



Article

SAKAP: SGX-Based Authentication Key Agreement Protocol in IoT-Enabled Cloud Computing

Tsu-Yang Wu ¹, Liyang Wang ¹, Xinglan Guo ¹, Yeh-Cheng Chen ² and Shu-Chuan Chu ^{1,*}¹ College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China² Department of Computer Science, University of California, Davis, CA 001313, USA

* Correspondence: scchu0803@sdust.edu.cn

Abstract: With the rapid development of the Internet, Internet of Things (IoT) technology is widely used in people's daily lives. As the number of IoT devices increases, the amount of data to be processed also increases. The emergence of cloud computing can process the data of IoT devices in a timely manner, and it provides robust storage and computing capabilities to facilitate data resource sharing. Since wireless communication networks are unstable and open, it is easy for attackers to eavesdrop, intercept, and tamper with the messages sent. In addition, authentication protocols designed for IoT-enabled cloud computing environments still face many security challenges. Therefore, to address these security issues, we propose an Intel software-guard-extensions (SGX)-based authentication key agreement protocol in an IoT-enabled cloud computing environment. The goal is to ensure data privacy and sustainable communication between the entities. Moreover, SGX can resist several well-known attacks. Finally, we show the security using the real-or-random model, ProVerif, and informal analysis. We also compare the security and performance of the proposed protocol with existing protocols. The comparison results show that our proposed protocol reduces the communication cost by 7.07% compared to the best one among the current protocols and ensures sufficient security.

Keywords: IoT; cloud computing; authentication; SGX

Citation: Wu, T.-Y.; Wang, L.; Guo, X.; Chen, Y.-C.; Chu, S.-C. SAKAP: SGX-Based Authentication Key Agreement Protocol in IoT-Enabled Cloud Computing. *Sustainability* **2022**, *14*, 11054. <https://doi.org/10.3390/su141711054>

Academic Editors: Marko Hölbl, SK Hafizul Islam, Marimuthu Karupiah and Chien-Ming Chen

Received: 26 July 2022

Accepted: 25 August 2022

Published: 5 September 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The Internet of Things (IoT) [1–4] refers to a network that connects the Internet with any entity according to a specified protocol, which exchanges information and completes the communication through information-sensing equipment to realize the intelligent identification, positioning, and monitoring of entities. With the development of IoT technology, the information-collection feature of the IoT has been applied in many scenarios, such as artificial intelligence [5–8], transportation systems [9–11], smart grids [12], smart cities [13], and health systems [14,15]. IoT technology has increased the efficiency of production methods and improved the quality of people's lives.

With the increase in the number of IoT devices, the generated data have also increased gradually to handle the data generated by IoT devices more effectively, leading to the introduction cloud computing [16–19]. Cloud computing is a form of distributed computing, which provides computing power, database storage, data analysis, and other information technology resources on demand. Cloud computing provides an efficient and convenient method for information and resource sharing, and its combination with the IoT can compensate their respective drawbacks. The IoT can benefit from the powerful storage capacity and computing power of cloud computing. Similarly, combining cloud computing with IoT devices can result in providing new services in real-life scenarios to expand the ability to solve practical problems. The applications of cloud computing and IoT were mentioned in [13,20,21]. In 2020, Kang et al. [20] designed a lightweight authentication and

key agreement (AKA) protocol based on the IoT-enabled cloud computing environment. Huang et al. [13] proposed an AKA protocol that combines IoT and cloud computing and implemented it in a smart city environment in 2021. Iqbal et al. [21] proposed an AKA protocol for an IoT and cloud computing architecture in 2022. The architecture of IoT-enabled cloud computing is shown in Figure 1, and the communication entities include the user, cloud server, and control server. The cloud service provider deploys cloud servers in the region where the cloud service is provided and configures a control server to manage the cloud servers and users. In addition, only legitimate users can select cloud servers using IoT devices to handle large amounts of data.

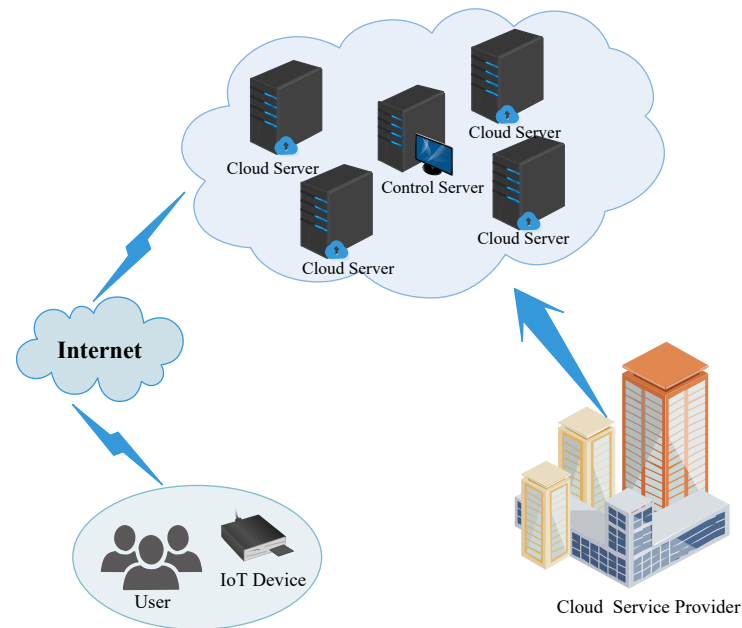


Figure 1. IoT-enabled cloud computing architecture.

However, there are still significant security risks in IoT-enabled cloud-computing environments. For example, malicious attackers can intercept messages on public channels and then tamper with or crack the data information, resulting in the confidentiality, privacy, and integrity of user data not being able to be guaranteed. Moreover, AKA protocols designed in IoT-enabled cloud computing environments are subject to impersonation [22,23], offline password guessing [20], and replay attacks [24]. The presence of these security risks and attacks does not ensure network sustainability.

Intel software guard extensions (SGX) [25–27] can be introduced to improve the security of AKA protocols designed for IoT-enabled cloud computing environments. SGX is an extension of the Intel instruction set, which protects the security of programs in the running state. SGX is divided into a trusted execution environment and an untrusted execution environment. Because a malicious attacker cannot access the trusted execution environment, storing data in this environment ensures data integrity, privacy, and confidentiality. The core of SGX is an enclave of the memory, and it is an encrypted area in the memory address space, which stores the running code and program data. The application program can transmit the data to be calculated to the enclave through the SGX interface for calculation. The enclave then sends the operation results to the application program. It is not affected by malware or other instructions with the highest authority during the entire operational process [28]. Referring to the AKA protocol proposed by Liu et al. [26] in the wireless sensor network architecture and Wu et al. [27] in the Internet of Vehicles and fog computing, we introduce SGX into the IoT-enabled cloud computing environment to design the AKA protocol. Our goal is to ensure data privacy and sustainable communication between the entities. The following are the primary contributions of this paper:

- (1) We propose a lightweight AKA in the IoT-enabled cloud computing environment. In our protocol, the user, cloud server, and control server achieve mutual authentication, and the session key is successfully established for communication.
- (2) We first introduce SGX into an IoT-enabled cloud computing environment and use it on the cloud server and control server. Because SGX has limitations in both storage and computation, we only use it to store the shared key. According to the safety features of SGX, even if an attacker can access the data in memory, he/she cannot obtain the shared key in SGX. Thus, privileged insider attacks are invalid for our protocol.
- (3) We use the real-or-random (ROR) model and ProVerif tool to verify the security of the proposed protocol, and informal security analysis shows that the protocol protects against known attacks.
- (4) Finally, we compare the security and performance with current protocols, and the results show that our protocol ensures greater security under a similar efficiency.

The remainder of this paper is organized as follows. We review the research related to the IoT, cloud computing, and SGX in Section 2. In Section 3, we describe the system model and protocol in detail. Section 4 describes the process by which we used the ROR model, the ProVerif tool, and informal security analysis do assess the security of the proposed protocol. Section 5 describes the comparison between our proposed protocol and existing protocols in terms of both security and performance. The conclusion of the study is presented in Section 6.

2. Related Work

Turkanovic et al. [29] designed an AKA protocol based on the IoT environment in 2014 that utilized lightweight primitives and provided enhanced security. However, Farash et al. [30] found that the protocol could not ensure the anonymity of the user and sensor node and was vulnerable to session key disclosure attacks and man-in-the-middle attacks. Farash et al. [30] designed an improved protocol and declared that the protocol could guarantee secure communication. However, Amin et al. [31] found that Farash et al.'s protocol [30] was susceptible to offline password guessing, smart card theft, user impersonation, and known temporary value disclosure attacks. Similarly, Amin et al. [31] proposed an AKA protocol for anonymity-protected three-factor authentication key exchange. However, Wu et al. [32] discovered that the protocol of Amin et al. [31] was not resistant to sensor capture, session key disclosure, and user impersonation attacks and could not guarantee mutual authentication, and they proposed an AKA protocol based on multiple gateways in the IoT environment.

In 2014, Liu et al. [33] proposed an AKA protocol for sharing privileges and guaranteeing privacy in a cloud computing environment. Tsai and Lo [34] designed a privacy authentication protocol based on a cloud computing environment and used bilinear pairs in their protocol. However, He et al. [35] pointed out that their protocol was not resistant to server impersonation attacks and designed an efficient and private authentication protocol using bilinear pairs. Kumar et al. [36] designed a bidirectional AKA protocol for healthcare systems in a cloud environment using elliptic curves. Lopes and Gond [37] proposed an AKA protocol for device-to-device communication applied to an electronic health system based on cloud computing and declared that the protocol could ensure secure communication between entities. Iqbal et al. [21] proposed an AKA protocol for the IoT and cloud computing, which used elliptic curves and symmetric encryption/decryption. Zhou et al. [23] proposed a lightweight AKA protocol based on the IoT in cloud computing. However, Wang et al. [38] discovered that the protocol did not ensure forward secrecy and was vulnerable to temporary value disclosure and impersonation attacks. Martinez-Pelaez et al. [24] designed an enhanced AKA protocol in the cloud computing environment. However, Yu et al. [39] found that their protocol could not guarantee mutual authentication and user anonymity and that it suffered from session key disclosure and replay and offline password guessing attacks. Kang et al. [20] designed an improved AKA protocol for IoT-enabled cloud computing. However, Huang et al. [13] discovered that Kang et al.'s

protocol [20] was subject to offline password guessing attacks and highlighted the designed redundancy in the user registration step. Wu et al. [40] designed an authentication protocol in IoT-enabled cloud computing and showed that the protocol was resistant to various attacks and provided better security.

