sensors

MDPI

Article

On the Security of a PUF-Based Authentication and Key Exchange Protocol for IoT Devices

Da-Zhi Sun [1,*], Yi-Na Gao [1] and Yangguang Tian [2]

[1] Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, Tianjin 300350, China; gaoyina@tju.edu.cn
[2] Department of Computer Science, University of Surrey, Surrey GU2 7XH, UK; yangguang.tian@surrey.ac.uk
* Correspondence: sundazhi@tju.edu.cn; Tel.: +86-22-2740-1091

Abstract: Recently, Roy et al. proposed a physically unclonable function (PUF)-based authentication and key exchange protocol for Internet of Things (IoT) devices. The PUF protocol is efficient, because it integrates both the Node-to-Node (N2N) authentication and the Node-to-Server (N2S) authentication into a standalone protocol. In this paper, we therefore examine the security of the PUF protocol under the assumption of an insider attack. Our cryptanalysis findings are the following. (1) A legitimate but malicious IoT node can monitor the secure communication among the server and any other IoT nodes in both N2N authentication and N2S authentication. (2) A legitimate but malicious IoT node is able to impersonate a target IoT node to cheat the server and any other IoT nodes in N2N authentication and the server in N2S authentication, respectively. (3) A legitimate but malicious IoT node can masquerade as the server to cheat any other target IoT nodes in both N2N authentication and N2S authentication. To the best of our knowledge, our work gives the first non-trivial concrete security analysis for the PUF protocol. In addition, we employ the automatic verification tool of security protocols, i.e., Scyther, to confirm the weaknesses found in the PUF protocol. We finally consider how to prevent weaknesses in the PUF protocol.

Keywords: physically unclonable function; authentication; key exchange; insider attack; surveillance; impersonation

Citation: Sun, D.-Z.; Gao, Y.-N.; Tian, Y. On the Security of a PUF-Based Authentication and Key Exchange Protocol for IoT Devices. *Sensors* **2023**, *23*, 6559. https://doi.org/10.3390/s23146559

Academic Editors: Habtamu Abie and Sandeep Pirbhulal

Received: 10 June 2023
Revised: 5 July 2023
Accepted: 13 July 2023
Published: 20 July 2023

1. Introduction

In recent years, the Internet of Things (IoT) has been growing rapidly and all kinds of IoT devices are often present in our personal lives in the form of various applications, such as smart homes, smart transportation, and smart cities. Figure 1 depicts the communication scenario between the IoT server and devices (nodes). However, security is one of the major challenges when IoT enters sensitive fields, e.g., a smart street light monitoring system. Unlike traditional networks, IoT consists of a large number of complex, heterogeneous, and interoperable IoT devices [1]. Hence, under a hostile environment, IoT devices are often vulnerable to attacks and therefore require stronger security techniques to overcome their potential security weaknesses, such as counterfeit identity and sensitive data leakage. Authentication and key exchange protocols can provide the service of authentication and secret session key establishment between various IoT devices. Here, the session key is used to support confidential data transfer between IoT devices.

However, designing authentication and key exchange protocols is an intractable task for IoT applications. Firstly, there is a huge number of IoT devices, but their computing and storage resources are usually limited. Therefore, lightweight protocols are urgently needed to realize efficient interaction between devices without overloading their performance. Secondly, IoT devices are often deployed in complex, heterogeneous network environments and can be connected to and accessed by multiple unknown and untrusted devices, leading to extreme security challenges. Robust authentication and key exchange protocols should satisfy all strict requirements from IoT devices and their applications.

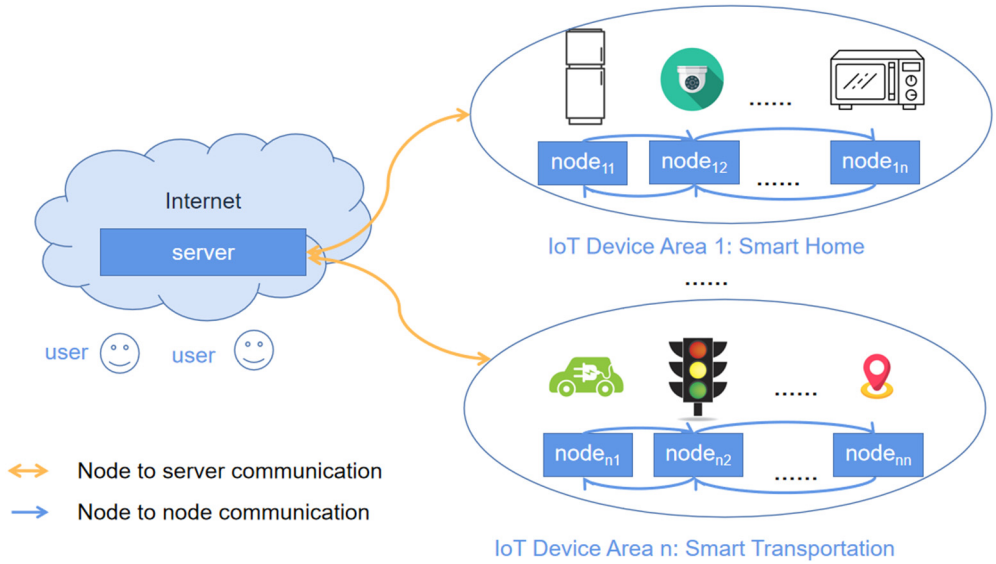


Figure 1. Architecture of IoT authentication network.

The physically unclonable function (PUF) [2] can help build robust security protocols. PUF is regarded as a basic security primitive for resource-constrained IoT devices. PUF generates a unique digital fingerprint from micro-variations in physical devices. This fingerprint is unable to be cloned and is sensitive to being tampered with. PUF generates an unpredictable output (i.e., response) for corresponding input (i.e., challenge). The process is irreversible and is independent of storage memory. PUF is realized in various scenarios, such as field programmable gate array (FPGA) [3], cloud storage, and IoT, to protect data from unauthorized access and attacks. The advantage of PUF is that resource-constrained devices do not require the maintenance of the secret key [4]. Moreover, it reduces the risks associated with key transmission, storage, and management to a greater extent than other common cryptographic techniques. Therefore, PUF is as a promising alternative to designing lower-cost authentication and key exchange protocols.

On the one hand, PUF has advantages in efficiency and security compared with cryptographic algorithms. Hence, it is expected to design more practical authentication and key exchange protocols for IoT. On the other hand, PUF-based authentication and key exchange protocols are relatively new, compared with the protocols using purely cryptographic algorithms and/or smart cards. And there are still many security issues in its designs, which should be addressed. Hence, in this paper, we investigate the PUF-based authentication and key exchange protocol for IoT devices.

1.1. Related Work

Cryptographic algorithms are widely used in security protocols. According to the type of cryptographic algorithm, we divide PUF-based authentication and key exchange protocols into two categories as follows:

- *The PUF-based authentication and key exchange protocols using public key algorithms.* The public key algorithms have high key management flexibility. This is because the communicating parties have two different keys, one of which is kept as a secret and the other is made public. There is no need to share the secret key as in symmetric key algorithms. The PUF-based protocols using public key algorithms inherit this feature. Yilmaz et al. [5] applied the RSA algorithm in the lightweight PUF-based authentication protocol. Chatterjee et al. [6] developed an authentication and key exchange protocol combining PUFs, identity-based encryption (IBE), and the keyed hash function. Chuang et al. [7] designed a PUF-based authenticated key exchange protocol for IoT devices without verifiers and an explicit challenge–response pair (CRP). Using certificateless public key cryptography (CL-PKC), Li et al. [8] proposed a PUF-

based end-to-end mutual authentication and key exchange protocol for IoT devices. Chaterjee et al. [9] proposed a private PUF-based anonymous authentication protocol. Siddiqui et al. [10] proposed a PUF-based authentication protocol, which is dependent on two certificate authorities of the cloud IoT system. Harishma et al. [11] proposed a PUF-based operationally asymmetric mutual authentication and key exchange protocol for secure communication. Qureshi and Munir [12] introduced a lightweight CRP obfuscation mechanism using XOR and shuffle operations and further used it to design the PUF-based key exchange protocol.

