History-based P2P Key Management Protocol for IoT Environments

Hyeon Ho Lee
Information & Communication Engineering
Chungbuk national university
Chungju-si, South Korea
hhl9438@cbnu.ac.kr

Won Seok Choi
Information & Communication Engineering
Chungbuk national university
Chungju-si, South Korea
choiws@cbnu.ac.kr

Seong Gon Choi
Information & Communication Engineering
Chungbuk national university
Chungju-si, South Korea
choisg@cbnu.ac.kr

***Abstract*— We propose a history-based peer to peer (P2P) key management protocol in an Internet of Things (IoT) environment. Currently, key management for data encryption for secured channels in IoT relies on Public Key Infrastructure (PKI), which causes a single point of failure due to data sovereignty and centralization issues with certification authority (CA). Therefore, we propose a P2P key management protocol that utilizes data exchanged in an IoT environment. Our protocol features pre-shared key setup for initial authentication and efficient P2P key exchange for secure communication establishment. The proposed protocol registers data between devices in advance when they are not connected to the network, and then updates new keys based on the key exchange process and communication logs using that data. We implemented a prototype at the laboratory level to verify the feasibility of the proposed protocol.**

Keywords*— *Internet of Things(IoT), Peer to Peer(P2P), Key Management, History-based, Security

I. INTRODUCTION

With the rapid spread of Internet of Things (IoT) technology, the number of connected devices is growing exponentially. The number of connected IoT devices is expected to continue to increase, with over 400 billion IoT devices expected to be active worldwide by 2030 [1]. IoT devices are used in various applications such as smart homes, wearable devices, autonomous vehicles, and industrial IoT, continuously collecting and processing sensitive personal information such as users' behavior patterns, location information, biometric data, and environmental information. In this context, our data is transmitted through the edge-cloud path, and privacy exposure issues arising during this transmission process, as well as the security measures to prevent them, are critical. Currently, IoT devices communicating over networks are vulnerable to security attacks [2].

In such large-scale IoT environments, robust security is essential to ensure secure communication between devices. In particular, device authentication and encryption key management are core elements of IoT security, and most current IoT systems rely on Public Key Infrastructure (PKI). PKI is a proven security framework that uses digital certificates to verify device identities and establish secure communication channels, and has been widely used in traditional Internet environments [3]. However, PKI has inherent single failure issues due to the centralization of the Certificate Authorities (CA) that issues certificates. If the centralized certification authority of a manufacturer or service provider is attacked or experiences a failure, all devices dependent on that CA are exposed to risk, which can result in a serious problem where numerous devices are simultaneously exposed to security risks. The centralization of the CA system concentrates trust in a relatively small number of entities, creating a single point of failure. If the CA is attacked or acts maliciously, all encrypted communications of IoT devices dependent on it can be decrypted, potentially causing severe privacy issues [4][5].

Furthermore, in PKI, there are still factors that could compromise the data sovereignty of individuals and organizations. Core security data such as certificate issuance history, revocation information, and revocation lists are under the control of CA operators, and users cannot fully control even the authentication information related to their devices. This means that users are unaware if CA analyze their data without consent, or if CA provide user data to government agencies upon request. In IoT environments, the risk of exposing personal data increases significantly. This structure causes users to lose autonomy over the security infrastructure and fails to guarantee data ownership and control, which are essential in the data economy era [6][7].

To address these issues, users need to be able to manage keys directly without the intervention of central authorities or third parties. Therefore, we propose a history-based peer to peer (P2P) Key management protocol. The key idea is to combine offline pre-shared secrets with accumulated communication history, enabling autonomous key management without central authorities while maintaining forward secrecy

This paper is organized as follows. In Section 2 reviews existing studies. In Section 3 details the pre-shared key setup, P2P key exchange and history-based update protocol of the proposed protocol. In Section 4 implements the proposed protocol as a prototype to verify its feasibility. Finally, In Section 5 presents conclusions and future research directions.

II. RELATED WORK

In this section review the main approaches to P2P key exchange in recent IoT environments.