In 2016, Costan and Devadas [25] described the architecture and operational mechanism of SGX in detail, provided a detailed description of SGX's public information, and analyzed its security properties. Fisch et al. [41] constructed a provably secure and practical functional encryption mechanism using Intel SGX and showed that the performance of this mechanism exceeded the known encryption schemes. Sun et al. [42] proposed a dynamic network identity authentication scheme using SGX, which can continuously update the key. Conde et al. [43] designed and implemented an identity authentication module based on SGX in a Unix operating system. The module uses an enclave in SGX to process data and improve the security of the module. Song et al. [44] proposed a privacy and anonymity protection authentication scheme based on blockchain and SGX, claiming that the scheme would not reveal users' personal information. Liu et al. [26] designed an AKA protocol that uses SGX based on a wireless sensor network architecture, dynamically updating the authentication credentials, and declared that the protocol achieves better security with less overhead.

3. Proposed Protocol: SAKAP

In this section, we describe the system model and the specific protocol procedure in detail. Table 1 lists the notations used in this protocol.

Table 1. Notations.

Notations	Description
U_i	i -th user
ID_i	Identity of U_i
$PID_i, PSID_j$	Pseudo-identity of U_i and S_j
PW_i, BIO_i	Password and biometric of U_i
S_j	j -th cloud server
SID_j	Identity of S_j
CS	The control server
k_u	Shared key between U_i and CS
k_s	Shared key between S_j and CS
SK	Session key
T_i	Timestamp
$Gen(\cdot)/Rep(\cdot)$	Fuzzy generator/reproduction function
$h(\cdot)$	Secure hash function

3.1. System Model

The system model has three entities: user U_i , cloud server S_j , and control server CS, as shown in Figure 2. Each S_j and CS requires SGX to be installed before deploying the environment. A detailed description of each entity is as follows:

- (1) User (U_i): U_i refers to people who intend to use cloud computing services in the IoT-enabled cloud computing environment. Only legitimate U_i can use cloud computing services through IoT devices.
- (2) Cloud server (S_j): S_j is deployed in the area to provide services to process and store the data of IoT devices. S_j is a semi-trusted entity; it can misbehave, but cannot collaborate with other participants. In addition, S_j has powerful storage capacity and computing power.
- (3) Control server (CS): CS is the control center for the cloud service provider to manage the S_j in the service area. CS is a semi-trusted entity; it can misbehave, but cannot

collaborate with other participants. Furthermore, CS is in charge of registration and authentication.

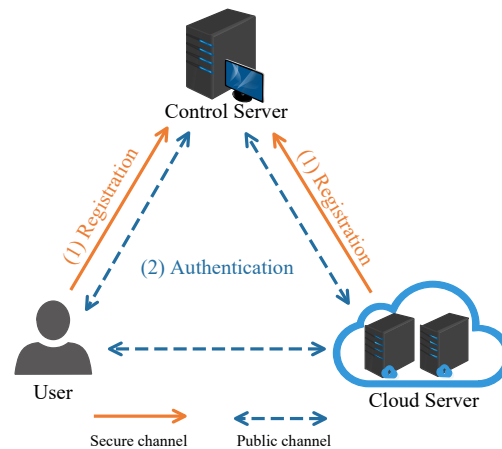


Figure 2. IoT-enabled cloud computing system model.

3.2. Concrete Protocol

The entire protocol consists of a registration phase, login phase, and a key agreement phase.

3.2.1. Registration Phase

In this section, we introduce the registration phase, which is divided into the U_i registration phase and S_j registration phase.

User (U_i) registration phase: When U_i wants to use the cloud computing service provided by the cloud service provider, U_i needs to register with CS. Moreover, to ensure the security and performance of confidential computation, a 1024-bit key length shared key was used in SGX. Figure 3 depicts the procedure for U_i registration; the processes involved are described below:

- (1) Initially, U_i selects ID_i , PW_i , and BIO_i . Next, U_i computes $(\sigma_i, \tau_i) = Gen(BIO_i)$ and $PID_i = h(ID_i \parallel PW_i \parallel \sigma_i)$ and transmits $\{PID_i\}$ to CS through a secure channel.
- (2) When CS receives the $\{PID_i\}$, it chooses a random number r_i and a shared key k_u , to compute $QU_i = h(PID_i \parallel k_u \parallel r_i)$. Next, CS stores $\{PID_i, r_i\}$ in its database, $\{PID_i, k_u\}$ in SGX, and $\{QU_i, r_i\}$ in the smart card (SC). Finally, CS transmits SC to U_i via the secure channel.
- (3) Upon receiving the SC from CS, U_i calculates $RU = r_i \oplus h(ID_i \parallel \sigma_i)$, $SU = h(PW_i \parallel r_i \parallel \sigma_i)$, $TU = QU_i \oplus h(r_i \parallel \sigma_i)$. Finally, U_i deletes $\{QU_i, r_i\}$ from SC and stores $\{RU, SU, TU\}$ in SC.

Cloud server (S_j) registration phase: S_j registers with CS before providing high-density computing services to the users. Figure 4 describes the S_j registration process, and the specific steps are described below:

- (1) First, S_j selects SID_j and a random number r_j . Next, S_j computes $PSID_j = h(SID_j \parallel r_j)$ and transmits $\{PSID_j, r_j\}$ to CS via the secure channel.
- (2) Upon receiving the $\{PSID_j, r_j\}$, CS first selects a shared key k_s and computes the value $QS_j = h(PSID_j \parallel k_s \parallel r_j)$. CS stores $\{PSID_j, r_j\}$ in its database and stores $\{PSID_j, k_s\}$ in SGX. Then, CS transmits $\{QS_j, k_s\}$ to S_j through the secure channel.
- (3) Upon receiving $\{QS_j, k_s\}$, S_j computes the value $RS = h(SID_j \parallel k_s) \oplus QS_j$. Finally, S_j stores $\{PSID_j, RS\}$ in the database and stores $\{k_s\}$ in SGX.

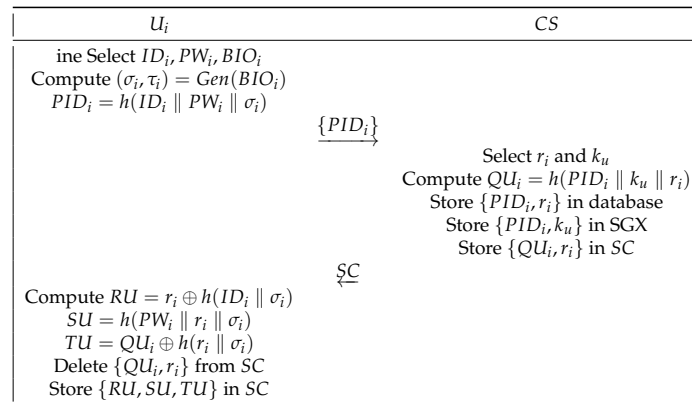


Figure 3. U_i 's registration phase.

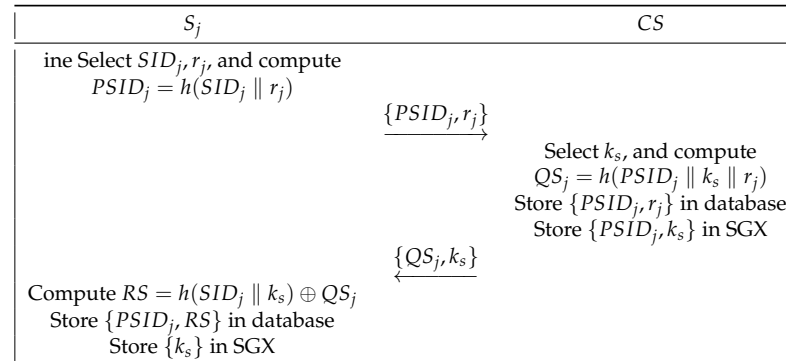


Figure 4. S_j 's registration phase.