- *The PUF-based authentication and key exchange protocols using symmetric key algorithms.* The advantage of symmetric key algorithms is low computational cost. These algorithms are therefore well suited for authentication and key exchange protocols for resource-constrained IoT devices and help to reduce the computational burden of these IoT devices. In the PUF-based protocols using symmetric key algorithms, shared keys are generated by PUFs and hash functions are commonly used to authenticate messages. The researchers [13,14] proposed PUF-based mutual authentication protocols for IoT systems. Their protocols are presented for two scenarios, that is, device-to-device communication and device-to-server communication. Qureshi and Munir [15] presented a PUF-based identity-preserving protocol. During the authentication run of their protocol, the server does not store, generate, transmit, or receive complete devices' CRPs. Lounis and Zulkernine [16] proposed PUF-based Thing-to-Thing (T2T) architectures, where devices autonomously authenticate each other without any human intervention involved. Nimmy et al. [17] proposed a protocol that avoids explicit storage of CRPs for verification by using geometric threshold secret sharing. Zheng and Chang [18] proposed a new lightweight PUF-based mutual authentication and key exchange protocol for two PUF-embedded IoT devices. Wang et al. [19] leveraged PUFs to generate symmetric shared keys in the lightweight protocol. Ebrahimabadi et al. [20] designed an authentication protocol to thwart modeling attacks for PUF-based IoT devices. Zerrouki et al. [21] proposed a mutual authentication and session key establishment protocol for IoT devices based on PUFs. Sun and Tian [22] further gave security analysis and improvements to this protocol and suggested the idea of a key compromise to evaluate other novel PUF protocol designs. Wang et al. [23] introduced a supplementary sub-protocol to the PUF-based authentication protocol for the purpose of enhancing resistance to desynchronization attacks. Park and Park [24] employed PUFs to realize an improved authentication and key agreement protocol for cloud-enabled IoT devices. To achieve decentralization and security, Aseeri et al. [25] introduced a secure, lightweight, cost-efficient reinforcement machine learning framework (SLCR-MLF) with PUFs. In addition, some researchers [26–28] proposed PUF-based two-factor authentication mechanisms.

1.2. Our Motivations and Contributions

In *IEEE Internet Things J.* 2023, 10, 8547–8559, Roy et al. [29] proposed a PUF-based authentication and key exchange protocol. Roy et al.'s protocol is interesting because it tries to leverage cryptographic XOR operation and hash function for traditional secure communication and PUF for preventing physical attacks. Moreover, this standalone protocol can perform device-to-device and device-to-server authentication, eradicating the need for disparate protocols. In addition, using the Scyther verifier tool, the security of Roy et al.'s protocol is evaluated by the formal method.

Due to their inherent vulnerability, IoT devices (nodes) are easily compromised by insider attackers. Therefore, in this paper, we carefully study the security of Roy et al.'s protocol under the assumption of an inside attack. Our results are summarized as follows:

(1) A legitimate but malicious IoT node can monitor the secure communication among the server and any other IoT nodes, because the malicious IoT node can reveal their secret session key. The only requirement of the malicious IoT node is to observe the

transmitted messages over public channel during the authentication and key exchange phase of Roy et al.'s protocol.

(2) A legitimate but malicious IoT node is able to impersonate a target IoT node to cheat the server and any other IoT nodes during the authentication and key exchange phase of Roy et al.'s protocol. Finally, the malicious IoT node can establish a secret session key for the subsequent secure communication.

(3) A legitimate but malicious IoT node can masquerade as the server to cheat any other target IoT nodes during the authentication and key exchange phase of Roy et al.'s protocol. Here, the malicious IoT node is able to generate the valid session key for the target IoT nodes.

2. Basic Knowledge of PUF

PUFs make use of the intrinsic random variability in the physical microstructure of integrated circuits (ICs) to produce a unique n-bit response R_i to an m-bit challenge C_i for any $i \in \mathbb{N}$. We can write a PUF's instance as follows.

$$f_{PUF}: C_i \rightarrow R_i, \tag{1}$$

where $C_i \in \{0,1\}^m$ and $R_i \in \{0,1\}^n$.

PUF has the following safety features:

- Uniqueness: PUFs cannot output the same response for different input challenges, while different PUFs output different responses for the same input challenge. For two different C_i and C_j, with $i, j \in \mathbb{N}$, and any two different PUF instances ${f^A}_{PUF}$ and ${f^B}_{PUF}$, the uniqueness expression is as follows.

$${f^A}_{PUF}(C_i) \neq {f^A}_{PUF}(C_j). \tag{2}$$

$${f^A}_{PUF}(C_i) \neq {f^B}_{PUF}(C_i). \tag{3}$$

- Reliability: This feature measures the reproducibility of a PUF response under different operating conditions for a given challenge. The reliability expression of the PUF instance for different time periods t_1 and t_2 is:

$$f_{PUF}(C_i) \mid_{t=t_1} = f_{PUF}(C_i) \mid_{t=t_2}. \tag{4}$$

In fact, the PUF response is sensitive to various noise elements, such as temperature, voltage, and other environmental changes. Under the same challenge, the noise can cause the PUF to give an erroneous response, which is different from the original one. To eliminate the effects of such noise, Roy et al. [29] considered the use of lightweight error-correcting algorithms (ECAs) to stabilize the noisy PUF response and thus improved its reliability.

3. Review of Roy et al.'s Protocol

The one-time enrollment phase is responsible for enrolling the new IoT nodes in the server. The authentication and key exchange phase realizes mutual authentication between the server and the node(s). To maintain consistency, we employ the same notions of [29] and write them in Table 1.

Table 1. Used notation and symbols.

Notation	Description
A, B, C	IoT node A, IoT node B, IoT node C
id_A, id_B, id_C	A's identifier, B's identifier, C's identifier
C_{Ai}, C_{Bi}, C_{Ci}	A's ith iteration PUF input, B's ith iteration PUF input, C's ith iteration PUF input
R_{Ai}, R_{Bi}, R_{Ci}	A's PUF output for C_{Ai}, B's PUF output for C_{Bi}, C's PUF output for C_{Ci}
RN	Random number
T_{AB},T_{CB}	Secret sharing key between A and B, secret sharing key between C and B
${f^A}_{PUF}$, ${f^B}_{PUF}$, ${f^C}_{PUF}$	A's PUF function, B's PUF function, C's PUF function
$\oplus$	Bit-wise XOR operation
$hash()$	Cryptographic hash function

3.1. One-Time Enrollment Phase

This phase is run in a secure environment without the attacker. After completing this phase, the IoT nodes can be deployed in the IoT network. When any A wants to enroll in the server, the server randomly generates an input C_{Ai} (i.e., challenge) for A's PUF and collects its output R_{Ai} (i.e., response), where $R_{Ai} = {f^A}_{PUF}(C_{Ai})$. Then, the server stores A's ID-CRP $\{id_A, C_{Ai}, R_{Ai}\}$ in its secure database. The server only maintains one ID-CRP record for each IoT device.

3.2. Authentication and Key Exchange Phase

This phase has two communication models, i.e., Node-to-Node (N2N) communication and Node-to-Server (N2S) communication. N2N communication aims to provide the mutual authentication and key establishment of any two IoT nodes. Meanwhile, N2S communication realizes the mutual authentication and key establishment between any one IoT node and the server.

3.2.1. N2N Communication

Assume that two nearby A and B nodes want to authenticate each other and establish their session key. As shown in Figure 2, A, B, and the server perform the following steps.

Step 1. A sends a connection request $\{id_A\}$ to B. Upon receiving this request, B sends the N2N connection establishment request $\{id_A, id_B\}$ to the server.

Step 2. The server fetches corresponding ID-CRPs $\{id_A, C_{Ai}, R_{Ai}\}$ and $\{id_B, C_{Bi}, R_{Bi}\}$ from its secure database and generates two random numbers RN and T_{AB}. The server computes:

$$M_A \leftarrow R_{Ai} \oplus RN \tag{5}$$

$$M_B \leftarrow R_{Bi} \oplus RN \tag{6}$$

$$T'_{AB} \leftarrow T_{AB} \oplus RN \tag{7}$$

$$H_A \leftarrow hash(R_{Ai} \| M_A) \tag{8}$$

$$H_B \leftarrow hash(R_{Bi} \| M_B). \tag{9}$$

Then, the server sends the message $\{C_{Ai}, M_A, M_B, H_A, H_B, T'_{AB}\}$ to A and the message $\{C_{Bi}, M_A, M_B, H_A, H_B, T'_{AB}\}$ to B.

Step 3. Upon receiving the message $\{C_{Ai}, M_A, M_B, H_A, H_B, T'_{AB}\}$, A first computes:

$$R_{Ai} \leftarrow {f^A}_{PUF}(C_{Ai}) \tag{10}$$

$$RN \leftarrow R_{Ai} \oplus M_A \tag{11}$$

$$R_{Bi} \leftarrow M_B \oplus RN \tag{12}$$

$$H^*_A \leftarrow hash(R_{Ai} \| M_A) \tag{13}$$

$$H^*_B \leftarrow hash(R_{Bi} \| M_B). \tag{14}$$

To authenticate the server, A then checks whether H_A is equal to H^*_A and H_B is equal to H^*_B. If any verification fails, A terminates the session. Otherwise, A further calculates:

$$T_{AB} \leftarrow T'_{AB} \oplus RN \tag{15}$$

$$T_B \leftarrow T_{AB} \oplus R_{Bi}. \tag{16}$$

Finally, A sends $\{T_B\}$ to B.