Pham and Dang proposed a lightweight authentication protocol for D2D-based IoT systems [8]. They implemented direct authentication between devices without a server and minimized computational overhead using a symmetric key. However, this approach did not provide a secure method for sharing the initial key and lacked a mechanism for updating the key once set, increasing security risks over extended use.

Zheng et al. proposed a P2P IoT authentication protocol utilizing PUF (Physical Unclonable Function) [9]. They achieved secure authentication without a server by generating

unique keys using the unique physical characteristics of hardware. However, this approach requires special hardware with built-in PUF chips, making it impractical for the numerous existing IoT devices already deployed.

Khalid et al. proposed a distributed lightweight blockchain-based authentication mechanism for IoT [10]. While they removed central authority using blockchain, this approach requires each IoT device to act as a blockchain node, which is an unrealistic resource requirement. Additionally, transaction delays and energy consumption issues make it unsuitable for real-time IoT environments.

Therefore, existing approaches suffer from either practical deployment challenges, such as special hardware requirements and excessive resource demands, or fundamental security limitations in key management. This paper proposes a practical P2P key management protocol that addresses these issues through offline pre-shared key setup and history-based key updates. The proposed approach operates solely through software without additional hardware, making it applicable to all IoT devices.

III. Proposed Method

In this section we provide a detailed describes the target environment and the History-based P2P Key management protocol.

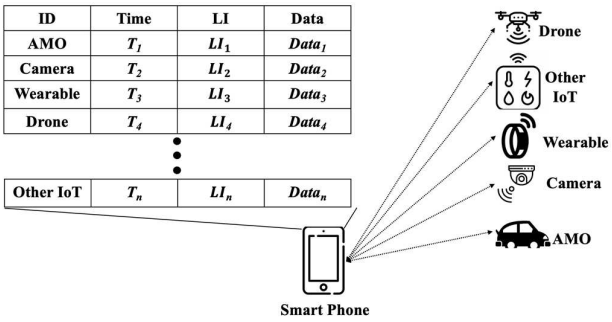

Fig. 1. Target Environment

A. Target Environment

Fig. 1 shows the target environment of the proposed protocol. Each IoT device is connected to a P2P network, and a smartphone acts as a hub. During initial setup, all devices must set a secret key (*SK*) when they are first launched. When setting up the first device, the user enters the *SK* directly. Subsequently, all devices must enter the same *SK* set on the first device. This secret key is known only to the user and is consistently used during the initial connection of all devices owned by the user. The format of the *SK* can be set as desired by the user. The first connection between devices must be established only in the initial state without a network connection. when a device connects to another device for the first time, it shares three pieces of information: the *SK*, the connection request time, and the location information at the time of the connection request. Therefore, if the user remembers the *SK* set on the first device, they can add an unlimited number of devices.

The table in Fig. 1 shows the three pieces of information stored by each device. The initial data field contains the *SK* shared by all devices. These three pieces of information are not the keys used for actual communication encryption. They are used as authentication means to exchange secure random session keys between devices. For example, when a smartphone and a wearable device communicate for the first time, they select two of the following three pieces of information *SK*, Time (*T*), and Location Information (*LI*) encrypt them and exchange them through a series of procedures to safely exchange a new random session key and authenticate each other. Subsequent actual data communication is encrypted using the exchanged random session key. At this point, the sender transmits the encrypted message along with the current time and location information, and the receiver stores this in a log. For example, when a drone transmits data collected at a specific time and location to a smartphone, the smartphone records *T*, *LI*, and *Data* as communication records. During key update, the devices select one of the accumulated communication records and set it as the new T_i, LI_i, and $Data_i$. This process accumulates communication records with each communication and periodically renews them, enabling the security context to evolve dynamically. Therefore, the initial three pieces of information are used only for the first key exchange, and subsequent communication records transmitted from each device continuously strengthen security.