3.2.2. Login and Key Agreement Phase

During this phase, U_i , S_j , and CS achieve mutual authentication and successfully establish an SK for future communication. Figure 5 depicts the full login and key agreement procedure; the exact steps are outlined below:

- First, U_i inputs ID_i , PW_i , and BIO_i . Subsequently, U_i computes $\sigma_i = Rep(BIO_i, \tau_i)$, $PID_i = h(ID_i \parallel PW_i \parallel \sigma_i)$, $r_i = RU \oplus h(ID_i \parallel \sigma_i)$, $SU^* = h(PW_i \parallel r_i \parallel \sigma_i)$ and checks $SU^* \stackrel{?}{=} SU$. If it is equal, U_i logs in successfully. Otherwise, the device rejects the login of U_i . Next, U_i computes $QU_i = TU \oplus h(r_i \parallel \sigma_i)$, generates a random number N_i , and chooses an SID_j , which is the private cloud server S_j 's identity. Then, U_i computes $W_1 = N_i \oplus h(r_i \parallel QU_i)$, $W_2 = SID_j \oplus h(N_i \parallel QU_i)$, and $V_1 = h(PID_i \parallel SID_j \parallel QU_i \parallel N_i)$. Finally, U_i retrieves the current timestamp T_1 and sends the message $M_1 = \{PID_i, W_1, W_2, V_1, T_1\}$ to S_j through the public channel.
- Upon receiving M_1 , S_j checks the freshness of T_1 . Then, S_j sends $\{SID_j, RS\}$ to the security interface of SGX and invokes the interface. The interface finds the k_s and uses it to calculate $QS_j = RS \oplus h(SID_j \parallel k_s)$. Then, SGX sends $\{QS_j\}$ to S_j through the interface. Next, S_j selects a random number N_j and computes $W_3 = h(SID_j \parallel QS_j) \oplus N_j$, $V_2 = h(PSID_j \parallel V_1 \parallel N_j \parallel QS_j)$. Finally, S_j retrieves the current timestamp T_2 and transmits the message $M_2 = \{PID_i, PSID_j, W_1, W_2, W_3, V_1, V_2, T_2\}$ to CS via the public channel.
- When CS receives the M_2 , it first verifies the freshness of T_2 . Then, CS retrieves $\{r_i\}$ in the database using PID_i and sends $\{PID_i, r_i\}$ to the security interface of SGX. Then, SGX invokes the interface and uses PID_i to match k_u , then it computes $QU_i = h(PID_i \parallel k_u \parallel r_i)$. Then, SGX sends $\{QU_i\}$ to CS through the interface. Next, CS computes $N_i = W_1 \oplus h(r_i \parallel QU_i)$, $SID_j = W_2 \oplus h(N_i \parallel QU_i)$, and $V_1^* = h(PID_i \parallel SID_j \parallel QU_i \parallel N_i)$. Then, CS checks whether $V_1^* \stackrel{?}{=} V_1$. If it holds, CS retrieves

$\{r_j\}$ in the database using $PSID_j$ and sends $\{PSID_j, r_j\}$ to the security interface of SGX. Then, SGX invokes the interface and uses $PSID_j$ to match k_s , then it computes $QS_j = h(PSID_j \parallel k_s \parallel r_j)$. Then, SGX sends $\{QS_j\}$ to CS through the interface. Next, CS computes $N_j = W_3 \oplus h(SID_j \parallel QS_j)$, $V_2^* = h(PSID_j \parallel V_1^* \parallel N_j \parallel QS_j)$ and checks the correctness of $V_2^* \stackrel{?}{=} V_2$. If it holds, CS authenticates the S_j . Otherwise, CS rejects the session. Further, CS selects a random number N_{cs} and calculates $W_4 = (N_i \parallel N_{cs}) \oplus h(QS_j \parallel N_j)$, $W_5 = (N_j \parallel N_{cs}) \oplus h(QU_i \parallel N_i)$, $SK = h(N_i \parallel N_j \parallel N_{cs})$, $V_3 = h(N_j \parallel QS_j \parallel SK)$, and $V_4 = h(N_i \parallel QU_i \parallel SK)$. Finally, CS generates the current timestamp T_3 and transmits the message $M_3 = \{W_4, W_5, V_3, V_4, T_3\}$ to S_j through the public channel.

- (4) Upon receiving the M_3 , S_j first checks the freshness of T_3 . Then, S_j computes $(N_i \parallel N_{cs}) = W_4 \oplus h(QS_j \parallel N_j)$, $SK = h(N_i \parallel N_j \parallel N_{cs})$, and $V_3^* = h(N_j \parallel QS_j \parallel SK)$. Then, S_j checks $V_3^* \stackrel{?}{=} V_3$. If $V_3^* = V_3$, S_j retrieves the current timestamp T_4 and sends the message $M_4 = \{W_5, V_4, T_4\}$ to U_i .
- (5) When U_i receives the M_4 , it first verifies the freshness of T_4 . Then, U_i calculates $(N_j \parallel N_{cs}) = W_5 \oplus h(QU_i \parallel N_i)$, $SK = h(N_i \parallel N_j \parallel N_{cs})$ and $V_4^* = h(N_i \parallel QU_i \parallel SK)$. Finally, U_i checks $V_4^* \stackrel{?}{=} V_4$. If it holds, U_i authenticates the S_j , and the entire authentication process is achieved.

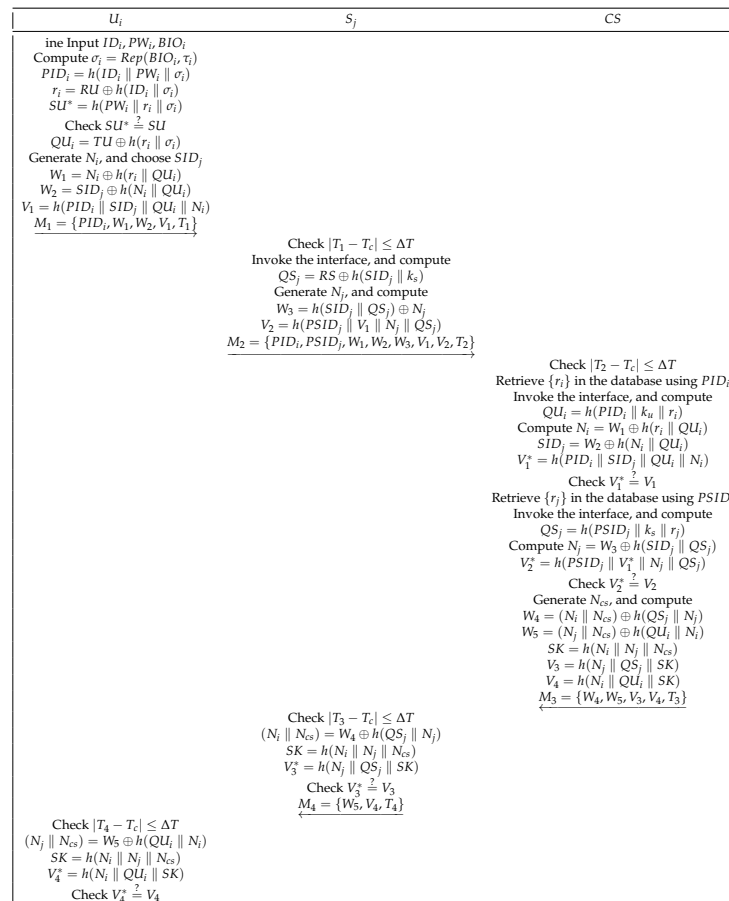


Figure 5. Login and key agreement phase.

4. Security Analysis

4.1. Formal Security Analysis

In this section, we utilize the ROR model [45,46] to formally demonstrate the security of the proposed protocol. By playing different games, we can calculate the probability of an attacker (\mathcal{A}) breaking the protocol (\mathcal{P}) under various conditions.

Attacker model: We assumed that \mathcal{A} has the following capabilities based on the well-known Dolev–Yao (DY) [47] and Canetti–Krawczyk (CK) [48] models:

- (1) \mathcal{A} has the ability to eavesdrop, intercept, tamper with, and replay messages sent between entities over a public channel.
- (2) \mathcal{A} can be a malicious insider in the CS or S_j and has access to the database’s information.
- (3) \mathcal{A} can steal the user’s SC and use power analysis [49] to extract information from the SC.
- (4) \mathcal{A} can affect the protocol’s security by obtaining random numbers.

4.1.1. Security Model

The proposed protocol contains three entities, U_i , S_j , and CS. Here, we used $\Pi_{U_i}^x$, $\Pi_{S_j}^y$, and Π_{CS}^z to represent the x -th user instance, y -th cloud server instance, and z -th control server instance, respectively. Suppose \mathcal{A} has the following query capabilities:

- (1) *Execute*(\mathcal{Z}): \mathcal{A} executing this query can intercept the messages M_i transmitted over the public channel between U_i , S_j , and CS, where $\mathcal{Z} = \{\Pi_{U_i}^x, \Pi_{S_j}^y, \Pi_{CS}^z\}$.
- (2) *Send*(\mathcal{Z}, M_i): When \mathcal{A} executes the query, \mathcal{A} sends an M_i to \mathcal{Z} , then receives a response from the \mathcal{Z} .
- (3) *Hash*(*string*): Through executing this query, \mathcal{A} can obtain the hash value of the *string* after the input *string*.
- (4) *Corrupt*(\mathcal{Z}): \mathcal{A} can obtain some private values by executing this query, such as the long-term private key, temporary value, and parameters in the SC.
- (5) *Test*(\mathcal{Z}): When \mathcal{A} performs this operation, he/she flips the coin c . If $c = 0$, \mathcal{A} can obtain a random value with the same length as SK. Otherwise, if $c = 1$, \mathcal{A} can obtain the correct SK.

Based on the attacker model and Section 4.1.1, the security of the proposed protocol is demonstrated using the theorem and proof below.

Theorem 1. *Within polynomial time complexity, the advantage that \mathcal{A} can break \mathcal{P} is $Adv_{\mathcal{A}}^{\mathcal{P}}(\xi) \leq q_{send}/2^{l-1} + 3q_{hash}^2/2^l + 2max\{C' \cdot q_{send}^s, q_{send}/2^l\}$. Here, q_{send} and q_{hash} denote the number of hashes and queries executed, respectively. l represents the bit length of the biometric, and C' and s' are constants.*

Proof. Seven rounds of games are played in the ROR model to verify the above theorem, denoted as GM_0 – GM_6 . Here, $Succ_{\mathcal{A}}^{GM_i}(\xi)$ refers to the event in which \mathcal{A} can win in GM_i . The process of \mathcal{A} simulating the queries is described in detail in Table 2. The steps of the proof are as follows.