Step 4. Upon receiving the messages $\{C_{Bi}, M_A, M_B, H_A, H_B, T'_{AB}\}$ and $\{T_B\}$, B computes:

$$R_{Bi} \leftarrow f^{\mathcal{B}}_{PUF}(C_{Bi}) \tag{17}$$

$$RN \leftarrow R_{Bi} \oplus M_B \tag{18}$$

$$R_{Ai} \leftarrow M_A \oplus RN \tag{19}$$

$$H^*_A \leftarrow hash(R_{Ai} \| M_A) \tag{20}$$

$$H^*_B \leftarrow hash(R_{Bi} \| M_B). \tag{21}$$

To authenticate the server, B checks H_A and H_B using H^*_A and H^*_B just like A. If any verification fails, B also terminates the session. Moreover, B evaluates:

$$T_{AB} \leftarrow T'_{AB} \oplus RN \tag{22}$$

$$R^*_{Bi} \leftarrow T_B \oplus T_{AB}. \tag{23}$$

Now, B checks whether R_{Bi} is equal to R^*_{Bi}. If the verification is incorrect, B terminates the session; otherwise, B computes:

$$T_A \leftarrow T_{AB} \oplus R_{Ai} \tag{24}$$

$$C_{B(i+1)} \leftarrow C_{Bi} \oplus R_{Bi} \tag{25}$$

$$R_{B(i+1)} \leftarrow f^{\mathcal{B}}_{PUF}(C_{B(i+1)}) \tag{26}$$

$$M_{SB} \leftarrow R_{B(i+1)} \oplus RN \tag{27}$$

$$H_{SB} \leftarrow hash(R_{B(i+1)} \| M_{SB}). \tag{28}$$

Finally, B sends $\{T_A\}$ to A and $\{M_{SB}, H_{SB}\}$ to the server.

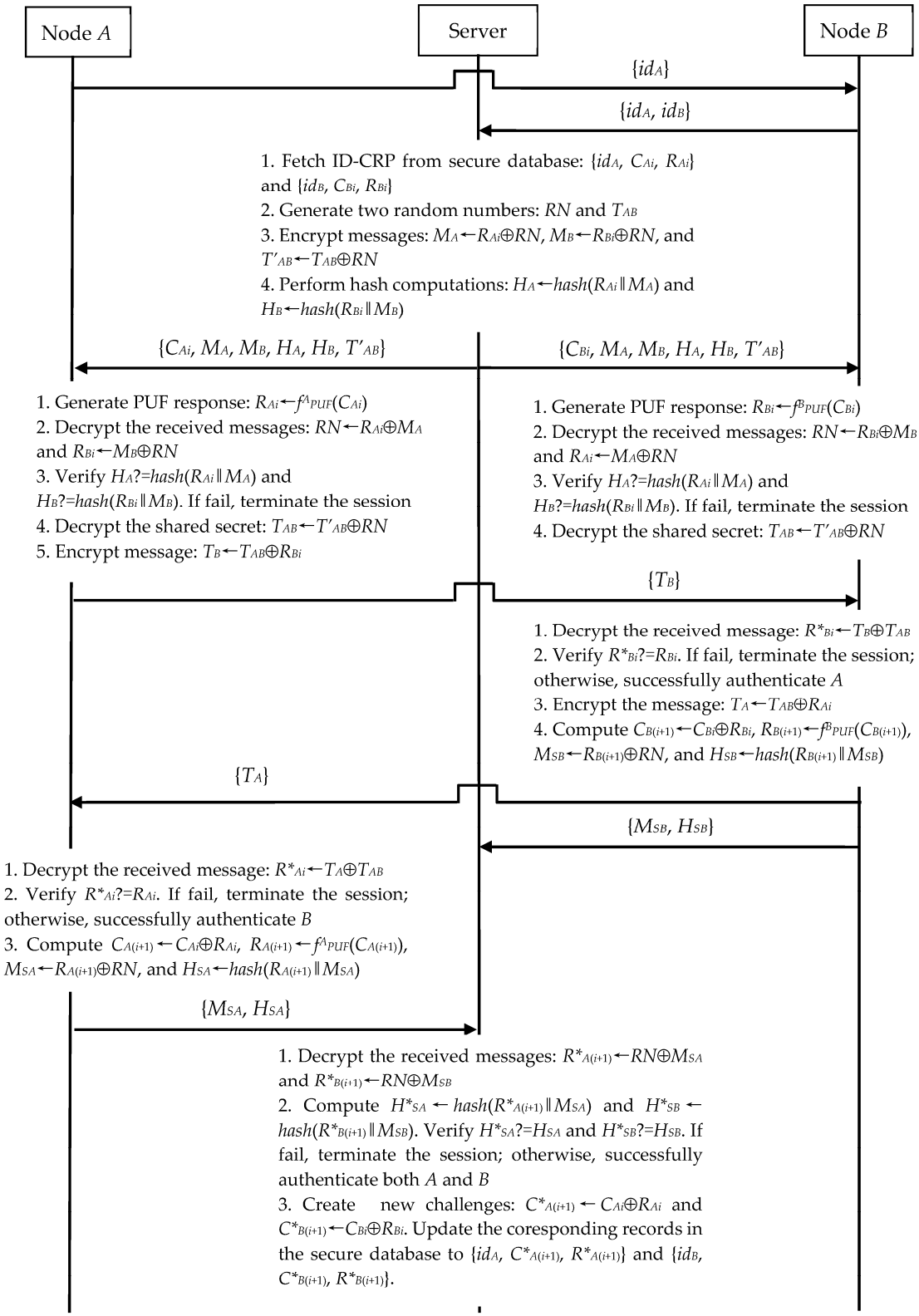


Figure 2. N2N's authentication and key exchange phase.

Step 5. Upon receiving the message $\{T_A\}$, A computes:

$$R^*_{Ai} \leftarrow T_A \oplus T_{AB}. \tag{29}$$

A checks whether R_{Ai} is equal to R^*_{Ai}. If the verification is incorrect, A terminates the session; otherwise, A computes:

$$C_{A(i+1)} \leftarrow C_{Ai} \oplus R_{Ai} \tag{30}$$

$$R_{A(i+1)} \leftarrow f^A{}_{PUF}(C_{A(i+1)}) \tag{31}$$

$$M_{SA} \leftarrow R_{A(i+1)} \oplus RN \tag{32}$$

$$H_{SA} \leftarrow hash(R_{A(i+1)} \| M_{SA}). \tag{33}$$

A then sends $\{M_{SA}, H_{SA}\}$ to the server.

Step 6. Upon receiving the messages $\{M_{SA}, H_{SA}\}$ and $\{M_{SB}, H_{SB}\}$, the server computes:

$$R^*_{A(i+1)} \leftarrow RN \oplus M_{SA} \tag{34}$$

$$R^*_{B(i+1)} \leftarrow RN \oplus M_{SB} \tag{35}$$

$$H^*_{SA} \leftarrow hash(R^*_{A(i+1)} \| M_{SA}) \tag{36}$$

$$H^*_{SB} \leftarrow hash(R^*_{B(i+1)} \| M_{SB}). \tag{37}$$

The server then checks whether H_{SA} is equal to H^*_{SA} and H_{SB} is equal to H^*_{SB}. If any verification is incorrect, the server terminates the session; otherwise, the server computes:

$$C^*_{A(i+1)} \leftarrow C_{Ai} \oplus R_{Ai} \tag{38}$$

$$C^*_{B(i+1)} \leftarrow C_{Bi} \oplus R_{Bi}. \tag{39}$$

Now, the server updates $\{id_A, C_{Ai}, R_{Ai}\}$ to $\{id_A, C^*_{A(i+1)}, R^*_{A(i+1)}\}$ and $\{id_B, C_{Bi}, R_{Bi}\}$ to $\{id_B, C^*_{B(i+1)}, R^*_{B(i+1)}\}$ in its secure database.

After the successful completion of above steps, T_{AB} is used as the secret session key for subsequent secure communication between A and B. Clearly, T_{AB} gets updated in every new session.

3.2.2. N2S Communication

Assume that B wants to authenticate the proximity server and establish a session key with the server. As shown in Figure 3, B and the server perform the following steps.

Step 1. B sends the N2S connection establishment request $\{id_B\}$ to the server.

Step 2. The server fetches the corresponding ID-CRP $\{id_B, C_{Bi}, R_{Bi}\}$ from its secure database and generates a random number RN. The server computes:

$$M_B \leftarrow R_{Bi} \oplus RN \tag{40}$$

$$H_B \leftarrow hash(R_{Bi} \| M_B). \tag{41}$$

The server then sends the message $\{C_{Bi}, M_B, H_B\}$ to B.

Step 3. Upon receiving the message $\{C_{Bi}, M_B, H_B\}$, B computes:

$$R_{Bi} \leftarrow f^B{}_{PUF}(C_{Bi}) \tag{42}$$

$$RN \leftarrow R_{Bi} \oplus M_B \tag{43}$$

$$H^*_B \leftarrow hash(R_{Bi} \| M_B). \tag{44}$$

To authenticate the server, B then checks whether H_B is equal to H^*_B. If any verification fails, B terminates the session. Otherwise, B further computes:

$$C_{B(i+1)} \leftarrow C_{Bi} \oplus R_{Bi} \tag{45}$$

$$R_{B(i+1)} \leftarrow f^B_{PUF}(C_{B(i+1)}) \tag{46}$$

$$M_{SB} \leftarrow R_{B(i+1)} \oplus RN \tag{47}$$

$$H_{SB} \leftarrow hash(R_{B(i+1)} \| M_{SB}). \tag{48}$$

Finally, B sends $\{M_{SB}, H_{SB}\}$ to the server.