B. Pre-shared Key Setup

The setup process consists of two steps. In the first step, each device stores the *SK* entered by the user during initial startup in a secure storage location. This key must be entered identically on all user-owned devices and serves as the trust root for all subsequent security operations. The second step is the security connection setup process performed when two devices with the same *SK* first encounter each other. The devices first confirm that the network connection is completely blocked, then establish a direct connection via Device to Device (D2D) such as Bluetooth or Wi-Fi Direct. Once the connection is established, the two devices perform mutual authentication using the pre-shared *SK*. This protocol assumes a trusted environment, so the SK is exchanged directly at this stage to verify that it matches. If the *SK* matches, each device generates the current time and location information and exchanges it with the other device. The *T* and *LI* generated at this point are unique values for the specific device pair and are completely independent of connections between other device pairs. For example, values generated in the connection between devices A and B are unrelated to values generated in the connection between devices B and C. After selecting one of the values generated by both sides or determining the final value through an agreed-upon method, each device stores the *SK*, *T*, and *LI* along with the other device's identifier in the security context of the connection. This process provides several key security features. Network isolation blocks remote attacks at their source. All initialization tasks are performed without an internet connection, preventing remote attackers from interfering with the initial setup process. The most vulnerable initial setup stage is restricted to attackers with physical access. Mutual authentication prevents spoofing attacks. Two devices must have the same *SK* to establish a secure connection, making it impossible for attackers to impersonate legitimate devices.

A new *T* and *LI* are generated each time a connection is established, preventing the reuse of connection information from past connections with other devices. This allows users to manage all security parameters directly without the involvement of other authentication authorities, thereby

securing sovereignty over the data they create. Each device pair maintains an independent security context, providing isolated security zones that prevent the spread of a breach from one connection to another, minimizing the impact of attacks and enhancing the overall resilience of the system. This setup is automatic, requiring no user intervention beyond the initial *SK* entry, and the physical proximity requirement for D2D communication provides an additional layer of security without causing inconvenience in everyday use. As a result, users only need to remember and manage a single *SK*, while *T* and *LI* are automatically generated during connection. This design effectively balances strong security and practicality, enabling scalable and secure device management in personal IoT environments.

C. P2P Key Exchange

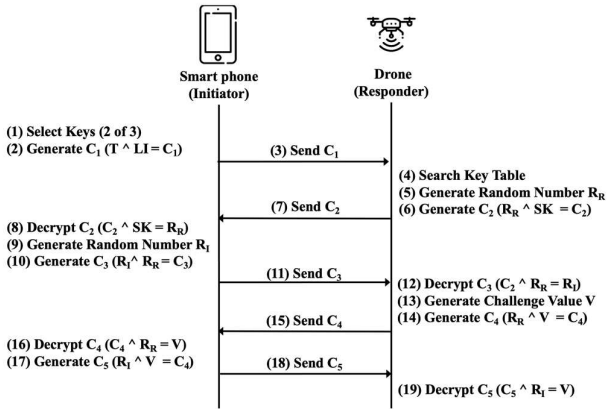

Fig. 2. P2P Key Exchange Protocol Example

Fig. 2 shows the key exchange process between a smartphone and a drone in a sequence diagram. For clarity, the encryption method has been simplified to XOR, and mutual authentication and key exchange are performed in five steps according to the Challenge-Response paradigm. The smartphone acts as the Initiator, which starts communication, and the drone acts as the Responder, which responds to the Initiator's request.

In the first step of the protocol, the smartphone randomly selects two pieces of information from the pre-shared *SK, T, LI* and performs encryption. After that, the drone receives C_1 and calculates all possible combinations using the three keys shared with the smartphone to find the pair that matches C_1. Only authorized devices possess the same three keys, so only pre-registered devices can correctly interpret C_1. The drone identifies the unused key, generates the encryption key R_R using that key, and transmits it as C_2. The smartphone receives C_2, decrypts it using the unused key, and obtains the drone's encryption key R_R. Next, the smartphone generates its own encryption key R_I, encrypts it using the recently obtained R_R, and sends it to C_3. The key point of this process is that each message is encrypted based on information exchanged in the previous step. This means that each step of the protocol is logically connected, preventing a man-in-the-middle attacker from arbitrarily inserting or modifying messages. The last two steps of the protocol involve mutual verification through a challenge-response mechanism. The drone generates a random challenge value V, encrypts it with R_R, and sends it to C_4. The smartphone decrypts it to obtain V, re-encrypts it with R_I, and responds with C_5. The drone verifies that the decrypted result of C_5 matches the original V to confirm that the smartphone holds the correct R_I.