GM_0 : GM_0 does not initiate query operations. Here, the game starts by flipping the coin c . Thus, the probability of GM_0 is

$$Adv_{\mathcal{A}}^{\mathcal{P}}(\xi) = |2Pr[Succ_{\mathcal{A}}^{GM_0}(\xi)] - 1|. \tag{1}$$

GM_1 : GM_1 adds the *Execute*(\mathcal{Z}) operation. At this point, \mathcal{A} intercepts messages $\{M_1, M_2, M_3, M_4\}$ transmitted over the public channel. Because the random numbers N_i , N_j , and N_{cs} are not available, \mathcal{A} using the *Test*(\mathcal{Z}) query cannot calculate SK. Therefore, the probability of GM_1 does not change and is

$$Pr[Succ_{\mathcal{A}}^{GM_1}(\xi)] = Pr[Succ_{\mathcal{A}}^{GM_0}(\xi)]. \tag{2}$$

GM_2 : GM_2 adds the *Send*(\mathcal{Z}) operation. GM_2 refers to Zipf’s law [50], and its probability is

$$|Pr[Succ_{\mathcal{A}}^{GM_2}(\xi)] - Pr[Succ_{\mathcal{A}}^{GM_1}(\xi)]| \leq q_{send}/2^l. \tag{3}$$

GM_3 : GM_3 introduces the $Hash(\mathcal{Z})$ operation while decreasing the $Send(\mathcal{Z})$ operation. Given the birthday paradox, it can be deduced that the probability of GM_3 is

$$|Pr[Succ_{\mathcal{A}}^{GM_3}(\xi)] - Pr[Succ_{\mathcal{A}}^{GM_2}(\xi)]| \leq q_{hash}^2/2^{l+1}. \quad (4)$$

GM_4 : In GM_4 , we assumed that \mathcal{A} uses $\Pi_{U_i}^x$, $\Pi_{S_j}^y$, or Π_{CS}^z to obtain a random number from the entities. Although \mathcal{A} can obtain a random number N_i chosen by U_i , the values of N_j and N_{cs} are unknown, and thus, \mathcal{A} cannot calculate SK . Similarly, assume that \mathcal{A} has access to N_j or N_{cs} , and SK is also not computed. Therefore, the probability of GM_4 is

$$|Pr[Succ_{\mathcal{A}}^{GM_4}(\xi)] - Pr[Succ_{\mathcal{A}}^{GM_3}(\xi)]| \leq q_{hash}^2/2^{l+1}. \quad (5)$$

GM_5 : In GM_5 , suppose that \mathcal{A} can obtain $\{RU, SU, TU\}$ in SC by executing a $Corrupt(\mathcal{Z})$ query. Subsequently, \mathcal{A} utilizes these parameters to perform an offline password guessing attack to determine the user's correct password. However, \mathcal{A} cannot obtain r_i and σ_i , and thus, \mathcal{A} cannot guess PW_i . The probability that \mathcal{A} guesses the biometric of l bits is $1/2^l$. Based on Zipf's law [50], the probability of \mathcal{A} guessing the correct PW_i when $q_{send} \leq 10^6$ is greater than $1/2$. Therefore, the probability of GM_5 is

$$|Pr[Succ_{\mathcal{A}}^{GM_5}(\xi)] - Pr[Succ_{\mathcal{A}}^{GM_4}(\xi)]| \leq \max\{C' \cdot q_{send}^{s'}, q_{send}/2^l\} \quad (6)$$

GM_6 : GM_6 is designed to demonstrate that the proposed protocol can withstand impersonation attacks. Assuming that \mathcal{A} can successfully obtain SK by using the $h(N_i \parallel N_j \parallel N_{cs})$ query, the game terminates. Therefore, the probability of GM_6 is

$$|Pr[Succ_{\mathcal{A}}^{GM_6}(\xi)] - Pr[Succ_{\mathcal{A}}^{GM_5}(\xi)]| \leq q_{hash}^2/2^{l+1}. \quad (7)$$

Both the success and failure probabilities for GM_6 are equal to $1/2$. Consequently, the probability that \mathcal{A} calculates SK is

$$Pr[Succ_{\mathcal{A}}^{GM_6}(\xi)] = 1/2. \quad (8)$$

Given the probabilities of GM_0 to GM_6 , we obtain

$$\begin{aligned} 1/2 Adv_{\mathcal{A}}^{\mathcal{P}}(\xi) &= |Pr[Succ_{\mathcal{A}}^{GM_0}(\xi)] - 1/2| \\ &= |Pr[Succ_{\mathcal{A}}^{GM_0}(\xi)] - Pr[Succ_{\mathcal{A}}^{GM_6}(\xi)]| \\ &= |Pr[Succ_{\mathcal{A}}^{GM_1}(\xi)] - Pr[Succ_{\mathcal{A}}^{GM_6}(\xi)]| \\ &\leq \sum_{i=0}^5 |Pr[Succ_{\mathcal{A}}^{GM_{i+1}}(\xi)] - Pr[Succ_{\mathcal{A}}^{GM_i}(\xi)]| \\ &= q_{send}/2^l + 3q_{hash}^2/2^{l+1} + \max\{C' \cdot q_{send}^{s'}, q_{send}/2^l\} \end{aligned} \quad (9)$$

Finally, we obtained the following:

$$Adv_{\mathcal{A}}^{\mathcal{P}}(\xi) \leq q_{send}/2^{l-1} + 3q_{hash}^2/2^l + 2\max\{C' \cdot q_{send}^{s'}, q_{send}/2^l\}. \quad (10)$$

□

4.2. Informal Security Analysis

4.2.1. Replay Attacks

In our protocol, each message transmitted on a public channel contains timestamp T_i . Only within a valid timestamp can the receiver pass the check and continue with the subsequent computation. Here, take message $M_1 = \{PID_i, W_1, W_2, V_1, T_1\}$ as an example. Suppose \mathcal{A} repeatedly sends M_1 to S_j . After receiving M_1 , S_j checks the validity of its timestamp by computing $|T_1 - T_c| \leq \Delta T$. Because T_1 in M_1 exceeds a limited time,

S_j terminates this session process. Therefore, the replay attacks were invalid for our proposed protocol.

4.2.2. Privileged Insider Attacks

Case 1: Suppose \mathcal{A} can steal data $\{PID_i, r_i\}$ and $\{PSID_j, r_j\}$ from CS and attempt to compute N_i and N_j using messages intercepted on the public channel, where $N_i = W_1 \oplus h(r_i \parallel QU_i)$ and $N_j = W_3 \oplus h(SID_j \parallel QS_j)$. However, \mathcal{A} cannot obtain the values QU_i , QS_j , and SID_j , and thus, N_i and N_j cannot be calculated. N_{cs} is a number randomly chosen by the CS and is different in each session such that \mathcal{A} does not obtain N_{cs} . Thus, \mathcal{A} does not obtain session key SK , where $SK = h(N_i \parallel N_j \parallel N_{cs})$.

Case 2: Assume that \mathcal{A} has access to the data $\{PID_i, RS\}$ in S_j and attempts to calculate the value N_j , where $N_j = W_3 \oplus h(SID_j \parallel QS_j)$. However, \mathcal{A} cannot obtain the values QS_j or SID_j ; thus, \mathcal{A} cannot compute N_j . Similarly, $(N_i \parallel N_{cs})$ cannot be computed by $(N_i \parallel N_{cs}) = W_5 \oplus h(QS_j \parallel N_j)$. Therefore, \mathcal{A} cannot successfully calculate the SK .

Table 2. Simulation of the *Send*, *Execute*, *Hash*, *Corrupt*, and *Test* queries.

Query	Description
<i>Send</i> (\mathcal{Z}, M_i)	For a query <i>Send</i> ($\Pi_{U_i}^x, \text{start}$), suppose $\Pi_{U_i}^x$ selects N_i, SID_j , and T_1 , and compute $W_1 = N_i \oplus h(r_i \parallel QU_i)$, $W_2 = SID_j \oplus h(N_i \parallel QU_i)$, $V_1 = h(PID_i \parallel SID_j \parallel QU_i \parallel N_i)$ in a normal state. Then, the query returns the output $M_1 = \{PID_i, W_1, W_2, V_1, T_1\}$.
	On a query <i>Send</i> ($\Pi_{S_j}^y, (PID_i, W_1, W_2, V_1, T_1)$), suppose $\Pi_{S_j}^y$ is in a normal state and performs the following operations: computes QS_j , and selects N_j, T_2 ; then, computes W_3, V_2 . The query is answered by $M_2 = \{PID_i, PSID_j, W_1, W_2, W_3, V_1, V_2, T_2\}$.
	On a query <i>Send</i> ($\Pi_{CS}^z, (PID_i, PSID_j, W_1, W_2, W_3, V_1, V_2, T_2)$), upon receiving the send query message $(PID_i, PSID_j, W_1, W_2, W_3, V_1, V_2, T_2)$, Π_{CS}^z computes QU_i, N_i, SID_j and checks V_1 . If it is equal, then it computes QS_j, N_j and checks V_2 . If it is equal, it generates N_{cs}, T_3 and computes W_4, W_5, SK, V_3, V_4 . Then, Π_{CS}^z returns the output $M_3 = \{W_4, W_5, V_3, V_4, T_3\}$.
	For a query <i>Send</i> ($\Pi_{S_j}^y, (W_4, W_5, V_3, V_4, T_3)$), suppose $\Pi_{S_j}^y$ computes $(N_i \parallel N_{cs})$, SK and checks V_3 in a normal state. If V_3 holds, it selects T_4 . Then, the query returns the output $M_4 = \{W_5, V_4, T_4\}$.
	For a query <i>Send</i> ($\Pi_{U_i}^x, W_5, V_4, T_4$), upon receiving the send query message (W_5, V_4, T_4) , $\Pi_{U_i}^x$ computes $(N_j \parallel N_{cs})$, SK and checks V_4 . If V_4 is incorrect, the query process is terminated. Finally, $\Pi_{U_i}^x$ accepts and terminates.
<i>Execute</i> (\mathcal{Z})	We proceed with the send query for the <i>Execute</i> (\mathcal{Z}) query as follows. $(PID_i, W_1, W_2, V_1, T_1) \leftarrow \text{Send}(\Pi_{U_i}^x, \text{start})$, $(PID_i, PSID_j, W_1, W_2, W_3, V_1, V_2, T_2) \leftarrow \text{Send}(\Pi_{S_j}^y, (PID_i, W_1, W_2, V_1, T_1))$, $(W_4, W_5, V_3, V_4, T_3) \leftarrow \text{Send}(\Pi_{CS}^z, (PID_i, PSID_j, W_1, W_2, W_3, V_1, V_2, T_2))$, $(W_4, W_5, V_3, V_4, T_3) \leftarrow \text{Send}(\Pi_{S_j}^y, (W_4, W_5, V_3, V_4, T_3))$. The query is answered with transcripts $(PID_i, W_1, W_2, V_1, T_1), (PID_i, PSID_j, W_1, W_2, W_3, V_1, V_2, T_2), (W_4, W_5, V_3, V_4, T_3)$, and (W_5, V_4, T_4) .
<i>Hash</i> (<i>string</i>)	For a <i>Hash</i> (<i>string</i>) query, if the query is executed and a record (<i>string</i> , <i>s</i>) appears in the query, $s = \text{hash}(\text{string})$ is returned. Otherwise, an element <i>s</i> is selected, and (<i>string</i> , <i>s</i>) is added to the list and returns <i>s</i> .
<i>Corrupt</i> (\mathcal{Z})	For a <i>Corrupt</i> ($\Pi_{U_i}^x$), if the $(\Pi_{U_i}^x)$ is accepted, executing the query returns the parameters $\{RU, SU, TU\}$ in SC.
<i>Test</i> (\mathcal{Z})	The coin <i>c</i> is flipped; if $c = 0$, return a random value with the same length as SK ; otherwise, return the correct SK .