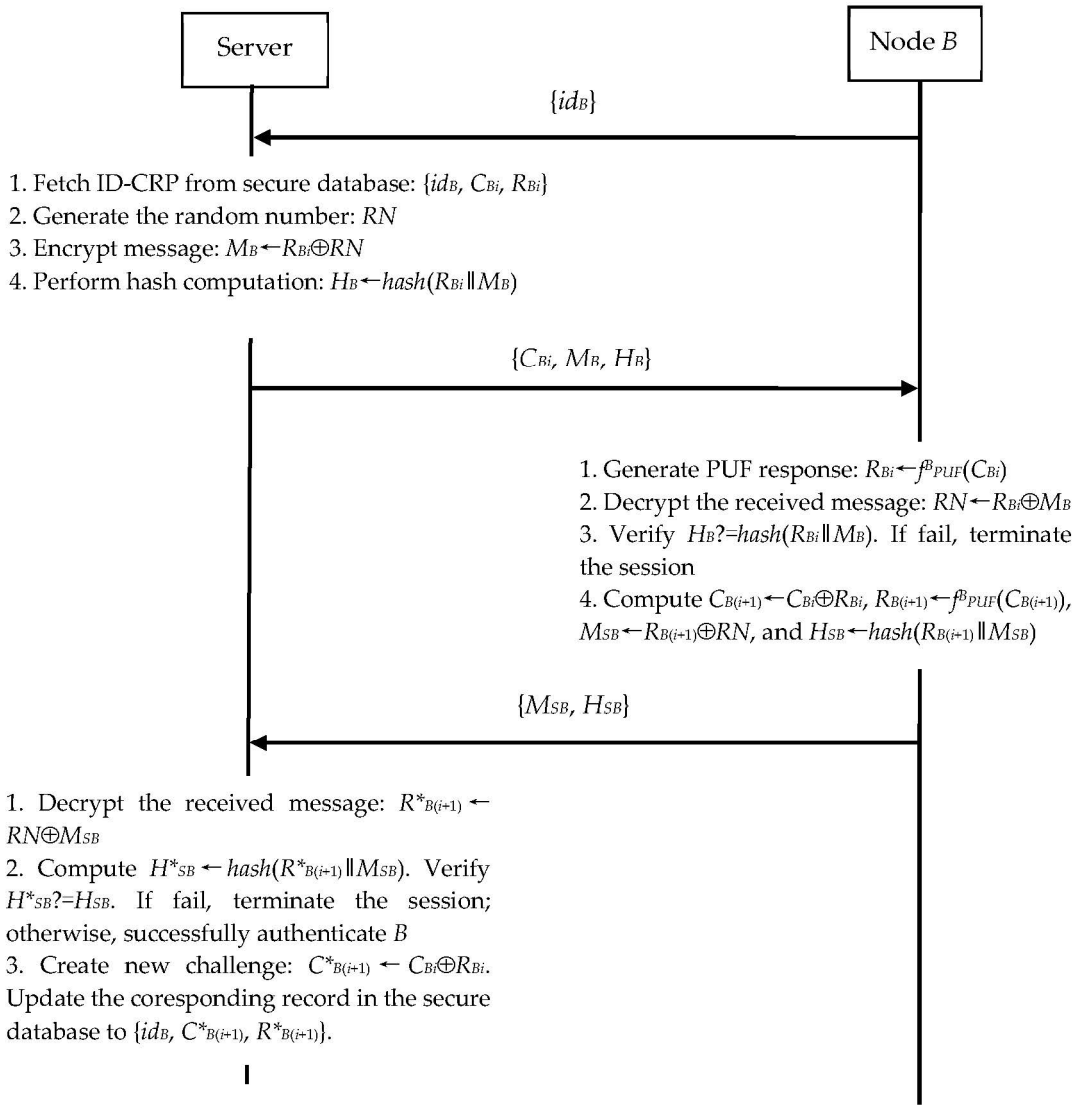


Figure 3. N2S's authentication and key exchange phase.

Step 4. Upon receiving the message $\{M_{SB}, H_{SB}\}$, the server computes:

$$R^*_{B(i+1)} \leftarrow RN \oplus M_{SB} \tag{49}$$

$$H^*_{SB} \leftarrow hash(R^*_{B(i+1)} \| M_{SB}). \tag{50}$$

The server then checks whether H_{SB} is equal to H^*_{SB}. If the verification is incorrect, the server terminates the session; otherwise, the server computes:

$$C^*_{B(i+1)} \leftarrow C_{Bi} \oplus R_{Bi}. \tag{51}$$

Now, the server updates $\{id_B, C_{Bi}, R_{Bi}\}$ to $\{id_B, C^*_{B(i+1)}, R^*_{B(i+1)}\}$ in its secure database.

Once the above authentication procedure is completed, B and the server share $R_{B(i+1)}$ as their secret session key. This key is updated in every new session establishment.

4. Insider Attack on Roy et al.'s Protocol

Let A be an insider attacker. It means that A has a legitimate ID-CRP record $\{id_A, C_{Ai}, R_{Ai}\}$ in the server's secure database but attempts to sabotage the server and other IoT nodes. When both A and B run the N2N communication model with the server, A executes the steps as described in Figure 2. At the same time, A as an insider further performs the following operations.

(1) A eavesdrops on B's $\{id_A, id_B\}$ during Step 1.
(2) A eavesdrops on the server's $\{C_{Bi}, M_A, M_B, H_A, H_B, T'_{AB}\}$ during Step 2.
(3) In Step 4, A eavesdrops on B's $\{M_{SB}, H_{SB}\}$.

Based on above, A can compute B's $C_{B(i+1)}$ by evaluating $C_{Bi} \oplus R_{Bi}$, because it knows R_{Bi} (see Equation (12)) during Step 3. And A can correctly recover B's $R_{B(i+1)}$ by calculating $M_{SB} \oplus RN$. Here, we know that A also computes the correct RN (see Equation (11)) during Step 3. Finally, we conclude that A obtains B's $\{id_B, C_{Bi}, R_{Bi}\}$ used in the next session, i.e., $\{id_B, C_{B(i+1)}, R_{B(i+1)}\}$.

4.1. Surveillance on IoT Nodes and Server

4.1.1. Surveillance Exploiting N2N Communication

To achieve authentication and establish the session key, B runs the N2N communication model with any C. As shown in Figure 4, A can eavesdrop on their session messages. And then, A further discloses the secret session key T_{CB} between C and B. Hence, A can monitor the subsequent secret channel using T_{CB}. For more detail, we describe A's behaviors on the session run of the N2N communication model between C and B.

(1) In Step 1, A eavesdrops on C's $\{id_c\}$.
(2) In Step 2, A eavesdrops on the server's $\{C_{Bi}, M_C, M_B, H_C, H_B, T'_{CB}\}$.

Now, A can obtain the RN by computing $R_{Bi} \oplus M_B$, where R_{Bi} is collected in A's previous session with B. A then recovers T_{CB} by computing $T'_{CB} \oplus RN$. Moreover, if A wants to monitor B's next session, it can further perform as follows.

In Step 4, A eavesdrops on B's $\{M_{SB}, H_{SB}\}$.

A can compute B's $C_{B(i+1)}$ by evaluating $C_{Bi} \oplus R_{Bi}$ and then recover B's $R_{B(i+1)}$ by calculating $M_{SB} \oplus RN$. In the end, A updates its $\{id_B, C_{Bi}, R_{Bi}\}$ to $\{id_B, C_{B(i+1)}, R_{B(i+1)}\}$ for B's next session. In addition, if A wants to monitor C's next session, it further performs in the following.

(1) In Step 2, A eavesdrops on the server's $\{C_{Ci}, M_C, M_B, H_C, H_B, T'_{CB}\}$.
(2) In Step 5, A eavesdrops on C's $\{M_{SC}, H_{SC}\}$.

In this situation, A computes R_{Ci} by using $M_C \oplus RN$ and $C_{C(i+1)}$ by using $C_{Ci} \oplus R_{Ci}$. A also recovers C's $R_{B(i+1)}$ by calculating $M_{SC} \oplus RN$. Finally, A can record $\{id_C, C_{C(i+1)}, R_{C(i+1)}\}$ for monitoring C's next session.

