, A. K. Das, S. Shetty, J. JPC Rodrigues, and Y. Park, "LDAKM-EIoT: Lightweight device authentication and key management mechanism for edge-based IoT deployment," *Sensors*, vol. 19, no. 24, p. 5539, 2019.

[8] M. Nakkar, R. AlTawy, and A. Youssef, "Lightweight Broadcast Authentication Protocol for Edge-based Applications," *IEEE Internet of Things Journal*, 2020.

[9] "Gartner," https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io, [Online; accessed 26-November-2021].

[10] M. Conti, A. Dehghantanha, K. Franke, and S. Watson, "Internet of Things security and forensics: Challenges and opportunities," 2018.

[11] P. Gope, J. Lee, and T. Q. Quek, "Lightweight and practical anonymous authentication protocol for RFID systems using physically unclonable functions," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2831–2843, 2018.

[12] G. Bansal, N. Naren, V. Chamola, B. Sikdar, N. Kumar, and M. Guizani, "Lightweight mutual authentication protocol for V2G using physical unclonable function," *IEEE Transactions on Vehicular Technology*, 2020.

[13] S. Li, T. Zhang, B. Yu, and K. He, "A Provably Secure and Practical PUF-based End-to-End Mutual Authentication and Key Exchange Protocol for IoT," *IEEE Sensors Journal*, 2020.

[14] M. N. Aman, K. C. Chua, and B. Sikdar, "Mutual authentication in IoT systems using physical unclonable functions," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1327–1340, 2017.

[15] P. Gope and B. Sikdar, "Lightweight and privacy-preserving two-factor authentication scheme for IoT devices," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 580–589, 2018.

[16] ——, "A Privacy-Aware Reconfigurable Authenticated Key Exchange Scheme for Secure Communication in Smart Grids," *IEEE Transactions on Smart Grid*, 2021.

[17] S. Kalra and S. K. Sood, "Secure authentication scheme for IoT and cloud servers," *Pervasive and Mobile Computing*, vol. 24, pp. 210–223, 2015.

[18] S. Kumari, M. Karuppiah, A. K. Das, X. Li, F. Wu, and N. Kumar, "A secure authentication scheme based on elliptic curve cryptography for IoT and cloud servers," *The Journal of Supercomputing*, vol. 74, no. 12, pp. 6428–6453, 2018.

[19] T. Shah and S. Venkatesan, "Authentication of IoT device and IoT server using secure vaults," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 819–824.

[20] A. Tewari and B. Gupta, "A lightweight mutual authentication protocol based on elliptic curve cryptography for IoT devices," *International Journal of Advanced Intelligence Paradigms*, vol. 9, no. 2-3, pp. 111–121, 2017.

[21] P. Tedeschi and S. Sciancalepore, "Edge and Fog Computing in Critical Infrastructures: Analysis, Security Threats, and Research Challenges," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2019, pp. 1–10.

[22] M. Seifelnasr, M. Nakkar, A. Youssef, and R. AlTawy, "A Lightweight Authentication and Inter-Cloud Payment Protocol for Edge Computing," in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*. IEEE, 2020, pp. 1–4.

[23] M. Akgün and M. U. Çağlayan, "Providing destructive privacy and scalability in RFID systems using PUFs," *Ad Hoc Networks*, vol. 32, pp. 32–42, 2015.

[24] M. Seifelnasr, R. A. Tawy, and A. Youssef, "Efficient Inter-cloud Authentication and Micropayment Protocol for IoT Edge Computing," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021.

[25] P. Gope, B. Sikdar, and O. Millwood, "A Scalable Protocol Level Approach to Prevent Machine Learning Attacks on PUF-based Authentication Mechanisms for Internet-of-Medical-Things," *IEEE Transactions on Industrial Informatics*, 2021.

[26] R. Melki, H. N. Noura, and A. Chehab, "Lightweight multi-factor

mutual authentication protocol for IoT devices," *International Journal of Information Security*, vol. 19, pp. 679–694, 2020.

[27] H. N. Noura, R. Melki, and A. Chehab, "Secure and lightweight mutual multi-factor authentication for IoT communication systems," in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*. IEEE, 2019, pp. 1–7.

[28] W. Trappe, "The challenges facing physical layer security," *IEEE communications magazine*, vol. 53, no. 6, pp. 16–20, 2015.

[29] P. Gope, "PMAKE: Privacy-aware multi-factor authenticated key establishment scheme for advance metering infrastructure in smart grid," *Computer Communications*, vol. 152, pp. 338–344, 2020.

[30] M. Bellare, "A Note on Negligible Functions." *Journal of Cryptology*, vol. 15, no. 4, 2002.

[31] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 148–160.

[32] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2004, pp. 523–540.

[33] S. Schulz, A.-R. Sadeghi, and C. Wachsmann, "Short paper: Lightweight remote attestation using physical functions," in *Proceedings of the fourth ACM conference on Wireless network security*, 2011, pp. 109–114.

[34] J. R. Wallrabenstein, "Practical and secure IoT device authentication using physical unclonable functions," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2016, pp. 99–106.

[35] J. Delvaux, "Machine-learning attacks on polypufs, ob-pufs, rpufs, lhs-pufs, and puf–fsms," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 8, pp. 2043–2058, 2019.

[36] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 237–249.