An analysis of the above two cases leads to the conclusion that privileged insider attacks are not valid for our proposed protocol.

4.2.3. Man-in-the-Middle Attacks

We assumed that \mathcal{A} can intercept the message $M_2 = \{PID_i, PSID_j, W_1, W_2, W_3, V_1, V_2, T_2\}$ sent from S_j to CS and attempt to modify the authentication values V_1 and V_2 , where

$V_1 = h(PID_i \parallel SID_j \parallel QU_i \parallel N_i)$ and $V_2 = h(PSID_j \parallel V_1 \parallel N_j \parallel QS_j)$. However, \mathcal{A} cannot obtain the values SID_j , QU_i , QS_j , N_i , and N_j ; \mathcal{A} cannot calculate V_1 and V_2 . Therefore, after the CS receives the message sent by \mathcal{A} , it is not allowed to pass this authentication. Similarly, \mathcal{A} eavesdrops the message $M_4 = \{W_5, V_4, T_4\}$ sent from S_j to U_i and attempts to change the authentication value V_3 , where $V_3 = h(N_j \parallel QS_j \parallel SK)$. Because \mathcal{A} cannot obtain the value N_j and cannot compute the SK , it cannot calculate the correct authentication value V_3 . It can be observed that the request sent by \mathcal{A} cannot be authenticated by U_i . Consequently, our protocol is immune to man-in-the-middle attacks.

4.2.4. User Impersonation Attacks

Suppose that \mathcal{A} can intercept message $M_1 = \{PID_i, W_1, W_2, V_1, T_1\}$. If \mathcal{A} imitates a legitimate U_i to communicate with the CS, \mathcal{A} must construct the correct authentication value V_1 , where $V_1 = h(PID_i \parallel SID_j \parallel QU_i \parallel N_i)$. However, \mathcal{A} cannot obtain SID_j , QU_i , and N_i ; therefore, the correct authentication value V_1 cannot be calculated. Therefore, the message delivered by \mathcal{A} cannot pass CS authentication. The proposed protocol can withstand user impersonation attacks.

4.2.5. Cloud Server Impersonation Attacks

Assume that \mathcal{A} wants to impersonate a legitimate S_j to establish communication with the CS. \mathcal{A} must intercept the message $M_1 = \{PID_i, W_1, W_2, V_1, T_1\}$ on the public channel and construct the correct authentication value V_2 , where $V_2 = h(PSID_j \parallel V_1 \parallel N_j \parallel QS_j)$. However, \mathcal{A} cannot obtain the values N_j and QS_j and cannot calculate the valid parameter V_1 ; thus, \mathcal{A} cannot compute the correct V_2 . Thus, the request sent by \mathcal{A} cannot be authenticated by CS. Our protocol can resist cloud server impersonation attacks.

4.2.6. Anonymity and Untraceability

In our protocol, the real identities of U_i and S_j are hidden using random numbers and a hash function. Only pseudo-identities PID_i or $PSID_j$ are used in the authentication process to ensure the anonymity of U_i and S_j . In addition, attackers cannot trace U_i or S_j through an intercepted message because both entities use pseudo-identities when communicating with CS. Furthermore, the random number in the message was different for each session, ensuring that each entity was untraceable. Thus, the proposed protocol guarantees anonymity and untraceability.

4.3. ProVerif

ProVerif is a simulation tool proposed by Blanchet [51] for the automatic verification of encryption protocols. ProVerif can handle basic encryption operations based on the DY model [47], such as hashing, XOR, and fuzzy extraction. In this study, to demonstrate the security of our protocol, we simulated the entire registration and authentication procedure for U_i , S_j , and CS using the ProVerif tool.

Figure 6 illustrates the ProVerif code symbols and operational definitions. Figure 7 shows the query operations and events. There are six events involved in the protocol, namely `IoTDeviceStarted()`, `IoTDeviceAuthed()`, `ControlServerAcIoTDevice()`, `ControlServerAcCouldServer()`, `CouldServerAcControlServer()`, and `IoTDeviceAcControlServer()`, which represent that U_i begins the authentication process, U_i completes the authentication, CS completes the authentication of the U_i , CS completes the authentication of the S_j , S_j completes the authentication of the CS, and U_i completes the authentication of the CS.

```

(* channel*)
free ch:channel. (* public channel *)
free sch:channel [private]. (* secure channel, used for registering *)
(* shared keys *)
free SKi:bitstring [private].
free SKj:bitstring [private].
free SKc:bitstring [private].
(* constants *)
free ku:bitstring [private]. (* the shared key between Ui and CS *)
free ks:bitstring [private]. (* the shared key between Sj and CS *)
(* functions & reductions & equations *)
fun h(bitstring) :bitstring. (* hash function *)
fun mult(bitstring,bitstring) :bitstring. (* scalar multiplication operation *)
fun add(bitstring,bitstring):bitstring. (* Addition operation *)
fun sub(bitstring,bitstring):bitstring. (* Subtraction operation *)
fun mod(bitstring,bitstring):bitstring. (* modulus operation *)
fun con(bitstring,bitstring):bitstring. (* concatenation operation *)
reduc forall m:bitstring, n:bitstring; getmess(con(m,n))=m.
fun xor(bitstring,bitstring):bitstring. (* XOR operation *)
equation forall m:bitstring, n:bitstring; xor(xor(m,n),n)=m.
fun senc(bitstring,bitstring):bitstring. (* symmetric encryption *)
reduc forall m:bitstring, key:bitstring; sdec(senc(m,key),key)=m.
fun Gen(bitstring):bitstring. (*Generator operation *)
fun Rep(bitstring,bitstring):bitstring.

```

Figure 6. Definitions.

```

query attacker(SKi).
query attacker(SKj).
query attacker(SKc).
query inj-event(IoTDeviceAuthed()) ==> inj-event(IoTDeviceStarted()).
query inj-event(ControlServerAcCouldServer()) ==> inj-event(ControlServerAcIoTDevice()).
query inj-event(CouldServerAcControlServer()) ==> inj-event(ControlServerAcCouldServer()).
query inj-event(IoTDeviceAcControlServer()) ==> inj-event(CouldServerAcControlServer()).

event IoTDeviceStarted().
event IoTDeviceAuthed().
event ControlServerAcIoTDevice().
event ControlServerAcCouldServer().
event CouldServerAcControlServer().
event IoTDeviceAcControlServer().

```

Figure 7. The queries and events.