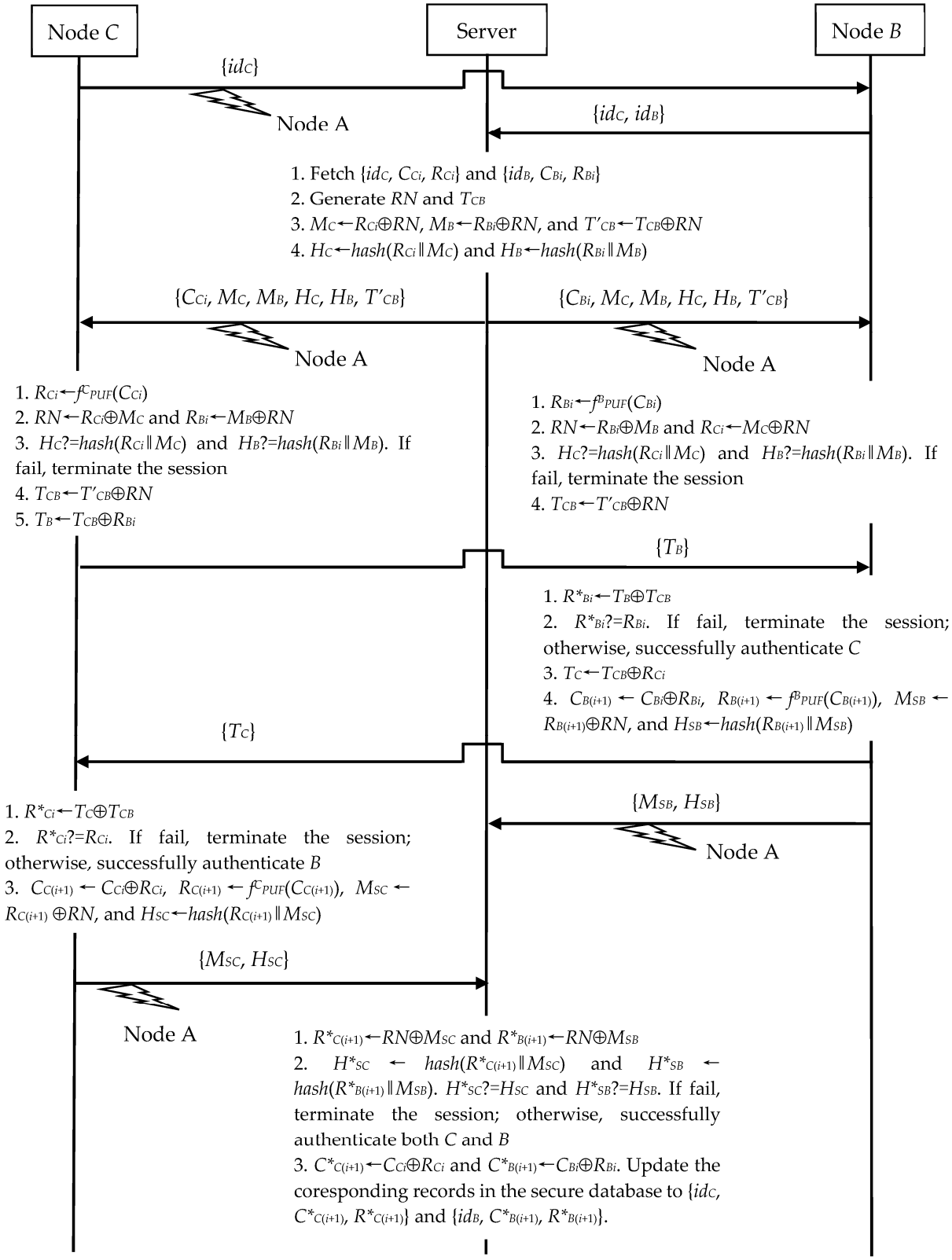


Figure 4. Eavesdropping on N2N's authentication and key exchange phase.

Comments. According to above surveillance, in the initial stage, the insider attacker needs to collect information about the target IoT node (i.e., *B*) by running a session with the target IoT node. Afterwards, the insider attacker can expand their surveillance to other IoT

nodes (e.g., *C*), which run the sessions with the target IoT node. Meanwhile, the insider attacker no longer needs to participate in those session runs and still obtains those secrets. This effectively hides the insider attacker and reduces the probability of the insider attacker being identified. Moreover, the insider attacker can establish a surveillance network of the selected IoT nodes based on our proposed attack.

4.1.2. Surveillance Exploiting N2S Communication

A is able to monitor the session run of the N2S communication model between *B* and the server. As shown in Figure 5, *A* can perform the following operations to reveal the secret session key $R_{B(i+1)}$ shared by *B* and the server.

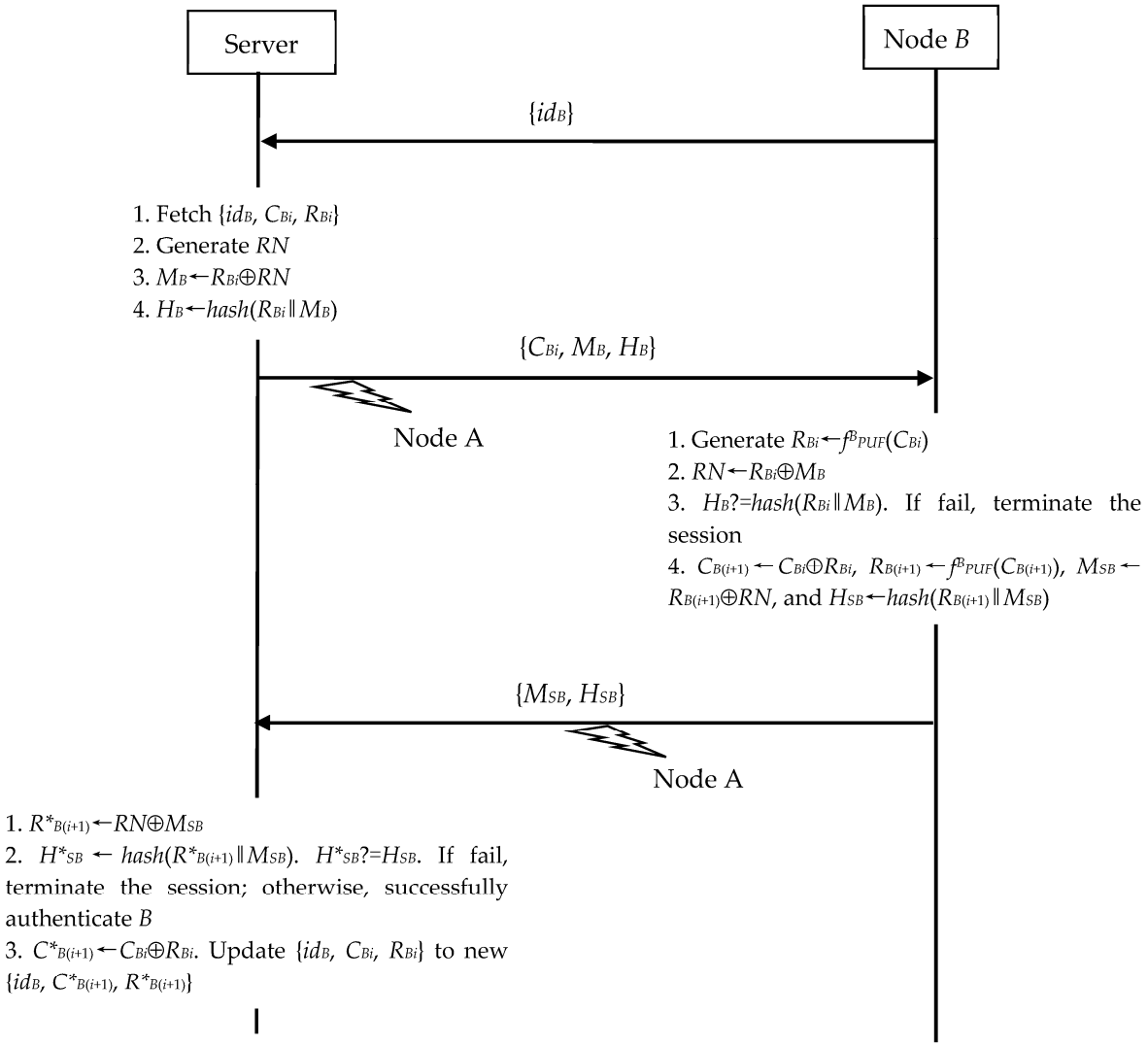


Figure 5. Eavesdropping on N2S's authentication and key exchange phase.

(1) In Step 2, *A* eavesdrops on the server's $\{C_{Bi}, M_B, H_B\}$.
(2) In Step 3, *A* eavesdrops on *B*'s $\{M_{SB}, H_{SB}\}$.

Because *A* knows *B*'s R_{Bi}, *A* can obtain the *RN* by computing $R_{Bi} \oplus M_B$. Now, *A* further recovers the secret session key $R_{B(i+1)}$ by computing $M_{SB} \oplus RN$. *A* can therefore observe the following secure communication using $R_{B(i+1)}$. Moreover, if *A* wants to continuously monitor the session run of the N2S communication model between *B* and the server, *A* merely computes $C_{B(i+1)} = C_{Bi} \oplus R_{Bi}$ and replaces the old $\{id_B, C_{Bi}, R_{Bi}\}$ with the new $\{id_B, C_{B(i+1)}, R_{B(i+1)}\}$.

4.2. Impersonating IoT Node

4.2.1. IoT Node Impersonation Exploiting N2N Communication

In N2N communication, A can imitate B run with any C, when A obtains a legal $\{id_B, C_{Bi}, R_{Bi}\}$. As shown in Figure 6, A replaces B to perform B's following steps during the run of N2N communication.