Algorithm 1a: P2P Key Exchange - Initiator

Input: (*SK, T, LI*)
Output: (R_I, R_R) or *FAIL*
1: *keys* ← [*SK, T, LI*]
2: (K_1, K_2) ← *RandomSelect*(*2 from keys*)
3: $C_1 \leftarrow K_1 \oplus K_2$
4: *Send*(C_1)
5: C_2 ← *Receive(timeout)* // *Wait for responder's key*
6: **if** C_2 = *NULL* **then**
7: **return** *FAIL*
8: **end if**
9: *unusedKey* ← *keys* – {K_1, K_2}
10: $R_R \leftarrow C_2 \oplus unusedKey$
11: R_I ← *GenerateRandom()*
12: $C_3 \leftarrow R_I \oplus R_R$
13: *Send*(C_3)
14: C_4 ← *Receive(timeout)* // *Receive challenge*
15: **if** C_4 = *NULL* **then**
16: **return** *FAIL*
17: **end if**
18: $V \leftarrow C_4 \oplus R_R$
19: $C_5 \leftarrow V \oplus R_I$
20: *Send*(C_5)
21: **if** *VerificationSuccess()* **then**
22: *SecureStore(PeerID,* R_I, R_R)
23: **return** (R_I, R_R)
24: **else**
25: **return** *FAIL*
26: **end if**

Algorithm 1a defines the actions performed by the initiator in the key exchange for establishing secure channel communication. The initiator randomly selects two of the three keys it owns and performs encryption to generate C_1. This random selection generates different values for each session to prevent replay attacks. For example, if the first session uses the *T* and *LI* pair, the next session will include the *SK* combined with either the *T* or *LI*. After receiving C_2, the initiator decrypts C_2 using the unused key to extract the respondent's encryption key R_R. This process includes a timeout to prevent infinite waiting, and if no response is received, *FAIL* is returned. This demonstrates the efficiency of immediately identifying the correct decryption key using pre-shared information. Subsequently, it generates its own encryption key R_I, encrypts it using the recently obtained R_R, and transmits it as C_3. This ensures the confidentiality of R_I and implicitly verifies that the responder successfully decrypted the initial challenge. The final step is challenge-response verification. Upon receiving C_4, the initiator decrypts it using R_R and then re-encrypts it using its own R_I before responding. This mutual verification process proves that both parties possess the correct keys. If verification succeeds, the protocol stores the encryption keys and peer IDs of both parties for use in subsequent secure communications. If the process fails, the connection attempt is terminated, ensuring that only authenticated connections are established.

Algorithm 1b: P2P Key Exchange - Responder

Input: (SK, T, LI)
Output: (R_I, R_R) or $FAIL$
1: $keys \leftarrow [SK, T, LI]$
2: $C_1 \leftarrow Receive(timeout)$ // Wait for initial challenge
3: **if** $C_1 = NULL$ **then**
4: **return** $FAIL$
5: **end if**
6: $(K_1, K_2) \leftarrow SearchKeyPair(C_1, keys)$
7: **if** *no match found* **then**
8: **return** $FAIL$
9: **end if**
10: $unusedKey \leftarrow keys - \{K_1, K_2\}$
11: $R_R \leftarrow GenerateRandom()$
12: $C_2 \leftarrow R_R \oplus unusedKey$
13: $Send(C_2)$
14: $C_3 \leftarrow Receive(timeout)$ // Wait for initiator's key
15: **if** $C_3 = NULL$ **then**
16: **return** $FAIL$
17: **end if**
18: $R_I \leftarrow C_3 \oplus R_R$
19: $V \leftarrow GenerateRandom()$
20: $C_4 \leftarrow V \oplus R_R$
21: $Send(C_4)$
22: $C_5 \leftarrow Receive(timeout)$
23: **if** $C_5 = NULL$ **then**
24: **return** $FAIL$
25: **end if**
26: $V' \leftarrow C_5 \oplus R_I$
27: **if** $V' = V$ **then**
28: $SecureStore(PeerID, R_I, R_R)$
29: $SendACK()$
30: **return** (R_I, R_R)
31: **else**
32: **return** $FAIL$
33: **end if**