Figure 8 shows the U_i execution process, S_j execution process, and CS verification process. Here, we take the process of U_i as an example to explain. “out(sch,(PIDi))” is the statement that U_i initiates registration to CS, and “in(sch,xQUi:bitstring,xri:bitstring)” represents that U_i receives messages from the CS during the registration phase, which means that the registration phase is over. “out(ch,(PIDi,W1,W2,V1,T1))” means U_i sends a authentication request to the CS. “in(ch,(xW5:bitstring,xV4:bitstring,xT4:bitstring))” represents that U_i receives messages from the CS. Finally, in Figure 9, we can observe that \mathcal{A} cannot calculate SK between U_i , S_j , and CS, which means that the proposed protocol is secure.

```

(* ----- Ui's process ----- *)
et Process Ui =
new IDi:bitstring; new PWi:bitstring;
new BIOi:bitstring;
let (a:bitstring,b:bitstring)=Gen(BIOi) in
let PIDi=h(con(con(IDi,PWi),a)) in
out(sch,(PIDi));
in(sch,(xQUi:bitstring,xri:bitstring));
let RU=h(con(con(PIDi,PWi),a)) in
let SU=xor(xri,h(con(IDi,a))) in
let TU=xor(xQUi,h(con(xri,a))) in
!( event IoTDeviceStarted();
let a=Rep(BIOi,b) in
let PID=h(con(con(IDi,PWi),a)) in
let RU'=h(con(con(PIDi,PWi),a)) in
if RU'=RU
then let ri=xor(SU,h(con(IDi,a))) in
let xQUi=xor(TU,h(con(ri,a))) in
new Ni:bitstring; new SIDj:bitstring;
new T1:bitstring;
let W1=xor(Ni,h(con(ri,xQUi))) in
let W2=xor(SIDj,h(con(Ni,xQUi))) in
let V1=h(con(con(con(PIDi,SIDj),xQUi),Ni)) in
out(ch,(PIDi,W1,W2,V1,T1));
event IoTDeviceAuthed();
in(ch,(xW5:bitstring,xV4:bitstring,xT4:bitstring));
new Nj:bitstring; new Ncs:bitstring;
let x1=con(Nj,Ncs) in
let x1=xor(xW5,h(con(xQUi,Ni))) in
let SKi=h(con(con(Ni,Nj),Ncs)) in
let V4'=h(con(con(Ni,xQUi),SKi)) in
if V4'=xV4
then event IoTDeviceAcControlServer();
0).

(* ----- Sj process ----- *)
let ProcessSj=
new SIDj:bitstring; new rj:bitstring;
let PSIDj=h(con(SIDj,rj)) in
out (sch,(PSIDj,rj));
in(sch,(yQSj:bitstring,yks:bitstring));
let RS=xor(h(con(SIDj,yks)),yQSj) in
!(in(ch,(yPIDi:bitstring,yW1:bitstring,
yW2:bitstring,yV1:bitstring,yT1:bitstring));
let yQSj=xor(RS,h(con(SIDj,yks))) in
new Nj:bitstring; new T2:bitstring;
let W3=xor(h(con(SIDj,yQSj)),Nj) in
let V2=h(con(con(con(PSIDj,yV1),Nj),yQSj)) in
out(ch,(yPIDi,PSIDj,yW1,yW2,W3,yV1,V2,T2));
in(ch,(yW4:bitstring,yW5:bitstring,yV3:bitstring,
yV4:bitstring,yT3:bitstring));

new Ni:bitstring; new Ncs:bitstring;
let y1=con(Ni,Ncs) in
let y1=xor(yW4,h(con(yQSj,Nj))) in
let SKj=h(con(con(Ni,Nj),Ncs)) in
let V3'=h(con(con(Nj,yQSj),SKj)) in
if V3'=yV3
then event CouldServerAcControlServer();
new T4:bitstring;
out(ch,(yW5,yV4,T4));
0).

(* ----- CS process ----- *)
let UiReg =
new ri:bitstring;
in(sch,(zPIDi:bitstring,ri:bitstring));
let QUI=h(con(con(zPIDi,ku),ri)) in
out(sch,(QUI)); 0.
(* -----Ui registration ----- *)
let SjReg = in (sch,(zPSIDj:bitstring,zrj:bitstring));
let QSj=h(con(con(zPSIDj,ks),zrj)) in
out(sch,(QSj,ks));
0.
(* -----Sj registration ----- *)
let CSAuth = in(ch,(zPIDi:bitstring,zPSIDj:bitstring,
zW1:bitstring,zW2:bitstring,zW3:bitstring,
zV1:bitstring,zV2:bitstring,zT2:bitstring));
new ri:bitstring;
let QUI=h(con(con(zPIDi,ku),ri)) in
let Ni=xor(zW1,h(con(ri,QUI))) in
let SIDj=xor(zW2,h(con(Ni,QUI))) in
let V1'=h(con(con(con(zPIDi,SIDj),QUI),Ni)) in
if V1'=zV1
then event ControlServerAcIoTDevice();
(*-----CS verifies Ui-----*)
new rj:bitstring;
let QSj=h(con(con(zPSIDj,ks),rj)) in
let Nj=xor(zW3,h(con(SIDj,QSj))) in
let V2'=h(con(con(con(zPSIDj,zV1),Nj),QSj)) in
if V2'=zV2
then event ControlServerAcCouldServer();
(*-----CS verifies Sj-----*)
new Ncs:bitstring; new T3:bitstring;
let W4=xor(con(Ni,Ncs),h(con(QSj,Nj))) in
let W5=xor(con(Nj,Ncs),h(con(QUI,Ni))) in
let SKc=h(con(con(Ni,Nj),Ncs)) in
let V3=h(con(con(Nj,QSj),SKc)) in
let V4=h(con(con(Ni,QUI),SKc)) in
out(ch,(W4,W5,V3,V4,T3)); 0.
let Process CS = UiReg | SjReg | CSAuth.
(* ----- Main ----- *) process
(!ProcessUi | !ProcessSj | !ProcessCS)

```

Figure 8. Execution process of U_i , S_j , and CS.

```

Verification summary:
Query not attacker(SKi[]) is true.
Query not attacker(SKj[]) is true.
Query not attacker(SKc[]) is true.
Query inj-event(IoTDeviceAuthed) ==> inj-event(IoTDeviceStarted) is true.
Query inj-event(ControlServerAcCouldServer) ==> inj-event(ControlServerAcIoTDevice) is true.
Query inj-event(CouldServerAcControlServer) ==> inj-event(ControlServerAcCouldServer) is true.
Query inj-event(IoTDeviceAcControlServer) ==> inj-event(CouldServerAcControlServer) is true.

```

Figure 9. Verification result.

5. Security and Performance Comparisons

In this section, the security and performance of our proposed protocol are compared with that of existing protocols [13,23,24,40].

5.1. Security Comparisons

In this section, the security of our proposed protocol is compared with that of current protocols. ✓ indicates that the protocol can withstand an attack, whereas × indicates that the protocol cannot. The primary attacks include: S1, mutual authentication; S2, session key disclosure attacks; S3, forward secrecy; S4, user anonymity; S5, temporary value disclosure attacks; S6, impersonation attacks; S7, replay attacks; S8, offline password guessing attacks. Table 3 shows that Zhou et al.'s protocol [23] does not guarantee forward secrecy and is not resistant to temporary value disclosure attacks and impersonation attacks, and Martinez-Pelaez et al.'s protocol [24] does not guarantee mutual authentication and anonymity and is not resistant to session key disclosure attacks, impersonation attacks, and replay attacks. The protocols of Huang et al. [13] and Wu et al. [40] have the same security as our proposed protocol, which can resist known attacks.

Table 3. Comparisons of security.

Attack Methods	Huang et al. [13]	Zhou et al. [23]	Martinez-Pelaez et al. [24]	Wu et al. [40]	Ours
S1	✓	✓	× [39]	✓	✓
S2	✓	✓	× [39]	✓	✓
S3	✓	× [38]	✓	✓	✓
S4	✓	✓	× [39]	✓	✓
S5	✓	× [38]	✓	✓	✓
S6	✓	× [38]	× [39]	✓	✓
S7	✓	✓	× [39]	✓	✓

5.2. Performance Comparisons

The performance comparison includes both the computational and communication costs. In comparing the computational cost, the cost of \oplus and \parallel is too small to be negligible, and the hash function and fuzzy extractor both execute at the same time [52]; thus, we used the time to execute a hash function to represent the execution time of the fuzzy extractor. Wang et al. [53] showed that the runtime of the system using SGX increases by only 20 us, which shows that the computational volume of SGX is relatively low. Therefore, we ignored the computational cost of SGX in the computational cost comparison. In addition, we conducted simulation experiments to estimate the computational cost of the protocols. Here, we used an MI 8 to simulate U_i , a Lenovo desktop computer to simulate S_j , and a Lenovo laptop to simulate CS. The phone used a packaged algorithmic time application, and the computer development software was IntelliJ idea version 2020.3. The equipment configuration and operation runtime are listed in Table 4, in which the execution time is obtained by running 10 times and averaging. Here, we compared only the protocol's login and key agreement phases. As can be observed from Table 5 and Figure 10, since Martinez-Pelaez et al. [24] used symmetric key encryption/decryption in their protocol, the computational cost of their protocol was the highest among all protocols. In addition, the computational cost of our protocol was slightly higher than that of Huang et al.'s protocol [13], but our protocol had a lower computational cost compared to Zhou et al. [23] and Wu et al. [40].

Table 4. The configuration of the equipment and operation times.

	MI 8	Lenovo Desktop Computer	Lenovo Laptop
Operating System	Android system	Windows 10	Windows 10
CPU	Qualcomm Snapdragon 845	Intel(R) Core(TM) i5-9500 CPU @ 3.00 GHz	Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz
Running memory	6 GB	16 GB	8 GB
Symmetric key encryption/decryption	0.2554 ms	0.1385 ms	0.1874 ms
Hash function	0.0045 ms	0.0026 ms	0.0035 ms

Table 5. Computational cost comparison.

Protocols	U_i (ms)	S_j (ms)	CS (ms)	Total (ms)
Huang et al. [13]	$8T_h \approx 0.036$	$4T_h \approx 0.010$	$10T_h \approx 0.035$	0.081
Zhou et al. [23]	$10T_h \approx 0.045$	$7T_h \approx 0.018$	$19T_h \approx 0.067$	0.130
Martinez-Pelaez et al. [24]	$3T_s + 7T_h \approx 0.798$	$3T_s + 6T_h \approx 0.431$	$2T_s + 26T_h \approx 0.466$	1.695
Wu et al. [40]	$12T_h \approx 0.054$	$8T_h \approx 0.021$	$19T_h \approx 0.067$	0.142
Ours	$T_f + 10T_h \approx 0.049$	$6T_h \approx 0.016$	$12T_h \approx 0.042$	0.107

Here, T_s denotes the symmetric key encryption/decryption operation’s execution time, T_f denotes the fuzzy extraction function’s execution time, and T_h denotes the hash operation’s execution time.