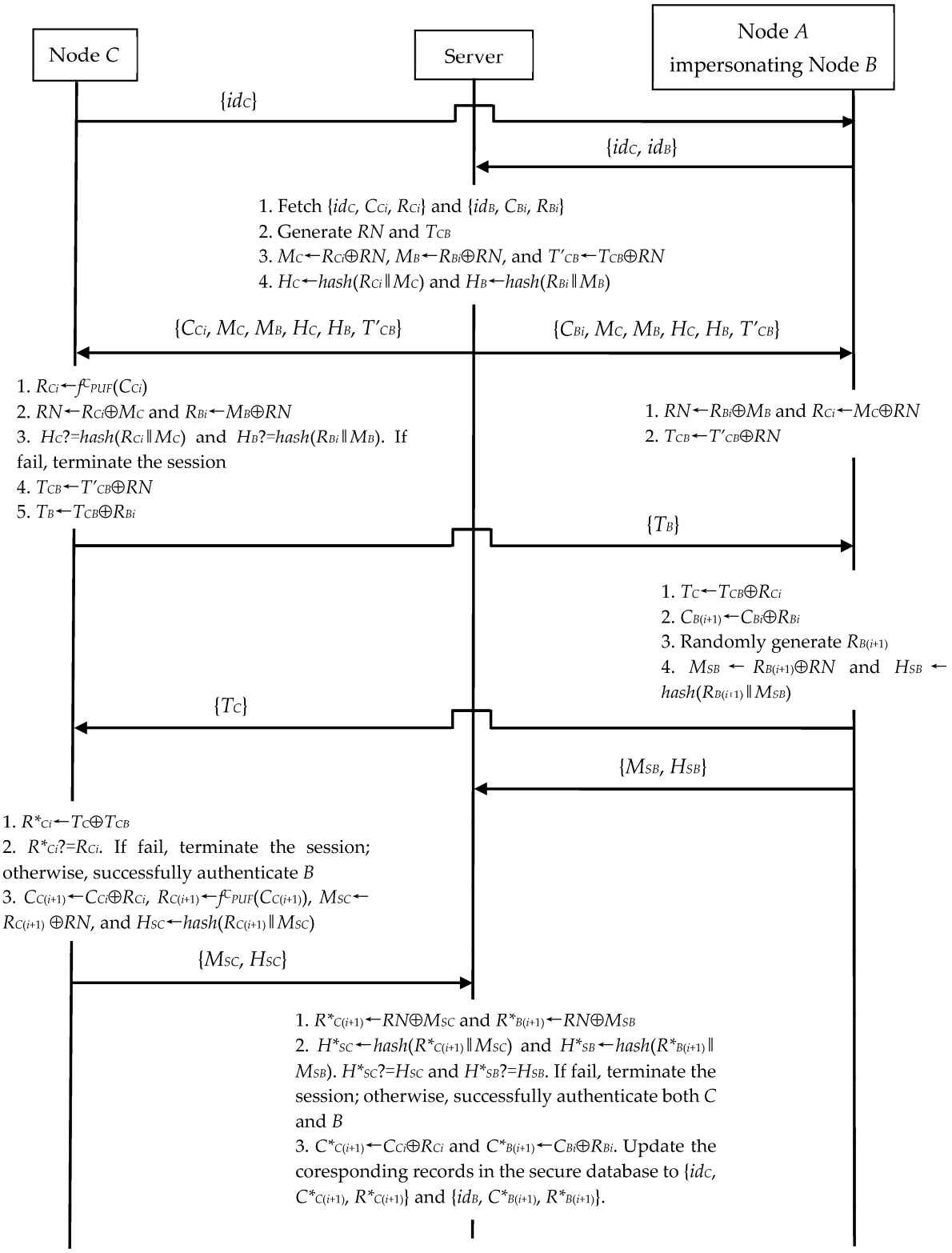


Figure 6. Node impersonation during N2N's authentication and key exchange phase.

(1) Upon receiving C's request $\{id_C\}$ in Step 1, A sends the N2N connection establishment request $\{id_C, id_B\}$ to the server.

(2) Upon receiving the messages $\{C_{Bi}, M_C, M_B, H_C, H_B, T'_{CB}\}$ and $\{T_B\}$ in Step 4, A computes $RN \leftarrow R_{Bi} \oplus M_B$, $R_{Ci} \leftarrow M_C \oplus RN$, $T_{CB} \leftarrow T'_{CB} \oplus RN$, $T_C \leftarrow T_{CB} \oplus R_{Ci}$, and $C_{B(i+1)} \leftarrow C_{Bi} \oplus R_{Bi}$. Then, A generates $R_{B(i+1)}$ at random. Next, A evaluates $M_{SB} \leftarrow R_{B(i+1)} \oplus RN$ and $H_{SB} \leftarrow hash(R_{B(i+1)} \| M_{SB})$. Finally, A sends $\{T_C\}$ to C and $\{M_{SB}, H_{SB}\}$ to the server.

Herein, A randomly generates $R_{B(i+1)}$ instead of B's $R_{B(i+1)}$ outputted by the valid PUF to evaluate M_{SB} and H_{SB} in Step 4. A can still pass the server's authentication, because the server never checks the PUF validness of its receiving $R_{B(i+1)}$ in Step 6. If A wants to continuously impersonate B, A further uses the new $\{id_B, C_{B(i+1)}, R_{B(i+1)}\}$ to replace the old $\{id_B, C_{Bi}, R_{Bi}\}$ for B's next session. In addition, A can also employ a similar attack to impersonate initiator C, if A obtains C's $\{id_C, C_{Ci}, R_{Ci}\}$.

Comments. After A completes above impersonation, B will no longer be able to successfully run the authentication and key exchange phase with the server and other IoT nodes. In this situation, we know that the server updates its $\{id_B, C_{Bi}, R_{Bi}\}$ to $\{id_B, C_{B(i+1)}, R_{B(i+1)}\}$, where $R_{B(i+1)}$ is randomly generated by A. When B runs the authentication and key exchange phase, the server should use the random R_{Bi} to generate M_B (see Equation (6)) and send it to B in Step 2. However, in Step 4, B retrieves its local R_{Bi} by invoking $f^B{}_{PUF}(C_{Bi})$ (see Equation (17)). Clearly, the server's R_{Bi} and B's R_{Bi} are always not equal because of the PUF's safety feature. Hence, B should recover an incorrect RN using Equation (18) and fail the subsequent authentication and key exchange procedure with the server and the counterpart IoT node. Therefore, B suffers from a denial of service attack due to A's impersonation.

4.2.2. IoT Node Impersonation Exploiting N2S Communication

In N2S communication, A can impersonate B to cheat the server, when A obtains B's $\{id_B, C_{Bi}, R_{Bi}\}$. As shown in Figure 7, we demonstrate A's operations as follows.

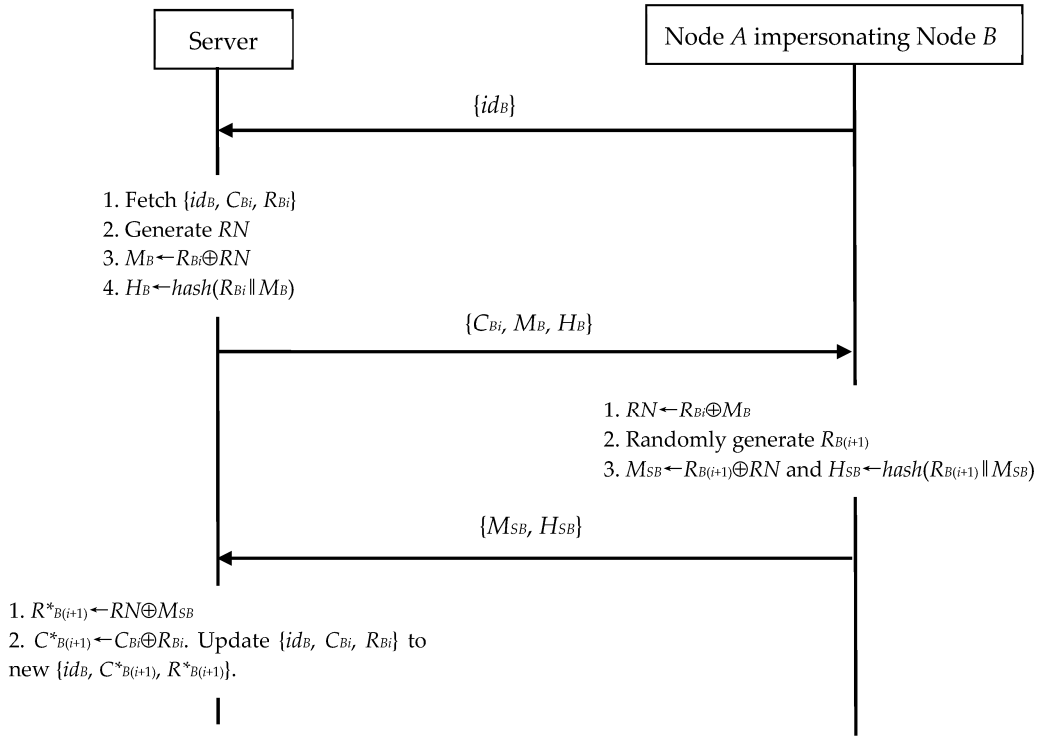


Figure 7. Node impersonation during N2S's authentication and key exchange phase.