[37] S. Sutar, A. Raha, and V. Raghunathan, "D-PUF: An intrinsically reconfigurable DRAM PUF for device authentication in embedded systems," in *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*. IEEE, 2016, pp. 1–10.

[38] S. Guilley and R. Pacalet, "SoCs security: a war against side-channels," in *Annales des télécommunications*, vol. 59, no. 7-8. Springer, 2004, pp. 998–1009.

[39] M. S. Kirkpatrick, S. Kerr, and E. Bertino, "System on chip and method for cryptography using a physically unclonable function," Jun. 10 2014, uS Patent 8,750,502.

[40] J. Sepulveda, F. Willgerodt, and M. Pehl, "SEPUFSoC: Using PUFs for memory integrity and authentication in multi-processors system-on-chip," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, 2018, pp. 39–44.

[41] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2001, pp. 453–474.

[42] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[43] G. Avoine, S. Canard, and L. Ferreira, "Symmetric-key authenticated key exchange (SAKE) with perfect forward secrecy," in *Cryptographers' Track at the RSA Conference*. Springer, 2020, pp. 199–224.

[44] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani *et al.*, "The AVISPA tool for the automated validation of internet security protocols and applications," in *Computer Aided Verification: 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005. Proceedings 17*. Springer, 2005, pp. 281–285.

[45] B. L. P. Gassend, "Physical random functions," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.

[46] A. Pfitzmann and M. Köhntopp, "Anonymity, unobservability, and pseudonymity—a proposal for terminology," in *Designing privacy enhancing technologies*. Springer, 2001, pp. 1–9.

[47] A. Pfitzmann and M. Hansen, "A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management," 2010.

[48] S. Rajamanickam, S. Vollala, R. Amin, and N. Ramasubramanian, "Insider attack protection: Lightweight password-based authentication techniques using ECC," *IEEE Systems Journal*, vol. 14, no. 2, pp. 1972–1983, 2019.

[49] A. R. Biswas and R. Giaffreda, "IoT and cloud convergence: Opportunities and challenges," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*. IEEE, 2014, pp. 375–376.

[50] J. Srinivas, S. Mukhopadhyay, and D. Mishra, "Secure and efficient user authentication scheme for multi-gateway wireless sensor networks," *Ad Hoc Networks*, vol. 54, pp. 147–169, 2017.

[51] C.-M. Chen and W.-C. Ku, "Stolen-verifier attack on two new strong-password authentication protocols," *IEICE Transactions on communications*, vol. 85, no. 11, pp. 2519–2521, 2002.

[52] U. Chatterjee, V. Govindan, R. Sadhukhan, D. Mukhopadhyay, R. S. Chakraborty, D. Mahata, and M. M. Prabhu, "Building PUF based authentication and key exchange protocol for IoT without explicit CRPs in verifier database," *IEEE transactions on dependable and secure computing*, vol. 16, no. 3, pp. 424–437, 2018.

[53] U. Chatterjee, R. Sadhukhan, V. Govindan, D. Mukhopadhyay, R. S. Chakraborty, S. Pati, D. Mahata, and M. M. Prabhu, "PUFSSL: an OpenSSL extension for PUF based authentication," in *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*. IEEE, 2018, pp. 1–5.

[54] M. Seifelnasr, R. AlTawy, and A. Youssef, "Efficient Inter-Cloud Authentication and Micropayment Protocol for IoT Edge Computing," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4420–4433, 2021.

[55] "bplib 0.0.6," https://pypi.org/project/bplib/, [Online; accessed 26-November-2021].

[56] Y. Zheng and C.-H. Chang, "Secure mutual authentication and key-exchange protocol between PUF-embedded IoT endpoints," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

Mohamed Seifelnasr received the B.Sc. degree in communication, electronics and computer engineering from Helwan University, Cairo, Egypt, in 2012 and M.Sc. degree in computer science from Huazhong University of Science and Technology (HUST), Wuhan, Hubei, China, in 2018. He is currently pursuing the Ph.D. degree at Concordia University, Montreal, Canada. His current research interests include Internet-of-things and edge computing security.

Riham AlTawy received the B.Sc. and M.Sc. degrees from AAST, Egypt, in 2005 and 2008, respectively, and the Ph.D. degree from Concordia University, Canada in 2016. Riham is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Victoria, BC, Canada. Previously, she was an NSERC Postdoctoral Fellow in the Department of Electrical and Computer Engineering, University of Waterloo, ON, Canada. Her research interests focus on IoT security, blockchains, and lightweight cryptography.

Amr Youssef received the B.Sc. and M.Sc. degrees from Cairo University, Cairo, Egypt, in 1990 and 1993 respectively, and the Ph.D. degree from Queens University, Kingston, ON., Canada, in 1997. Dr. Youssef is currently a professor at the Concordia Institute for Information Systems Engineering (CIISE) at Concordia University, Canada. Before joining CIISE, he worked for Nortel Networks, the Center for Applied Cryptographic Research at the University of Waterloo, IBM, and Cairo University. His research interests include cryptology, cybersecurity, and cyber-physical systems security. He has more than 230 referred journal and conference publications in areas related to his research interests. He also served on more than 60 technical program committees of cryptography and data security conferences. He was the co/chair for Africacrypt 2013 and Africacrypt 2020, the conference Selected Areas in Cryptography (SAC 2014, SAC 2006 and SAC 2001).