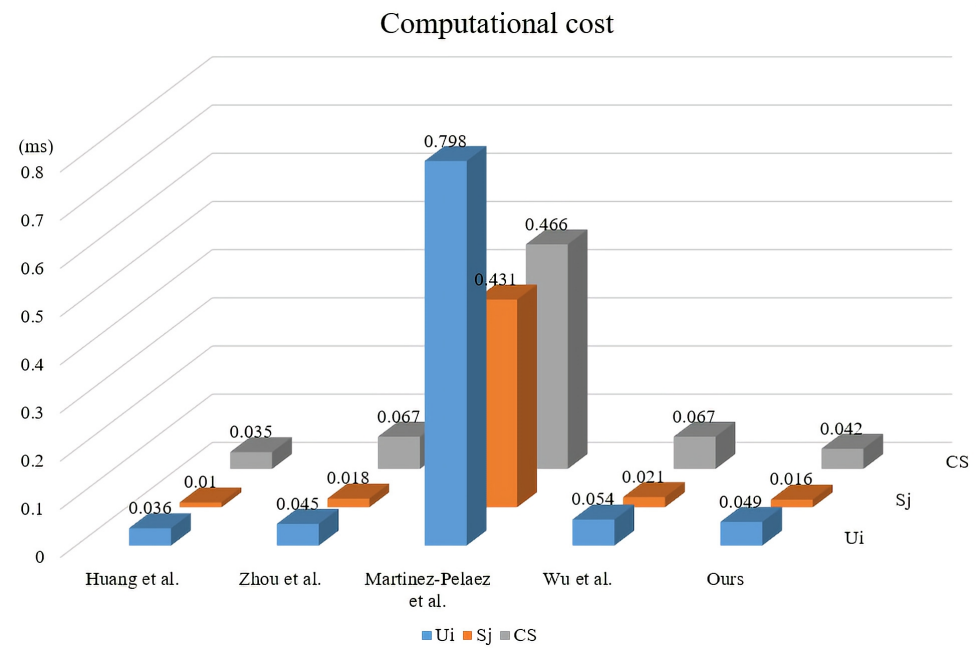


Figure 10. Computational cost comparison [13,23,24,40].

In comparing the communication costs, we assumed that the length of identity $|ID|$, timestamp $|T|$, one-way hash function $|H|$, random number $|Z_p^*|$, and symmetric key encryption/decryption $|E|$ were 160, 32, 256, 128, and 256 bits, respectively. In our proposed protocol, the messages transmitted on the public channel included $M_1 = \{PID_i, W_1, W_2, V_1, T_1\}$, $M_2 = \{PID_i, PSID_j, W_1, W_2, W_3, V_1, V_2, T_2\}$, $M_3 = \{W_4, W_5, V_3, V_4, T_3\}$, and $M_4 = \{W_5, V_4, T_4\}$. Thus, it can be calculated that the communication cost of our protocol is $8|Z_p^*| + 4|T| + 3|ID| + 6|H| = 3168$ bits; the communication costs of Huang et al.’s protocol [13] is $6|Z_p^*| + 3|T| + 3|ID| + 8|H| = 3392$ bits; Zhou et al.’s protocol [23] is $15|Z_p^*| + 3|ID| + 6|H| = 3936$ bits; Martinez-Pelaez et al.’s protocol [24] is $14|Z_p^*| + 3|T| + 3|ID| + 6|E| + |H| = 4160$ bits; Wu et al.’s protocol [40] is $15|Z_p^*| + 3|ID| + 7|H| = 4192$ bits. Table 6 shows that the communication cost of our protocol was lower than that of the protocols of Huang et al. [13], Zhou et al. [23], Martinez-Pelaez et al. [24], and Wu et al. [40]. Figure 11 shows visually the communication costs per protocol.

Table 6. Communication cost comparison.

Protocols	Rounds	Communication Cost
Huang et al. [13]	4	3392 bits(+7.07%)
Zhou et al. [23]	4	3936 bits(+24.24%)
Martinez-Pelaez et al. [24]	6	4160 bits(+31.31%)
Wu et al. [40]	5	4192 bits(+32.32%)
Ours	4	3168 bits

According to the above comparison results, it can be concluded that there are some vulnerabilities to attacks in the protocols of Zhou et al. [23] and Martinez-Pelaez et al. [24], while other protocols have the same security as our proposed protocol, which can resist known attacks. Although the computational cost of our proposed protocol was slightly higher than that of Huang et al. [13] by 0.026 ms, the communication cost was lower than that of Huang et al. [13] by 224 bits (7.07%).

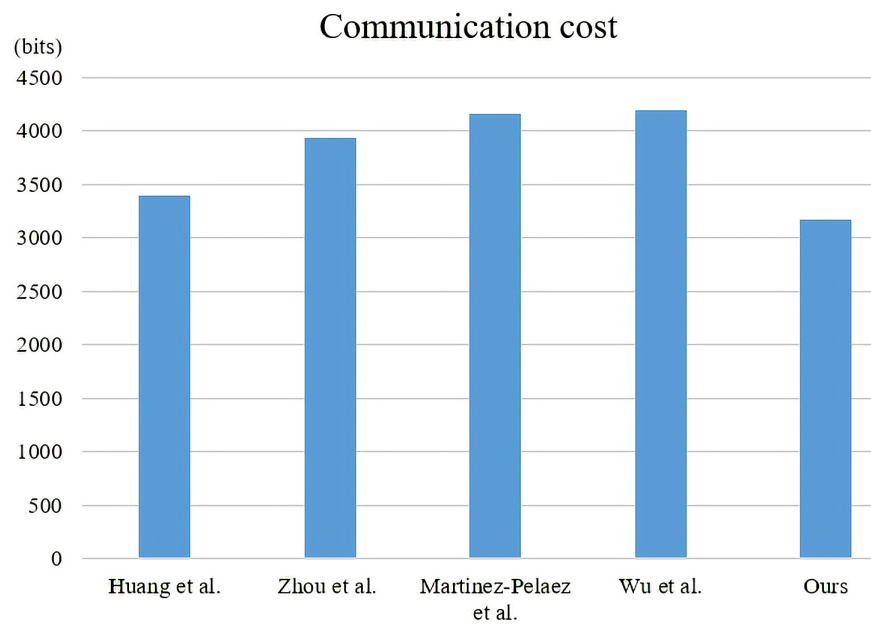


Figure 11. Communication cost comparison [13,23,24,40].

6. Conclusions

In this paper, we first described the necessity of combining the IoT with cloud computing. Simultaneously, we reviewed some AKA protocols designed in the IoT-enabled cloud computing environments and found that there are still some security problems. To address those problems, we proposed an SGX-based lightweight AKA protocol for IoT-enabled cloud computing. Our goal was to ensure data privacy and sustainable communication between the entities. In addition, the security of the proposed protocol was examined using the ROR model, the ProVerif tool, and informal security analysis. According to the comparison of the results of the security and performance, our proposed protocol can ensure sufficient security and reduce the communication cost by 7.07% compared with the best one among the current protocols. The limitation is that the computational cost is slightly higher, but it is acceptable in practical application. Therefore, we will continue to enhance the security and performance of protocols in the IoT-enabled cloud computing environments in future work.

Author Contributions: Conceptualization, T.-Y.W.; methodology, T.-Y.W. and L.W.; software, X.G.; formal analysis, Y.-C.C.; investigation, S.-C.C.; writing—original draft preparation, T.-Y.W., L.W., X.G., Y.-C.C. and S.-C.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: This study did not involve humans.

Data Availability Statement: The data are included in the article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
SGX	Software guard extensions
ROR	Real-or-random
AKA	Authentication and key agreement

References

- Goudos, S.K.; Dallas, P.I.; Chatziefthymiou, S.; Kyriazakos, S. A survey of IoT key enabling and future technologies: 5G, mobile IoT, semantic web and applications. *Wirel. Pers. Commun.* **2017**, *97*, 1645–1675. [CrossRef]
- Xue, X.; Wu, X.; Jiang, C.; Mao, G.; Zhu, H. Integrating sensor ontologies with global and local alignment extractions. *Wirel. Commun. Mob. Comput.* **2021**, *2021*, 6625184. [CrossRef]
- Huang, X.; Xiong, H.; Chen, J.; Yang, M. Efficient revocable storage attribute-based encryption with arithmetic span programs in cloud-assisted internet of things. *IEEE Trans. Cloud Comput.* **2021**. [CrossRef]
- Shen, S.; Yang, Y.; Liu, X. Toward data privacy preservation with ciphertext update and key rotation for IoT. *Concurr. Comput. Pract. Exp.* **2021**, e6729. [CrossRef]
- Meng, Z.; Pan, J.S.; Tseng, K.K. PaDE: An enhanced Differential Evolution algorithm with novel control parameter adaptation schemes for numerical optimization. *Knowl. Based Syst.* **2019**, *168*, 80–99. [CrossRef]
- Xue, X.; Zhang, J. Matching large-scale biomedical ontologies with central concept based partitioning algorithm and adaptive compact evolutionary algorithm. *Appl. Soft Comput.* **2021**, *106*, 107343. [CrossRef]
- Liu, G.; Zhu, Y.; Xu, S.; Chen, Y.C.; Tang, H. PSO-based power-driven X-routing algorithm in semiconductor design for predictive intelligence of IoT applications. *Appl. Soft Comput.* **2022**, *114*, 108114. [CrossRef]
- He, Z.; Yu, C. Clustering stability-based evolutionary k-means. *Soft Comput.* **2019**, *23*, 305–321. [CrossRef]
- Chaudhry, S.A. Combating identity de-synchronization: An improved lightweight symmetric key based authentication scheme for IoV. *J. Netw. Intell.* **2021**, *6*, 656–667.
- Xiong, H.; Chen, J.; Mei, Q.; Zhao, Y. Conditional privacy-preserving authentication protocol with dynamic membership updating for VANETs. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 2089–2104. [CrossRef]
- Wu, T.; Guo, X.; Chen, Y.; Kumari, S.; Chen, C. Amassing the security: An enhanced authentication protocol for drone communications over 5G networks. *Drones* **2021**, *6*, 10. [CrossRef]
- Luo, Y.; Zheng, W.M.; Chen, Y.C. An anonymous authentication and key exchange protocol in smart grid. *J. Netw. Intell.* **2021**, *6*, 206–215.
- Huang, H.; Lu, S.; Wu, Z.; Wei, Q. An efficient authentication and key agreement protocol for IoT-enabled devices in distributed cloud computing architecture. *EURASIP J. Wirel. Commun. Netw.* **2021**, *2021*, 150. [CrossRef]
- Wu, T.Y.; Wang, T.; Lee, Y.Q.; Zheng, W.; Kumari, S.; Kumar, S. Improved authenticated key agreement scheme for fog-driven IoT healthcare system. *Secur. Commun. Netw.* **2021**, *2021*, 6658041. [CrossRef]
- Yang, Y.; Zheng, X.; Guo, W.; Liu, X.; Chang, V. Privacy-preserving fusion of IoT and big data for e-health. *Future Gener. Comput. Syst.* **2018**, *86*, 1437–1455. [CrossRef]
- Mushtaq, M.F.; Akram, U.; Khan, I.; Khan, S.N.; Shahzad, A.; Ullah, A. Cloud computing environment and security challenges: A review. *Int. J. Adv. Comput. Sci. Appl.* **2017**, *8*, 183–195.
- Wu, T.Y.; Meng, Q.; Kumari, S.; Zhang, P. Rotating behind Security: A Lightweight Authentication Protocol Based on IoT-Enabled Cloud Computing Environments. *Sensors* **2022**, *22*, 3858. [CrossRef]
- Chen, X.; Zhang, J.; Lin, B.; Chen, Z.; Wolter, K.; Min, G. Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 683–697. [CrossRef]
- Zhang, J.; Li, M.; Chen, Z.; Lin, B. Computation offloading for object-oriented applications in a UAV-based edge-cloud environment. *J. Supercomput.* **2022**, *78*, 10829–10853. [CrossRef]
- Kang, B.; Han, Y.; Qian, K.; Du, J. Analysis and improvement on an authentication protocol for IoT-enabled devices in distributed cloud computing environment. *Math. Probl. Eng.* **2020**, *2020*, 1970798. [CrossRef]
- Iqbal, U.; Tandon, A.; Gupta, S.; Yadav, A.R.; Neware, R.; Gelana, F.W. A Novel Secure Authentication Protocol for IoT and Cloud Servers. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 7707543. [CrossRef]
- Amin, R.; Kumar, N.; Biswas, G.; Iqbal, R.; Chang, V. A light weight authentication protocol for IoT-enabled devices in distributed Cloud Computing environment. *Future Gener. Comput. Syst.* **2018**, *78*, 1005–1019. [CrossRef]
- Zhou, L.; Li, X.; Yeh, K.H.; Su, C.; Chiu, W. Lightweight IoT-based authentication scheme in cloud computing circumstance. *Future Gener. Comput. Syst.* **2019**, *91*, 244–251. [CrossRef]
- Martínez-Peláez, R.; Toral-Cruz, H.; Parra-Michel, J.R.; García, V.; Mena, L.J.; Félix, V.G.; Ochoa-Brust, A. An enhanced lightweight IoT-based authentication scheme in cloud computing circumstances. *Sensors* **2019**, *19*, 2098. [CrossRef]
- Costan, V.; Devadas, S. Intel SGX Explained. Cryptology ePrint Archive. 2016. Available online: <https://ia.cr/2016/086> (accessed on 21 February 2017).