(1) A sends the request $\{id_B\}$ to the server in Step 1.

(2) Upon receiving the message $\{C_{Bi}, M_B, H_B\}$ in Step 3, A computes $RN \leftarrow R_{Bi} \oplus M_B$. Then, A generates $R_{B(i+1)}$ at random. Next, A evaluates $M_{SB} \leftarrow R_{B(i+1)} \oplus RN$ and $H_{SB} \leftarrow hash(R_{B(i+1)} \| M_{SB})$. Finally, A sends $\{M_{SB}, H_{SB}\}$ to the server.

This node impersonation is similar to that of the impersonation discussed in Section 4.2.1. Clearly, the server should confirm the validity of A's H_{SB} in Step 4, and therefore we omit this operation in Figure 7 for simplicity. The slight difference is that, after session run, A applies $R_{B(i+1)}$ as the secret session key instead of T_{CB} as in Section 4.2.1.

4.3. Impersonating Server

4.3.1. Server Impersonation Exploiting N2N Communication

A can impersonate the server to cheat both C and B in N2N communication, if A obtains C's $\{id_C, C_{Ci}, R_{Ci}\}$ and B's $\{id_B, C_{Bi}, R_{Bi}\}$. Figure 8 shows the process of A's server impersonation. A performs the following operations.

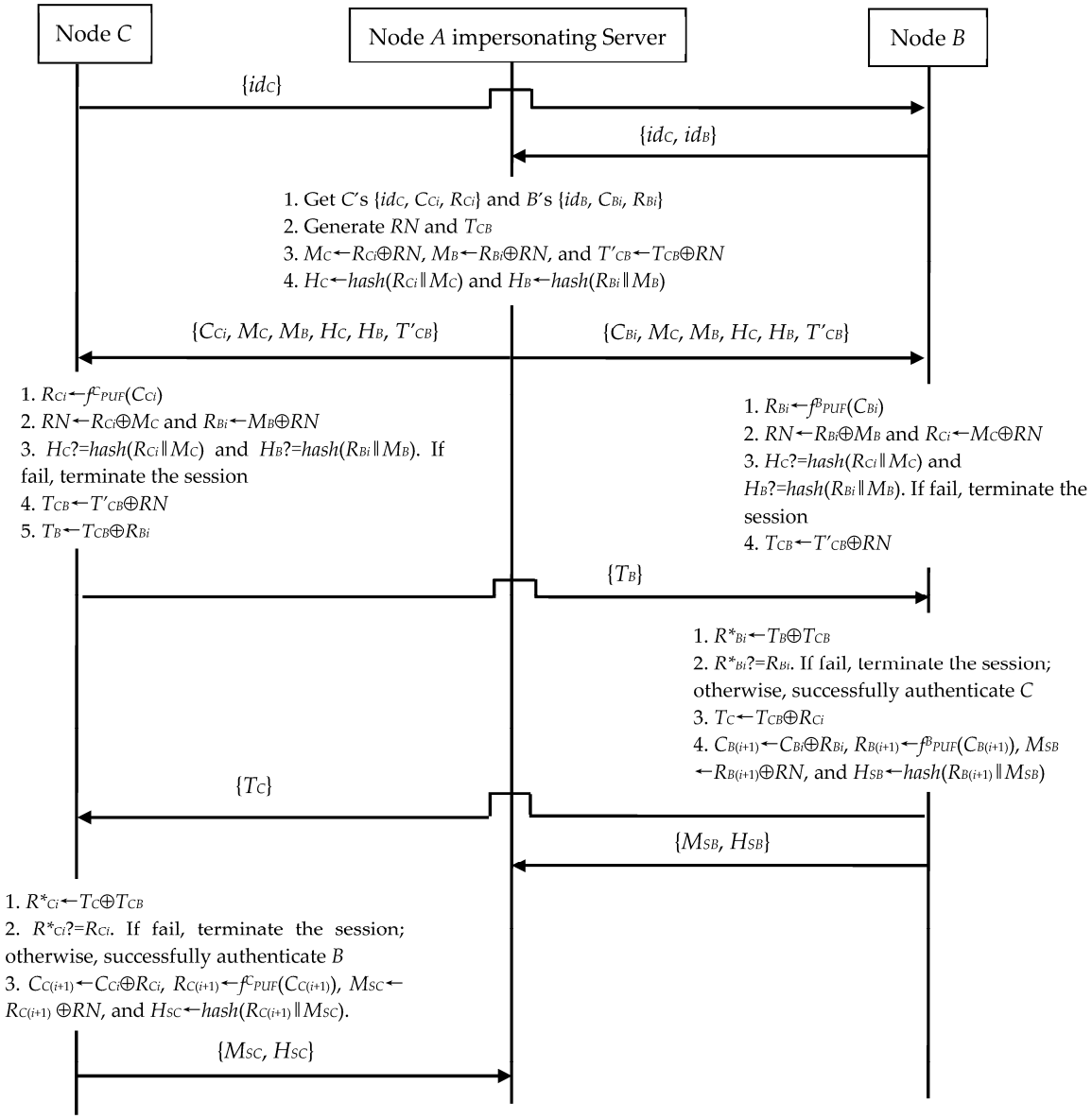


Figure 8. Server impersonation during N2N's authentication and key exchange phase.

(1) In Step 2, A randomly generates RN and T_{CB}. A further computes $M_C \leftarrow R_{Ci} \oplus RN$, $M_B \leftarrow R_{Bi} \oplus RN$, $T'_{CB} \leftarrow T_{CB} \oplus RN$, $H_C \leftarrow hash(R_{Ci} \| M_C)$, and $H_B \leftarrow hash(R_{Bi} \| M_B)$. Then, A

sends the message $\{C_{Ci}, M_C, M_B, H_C\ H_B, T'_{CB}\}$ to C and the message $\{C_{Bi}, M_C, M_B, H_C, H_B, T'_{CB}\}$ to B.

(2) Upon receiving the messages $\{M_{SC}, H_{SC}\}$ and $\{M_{SB}, H_{SB}\}$ in Step 6, A omits them.

A finally shares the session key T_{CB} with both C and B. Moreover, A still can reuse C's $\{id_C, C_{Ci}, R_{Ci}\}$ and B's $\{id_B, C_{Bi}, R_{Bi}\}$ to impersonate the server in the subsequent session, because C and B do not verify the freshness of C_{Ci} and C_{Bi} in each session run. Certainly, in Step 6, A can also compute $R^*_{C(i+1)} \leftarrow RN \oplus M_{SC}$, $R^*_{B(i+1)} \leftarrow RN \oplus M_{SB}$, $C^*_{C(i+1)} \leftarrow C_{Ci} \oplus R_{Ci}$, and $C^*_{B(i+1)} \leftarrow C_{Bi} \oplus R_{Bi}$. This means that A obtains new ID-CRPs for future attacks.

4.3.2. Server Impersonation Exploiting N2S Communication

When B wants to run N2S communication, A can make use of B's $\{id_B, C_{Bi}, R_{Bi}\}$ to impersonate the server. As shown in Figure 9, A executes the following operations to achieve it.

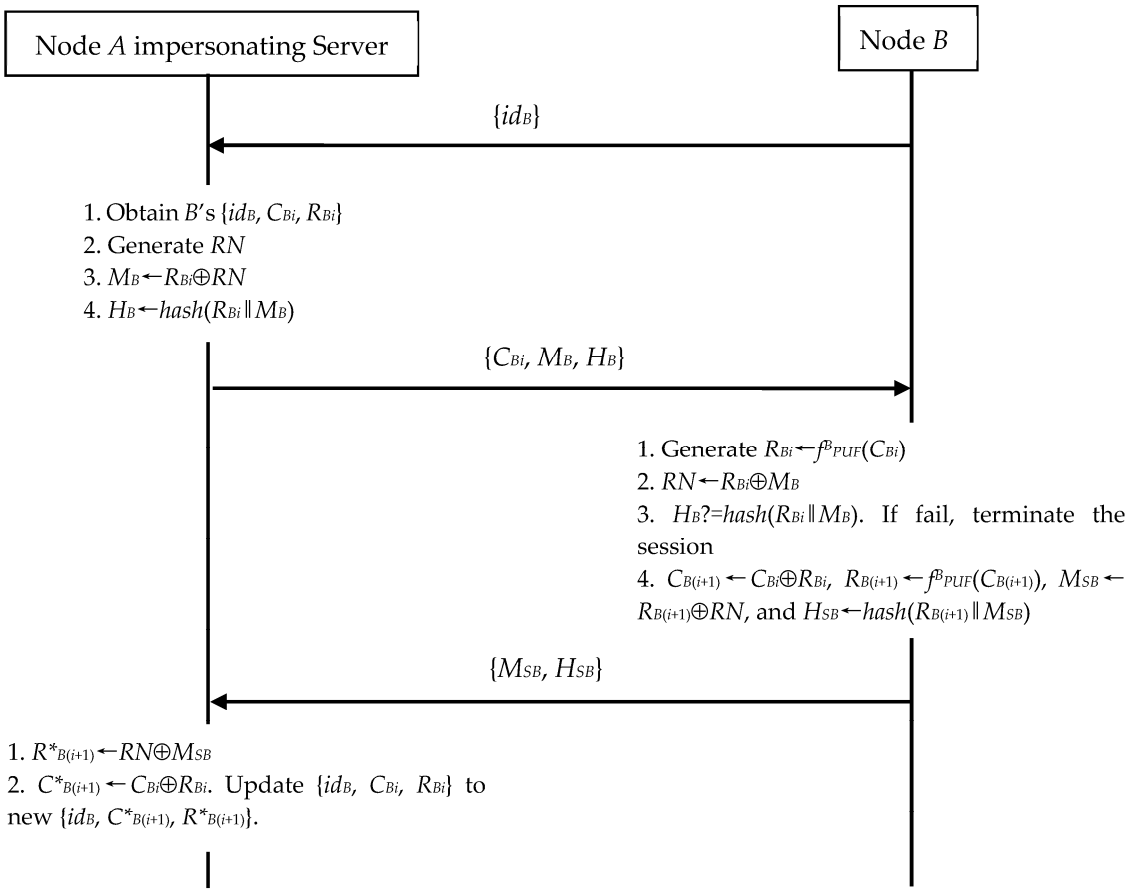


Figure 9. Server impersonation during N2S's authentication and key exchange phase.