Algorithm 1b defines the tasks performed by the responder in the key exchange for establishing secure channel communication. The responder waits to receive the initial challenge signal C_1 from the initiator. Upon receiving C_1, the responder searches all possible combinations of its three keys to identify the key pair used to generate C_1. This process functions as an authentication mechanism because only devices that possess the exact pre-shared key can successfully identify the combination. Once the key pair is identified, the responder determines which key is unused and generates its own encryption key R_R. It then uses this key to encrypt R_R and send it to C_2. This process ensures that only the legitimate initiator, who knows the original key, can decrypt R_R. After receiving C_3, the responder decrypts it with R_R to extract the initiator's encryption key R_I. This implicitly confirms that the initiator has correctly received and processed R_R, providing mutual authentication without the need for additional message exchanges. Finally, a challenge-response mechanism is used. The responder generates a random challenge value V, encrypts it with R_R, and sends it to C_4. Upon receiving the response C_5, it is decrypted using R_I, and the result is verified to match the original challenge. This proves that the sender possesses the correct R_I and has successfully completed the key exchange. If verification succeeds, the responder stores both encryption keys along with the peer ID and sends a confirmation response before returning the key pair. Any failure during this process results in immediate termination, maintaining the security integrity of the system. This design ensures that both parties have mutually authenticated each other and established separate encryption keys for bidirectional secure communication.

D. History-based key update

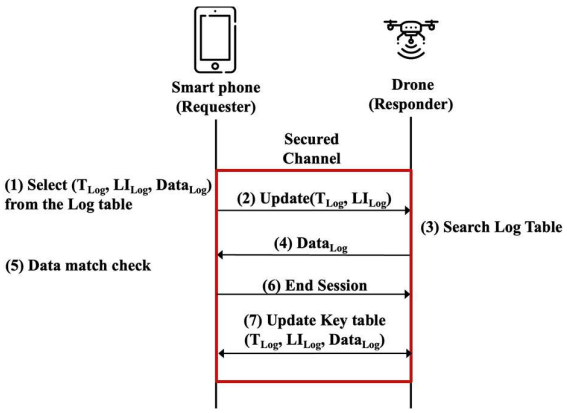

Fig. 3. History-based key update Example

Fig. 3 shows the key update process between a smartphone and a drone as an example. This mechanism is performed within the existing secured channel and utilizes past communication records as new security parameters. When communicating through the secured channel, the sender constructs a message that includes the current timestamp and location information along with the actual data, then encrypts the entire message. Specifically, the message structure consists of a header containing timestamp and location information, followed by the data payload, and the complete message is encrypted as $M = Encrypt(Message)$. The receiver decrypts the message, extracts the timestamp and location information from the header, and stores them along with the received data in a log table. The key point is that both the sender and receiver use the same timestamp and location values from the message header to create their respective logs. This ensures that both devices maintain fully synchronized communication records, which serve as the basis for mutual verification during future key updates.

Key updates can be initiated by either device. The device requesting the update selects an arbitrary record from the log table and transmits two of the three elements. This incomplete information serves as a knowledge proof mechanism, prompting the other device to provide the missing third element. The receiving device searches its log table based on the two received elements. If the communication partner is legitimate, it will have the same record and extract the missing third element from the matching entry to respond. If the received value matches the original value held by the requester, it is proven that both devices share the same communication record. If verification succeeds, a session termination message is transmitted, and both devices update the key table with the communication record T_{Log}, LI_{Log}, and $Data_{Log}$ as the new security context, replacing the three values that originally includes SK. Subsequent communications re-establish the secure connection using the updated parameters. If verification fails, the protocol is immediately terminated, and no key update is performed. This mechanism utilizes past

communication records, preventing external attackers from predicting or forging records, and ensures diversity by selecting different records for each update. Periodic key updates minimize the risk of long-term key exposure, and devices can autonomously enhance security without additional authentication infrastructure. In particular, as communication records accumulate, the number of update options available increases, improving security over time.