26. Liu, X.; Guo, Z.; Ma, J.; Song, Y. A secure authentication scheme for wireless sensor networks based on DAC and Intel SGX. *IEEE Internet Things J.* **2021**, *9*, 3533–3547. [[CrossRef](#)]
27. Wu, T.Y.; Guo, X.; Chen, Y.C.; Kumari, S.; Chen, C.M. SGXAP: SGX-Based Authentication Protocol in IoV-Enabled Fog Computing. *Symmetry* **2022**, *14*, 1393. [[CrossRef](#)]
28. Jain, P.; Desai, S.J.; Shih, M.W.; Kim, T.; Kim, S.M.; Lee, J.H.; Choi, C.; Shin, Y.; Kang, B.B.; Han, D. OpenSGX: An Open Platform for SGX Research. In Proceedings of the NDSS, San Diego, CA, USA, 21–24 February 2016; Volume 16, pp. 21–24.
29. Turkanović, M.; Brumen, B.; Hölbl, M. A novel user authentication and key agreement scheme for heterogeneous ad hoc wireless sensor networks, based on the Internet of Things notion. *Ad Hoc Netw.* **2014**, *20*, 96–112. [[CrossRef](#)]
30. Farash, M.S.; Turkanović, M.; Kumari, S.; Hölbl, M. An efficient user authentication and key agreement scheme for heterogeneous wireless sensor network tailored for the Internet of Things environment. *Ad Hoc Netw.* **2016**, *36*, 152–176.
31. Amin, R.; Islam, S.H.; Biswas, G.; Khan, M.K.; Leng, L.; Kumar, N. Design of an anonymity-preserving three-factor authenticated key exchange protocol for wireless sensor networks. *Comput. Netw.* **2016**, *101*, 42–62.
32. Wu, F.; Xu, L.; Kumari, S.; Li, X.; Shen, J.; Choo, K.K.R.; Wazid, M.; Das, A.K. An efficient authentication and key agreement scheme for multi-gateway wireless sensor networks in IoT deployment. *J. Netw. Comput. Appl.* **2017**, *89*, 72–85.
33. Liu, H.; Ning, H.; Xiong, Q.; Yang, L.T. Shared authority based privacy-preserving authentication protocol in cloud computing. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *26*, 241–251. [[CrossRef](#)]
34. Tsai, J.L.; Lo, N.W. A privacy-aware authentication scheme for distributed mobile cloud computing services. *IEEE Syst. J.* **2015**, *9*, 805–815. [[CrossRef](#)]
35. He, D.; Kumar, N.; Khan, M.K.; Wang, L.; Shen, J. Efficient privacy-aware authentication scheme for mobile cloud computing services. *IEEE Syst. J.* **2016**, *12*, 1621–1631. [[CrossRef](#)]
36. Kumar, V.; Jangirala, S.; Ahmad, M. An efficient mutual authentication framework for healthcare system in cloud computing. *J. Med Syst.* **2018**, *42*, 142. [[CrossRef](#)] [[PubMed](#)]
37. G Lopes, A.P.; Gondim, P.R. Mutual authentication protocol for D2D communications in a cloud-based e-health system. *Sensors* **2020**, *20*, 2072. [[CrossRef](#)]
38. Wang, F.; Xu, G.; Xu, G.; Wang, Y.; Peng, J. A robust IoT-based three-factor authentication scheme for cloud computing resistant to session key exposure. *Wirel. Commun. Mob. Comput.* **2020**, *2020*, 3805058. [[CrossRef](#)]
39. Yu, S.; Park, K.; Park, Y. A secure lightweight three-factor authentication scheme for IoT in cloud computing environment. *Sensors* **2019**, *19*, 3598. [[CrossRef](#)]
40. Wu, H.L.; Chang, C.C.; Zheng, Y.Z.; Chen, L.S.; Chen, C.C. A Secure IoT-Based Authentication System in Cloud Computing Environment. *Sensors* **2020**, *20*, 5604. [[CrossRef](#)]
41. Fisch, B.; Vinayagamurthy, D.; Boneh, D.; Gorbunov, S. Iron: Functional encryption using Intel SGX. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 765–782. [[CrossRef](#)]
42. Sun, H.; Xiao, S. DNA-X: Dynamic network authentication using SGX. In Proceedings of the 2nd International Conference on Cryptography, Security and Privacy, Guiyang, China, 16–19 March 2018; pp. 110–115. [[CrossRef](#)]
43. Condé, R.C.; Maziero, C.A.; Will, N.C. Using Intel SGX to protect authentication credentials in an untrusted operating system. In Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018; pp. 158–163. [[CrossRef](#)]
44. Song, T.; Wang, W.; Lang, F.; Ouyang, W.; Wang, Q.; Lin, J. P2A: Privacy Preserving Anonymous Authentication Based on Blockchain and SGX. In Proceedings of the International Conference on Information Security and Cryptology, Guangzhou, China, 11–14 December 2020; Springer: Berlin/Heidelberg, Germany, 2020; Volume 12612, pp. 257–276.
45. Abdalla, M.; Fouque, P.A.; Pointcheval, D. Password-based authenticated key exchange in the three-party setting. In Proceedings of the International Workshop on Public Key Cryptography, Les Diablerets, Switzerland, 23–26 January 2005; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3386, pp. 65–84.
46. Wu, T.Y.; Meng, Q.; Yang, L.; Guo, X.; Kumari, S. A provably secure lightweight authentication protocol in mobile edge computing environments. *J. Supercomput.* **2022**, *78*, 13893–13914. [[CrossRef](#)]
47. Dolev, D.; Yao, A. On the security of public key protocols. *IEEE Trans. Inf. Theory* **1983**, *29*, 198–208. [[CrossRef](#)]
48. Canetti, R.; Krawczyk, H. Analysis of key-exchange protocols and their use for building secure channels. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Innsbruck, Austria, 6–10 May 2001; Springer: Berlin/Heidelberg, Germany, 2001; Volume 2045, pp. 453–474.
49. Messerges, T.S.; Dabbish, E.A.; Sloan, R.H. Examining smart-card security under the threat of power analysis attacks. *IEEE Trans. Comput.* **2002**, *51*, 541–552. [[CrossRef](#)]
50. Wang, D.; Cheng, H.; Wang, P.; Huang, X.; Jian, G. Zipf’s law in passwords. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 2776–2791. [[CrossRef](#)]
51. Blanchet, B. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the CSFW*; Citeseer: Princeton, NJ, USA, 2001; Volume 1, pp. 82–96.

52. Wu, T.Y.; Lee, Z.; Yang, L.; Luo, J.N.; Tso, R. Provably secure authentication key exchange scheme using fog nodes in vehicular ad hoc networks. *J. Supercomput.* **2021**, *77*, 6992–7020. [[CrossRef](#)]
53. Wang, J.; Hao, S.; Li, Y.; Fan, C.; Wang, J.; Han, L.; Hong, Z.; Hu, H. Challenges towards protecting vnf with sgx. In Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, Tempe, AZ, USA, 21 March 2018; pp. 39–42. [[CrossRef](#)]