IV. Implementation

This section describes the actual implementation and experimental environment of the proposed protocol.

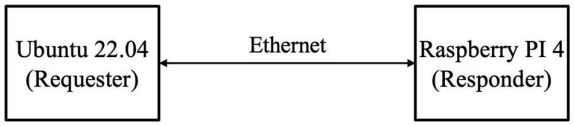

Fig. 4. Test Environment

The proposed protocol was implemented in C language to verify its feasibility. The experimental environment used a PC (Intel i7-12700, 32GB RAM) running Ubuntu 22.04 as the requester and a Raspberry Pi 4 as the responder, with the two devices directly connected via an Ethernet cable as shown in Fig. 4. This configuration is intended for laboratory-level verification, using basic XOR encryption operations, with the aim of demonstrating that the proposed protocol can be implemented in a real environment.

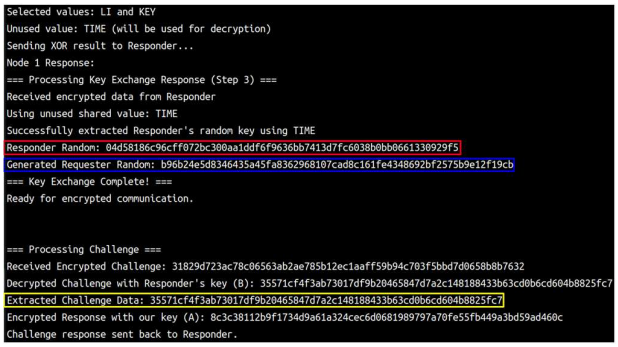

Fig. 5. Requester(PC)

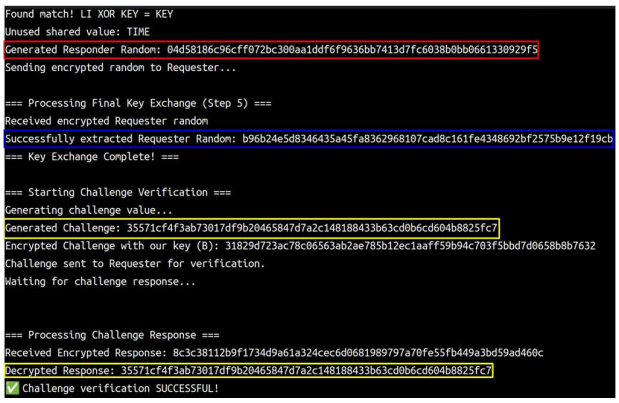

Fig. 6. Responder(Raspberry Pi 4)

Fig. 5 shows the protocol execution process on the requester side. In the initialization phase, the requester selects two of the three shared values and performs an XOR operation on them. In Fig. 5, LI and KEY are selected. The calculated result is sent to the responder, and the requester then waits for a response. The highlighted area indicates the random number received from the responder, which serves as the basis for generating the subsequent session key.

Fig. 6 shows the responder's response process. The responder analyzes the received XOR value and successfully identifies TIME as an unused value. This confirms that the protocol's core mechanism of finding unused values is functioning correctly. The responder generates its own random number for exchange with the requester and successfully extracts the other party's random number, as shown in the highlighted section. Both parties perform additional Challenge-Response verification using the exchanged random numbers. The "Challenge verified successfully" message appears on the requester's side, and the verification value is correctly decrypted and displayed on the responder's side, confirming the completion of the protocol. This implementation demonstrates that the proposed protocol enables secure key exchange even with encryption using only lightweight XOR operations.

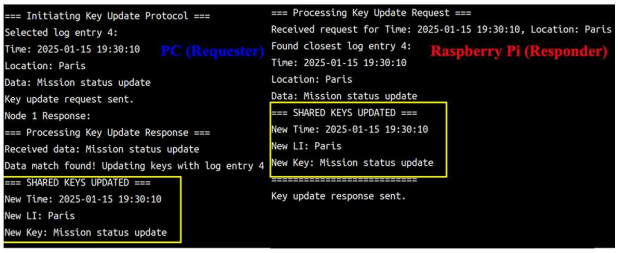

Fig. 7. Key Update

Fig. 7 shows the execution of the history-based key update protocol. The requester (PC, left) selects log index 4, which contains the timestamp "2025-01-15 19:30:10," location "Paris" and data "Mission status update" from the communication history. Upon receiving the update request, the responder (Raspberry Pi 4, right) successfully finds the matching entry in the synchronized log table. Both parties verify that they share the same communication history by confirming the data match. The highlighted section shows that all three shared values *T, LI* and *Data* have been successfully updated on both sides. This proves that the protocol effectively performs dynamic key update and provides forward secrecy by utilizing past communication records without additional key exchange overhead. The synchronized update demonstrates the protocol's ability to maintain a consistent security context between resource-constrained devices using only communication history data.

V. Conclusions

We propose a history-based P2P key management protocol for IoT environments. This protocol does not rely on central infrastructure and enables dynamic key updates by utilizing shared communication history between devices. By using only lightweight XOR operations and maintaining synchronized log tables, our approach provides forward secrecy and mutual authentication suitable for IoT environments. The implementation was carried out in C language, and experimental verification through direct Ethernet connection between a PC and Raspberry Pi 4 confirmed the feasibility of the protocol at the laboratory level.

In further study, we plan to expand the protocol, which is currently limited to single-user devices, to enable the establishment of secure channels between different users.

ACKNOWLEDGMENT

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. RS-2020-NR049604, 100%).

*corresponding author is S.G. Choi(choisg@cbnu.ac.kr).

REFERENCES

[1] S. Sinha, "State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally," IoT Analytics, September 2024. [Online]. Available: https://iot-analytics.com/ number-connected-iot-devices/

[2] W. S. Choi, S. Y. Lee, and S. G. Choi, "Implementation and design of a zero-day intrusion detection and response system for responding to network security blind spots," Mobile Inf. Syst., vol. 2022, pp. 1–13, April 2022.

[3] M. El-Hajj and P. Beune, "Lightweight public key infrastructure for the Internet of Things: A systematic literature review," J. Ind. Inf. Integr., vol. 41, September 2024.

[4] D. Díaz-Sánchez, A. Marín-Lopez, F. A. Mendoza, P. A. Cabarcos and R. S. Sherratt, "TLS/PKI Challenges and Certificate Pinning Techniques for IoT and M2M Secure Communications," in *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3502-3531, Fourthquarter 2019, doi: 10.1109/COMST.2019.2914453.

[5] J. Höglund, S. Lindemer, M. Furuhed, and S. Raza, "PKI4IoT: Towards public key infrastructure for the Internet of Things," Comput. Security, vol. 89, February 2020.

[6] Z. Siddiqui, J. Gao and M. Khurram Khan, "An Improved Lightweight PUF–PKI Digital Certificate Authentication Scheme for the Internet of Things," in *IEEE Internet of Things Journal*, vol. 9, no. 20, pp. 19744-19756, 15 Oct.15, 2022, doi: 10.1109/JIOT.2022.3168726

[7] P. Porambage, A. Braeken, P. Kumar, A. Gurtov, and M. Ylianttila, "Privacy and security for resource-constrained IoT devices and networks: Research challenges and opportunities," Sensors, vol. 19, no. 9, April 2019.

[8] C. D. M. Pham and T. K. Dang, "A lightweight authentication protocol for D2D-enabled IoT systems with privacy," Pervasive Mobile Comput., vol. 74, July 2021.

[9] Y. Zheng, W. Liu, C. Gu and C. -H. Chang, "PUF-Based Mutual Authentication and Key Exchange Protocol for Peer-to-Peer IoT Applications," in *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 3299-3316, 1 July-Aug. 2023, doi: 10.1109/TDSC.2022.3193570.

[10] U. Khalid, M. Asim, T. Baker, P. Hung, M. Tariq, and L. Rafferty, "A decentralized lightweight blockchain-based authentication mechanism for IoT systems," Cluster Comput., vol. 23, no. 3, pp. 2067–2087, September 2020.