(1) In Step 2, A generates the random number RN. A computes $M_B \leftarrow R_{Bi} \oplus RN$ and $H_B \leftarrow hash(R_{Bi} || M_B)$. Then, A sends the message $\{C_{Bi}, M_B, H_B\}$ to B.

(2) Upon receiving the message $\{M_{SB}, H_{SB}\}$ in Step 4, A computes $R^*_{B(i+1)} \leftarrow RN \oplus M_{SB}$ and $C^*_{B(i+1)} \leftarrow C_{Bi} \oplus R_{Bi}$ and further replaces its $\{id_B, C_{Bi}, R_{Bi}\}$ with $\{id_B, C^*_{B(i+1)}, R^*_{B(i+1)}\}$.

As a result, A and B establish the secret session key $R^*_{B(i+1)}$. A can omit to replace its $\{id_B, C_{Bi}, R_{Bi}\}$ with $\{id_B, C^*_{B(i+1)}, R^*_{B(i+1)}\}$ and still use $\{id_B, C_{Bi}, R_{Bi}\}$ for the subsequent impersonation. The reason is the same as in our discussion in Section 4.3.1.

4.4. Discussion of Insider Attacker

We discuss how an IoT node practically becomes a malicious IoT node, i.e., an insider attacker. In general, the IoT system contains plenty of IoT nodes. And some of them are possibly manufactured and provided by third parties. From a third-party perspective, these IoT nodes may not only want to work properly in the IoT system, but also engage in malicious behaviors such as surveillance and impersonation. Clearly, in this situation, the malicious IoT nodes can enter the IoT system during the one-time enrollment phase of Roy et al.'s protocol.

After the deployment of the IoT nodes, physical attacks are very common for the IoT system. They typically require physical proximity to the IoT system and can involve actions that limit the efficacy of the IoT nodes. In order to turn benign IoT nodes into malicious IoT nodes, the physical attack can further inject malicious codes into benign IoT nodes. These malicious codes include the logic that requires the IoT nodes to execute insider attacks in some cases. In addition, software attacks can be exploited to compromise the IoT nodes, that is, the attackers use the malware, such as viruses, worms, and Trojans, to manipulate the IoT nodes.

It is possible that malicious IoT nodes have limited resources and functionalities. This means that the malicious IoT nodes cannot handle and store the derived session keys for next session and for another IoT node, cannot intercept the transmitted messages over the public channel, cannot impersonate the server, and so on. To complete our insider attacks, the attacker can build a powerful auxiliary node to support malicious IoT nodes. Each malicious IoT node A merely outputs its $\{id_A, C_{Ai}, R_{Ai}\}$ to the auxiliary node. The auxiliary node can replace the malicious IoT node A to implement the insider attacks.

5. Experimental Verification of Proposed Insider Attacks

Scyther is a formal analysis tool for automatic verification of security protocols based on the security protocol description language (SPDL). Scyther can analyze protocols that contain multiple subjects, infinite session interaction, and the use of random numbers. We therefore use Scyther to confirm our insider attacks on Roy et al.'s protocol. We implement experimental verification on the 64-bit Windows 10 operating system, Graphviz v2.50, and Python v2.7 using the Compile-0.9.2 version of the Scyther tool.

When using the Scyther tool to verify Roy et al.'s protocol, we write secret PUF responses, XOR-encrypted messages, etc., into the SPDL script to model the protocol. The details of the Scyther tool parameter settings are shown in Table 2. Here, the attacker can obtain the secret PUF responses and furthermore derive the session key and random numbers. This is consistent with the capabilities of the malicious IoT nodes assumed in Section 4.

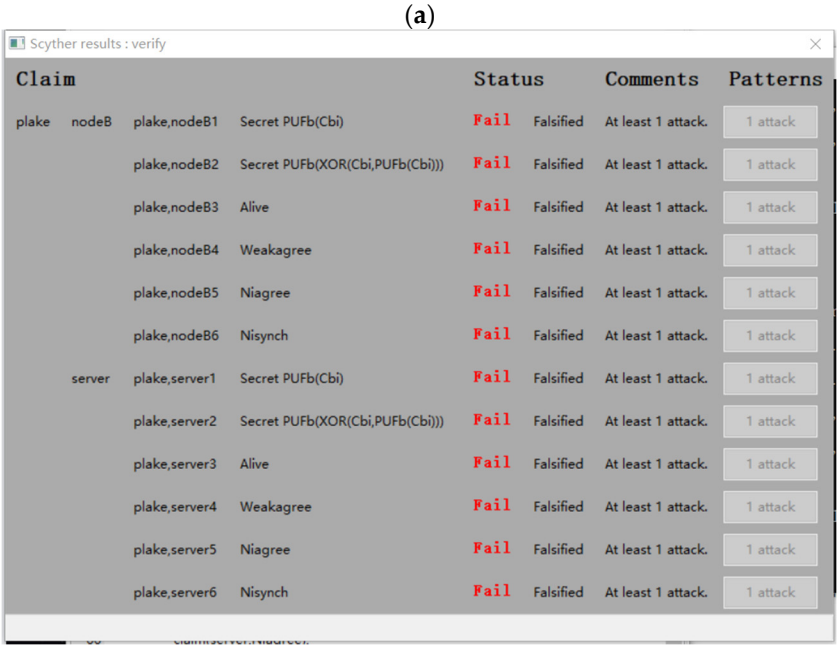
Table 2. Scyther tool parameters used for our analysis.

Parameters	Parameter Specification
Max. number of runs	5
Matching type	Typed matching
Search pruning	Find best attack
Max. patterns/claim	10
Long-term key reveal	None
Long-term key reveal after claim	None(DY)
Session key reveal	Checked
Random reveal	Checked
State reveal	None

For Roy et al.'s protocol, Figure 10 shows Scyther's verification results for N2N communication (see Figure 10a) and N2S communication (see Figure 10b). The results show that both N2N communication and N2S communication do not meet Scyther's automatic declaration requirements of Alive, Weakagree (weak agree), Nisynch (noninjective synchro-

nization), and Niagree (noninjective agreement). Moreover, the results also indicate that the secret session key T_{AB} in the N2N communication and the secret session key $R_{B(i+1)}$ in the N2S communication are both insecure. Hence, we conclude that the process of the authentication and key exchange in Roy et al.'s protocol is insecure and is subject to insider attacks on the IoT nodes and the server.

(**a**)



(**b**)

Figure 10. Scyther outputs for Roy et al.'s protocol. (**a**) N2N communication; (**b**) N2S communication.

6. Suggestion for Preventing Insider Attacks

We know that A is required to recover B's R_{Bi} (see Equation (12)) to verify B (see Equation (14)) in the N2N communication of Roy et al.'s protocol. Similarly, B needs to recovering A's R_{Ai} (see Equation (19)) to verify A (see Equation (20)). However, we know that both R_{Ai} and R_{Bi} are the secrets of A and B, respectively. It means that A and B obtain each other secrets after Step 4 of the N2N communication, which incurs the secure breach. In fact, these verifications conducted by A and B are unnecessary because both A and B believe the server. Therefore, one suggestion is to cancel the verifications, that is, the server does not send M_B and H_B to A and M_A and H_A to B during Step 2 of the N2N communication.

Moreover, we find that both A and B share a random number RN in the N2N communication. RN is used to encrypt their all secrets. Because of sharing the RN, the malicious IoT node can decrypt other IoT node's secrets. Hence, another suggestion is that the server must randomly select two random numbers instead of just one RN, that is, one random number for A and the other one for B.

The above two suggestions may defend against our proposed insider attacks. However, accurate security results for improvements require formal security models, security assumptions, security definitions, and reductions. We keep these for future work.

7. Conclusions

In the N2N communication of Roy et al.'s protocol, the IoT node requires the counterpart IoT node's secret output of the PUF to verify it. More importantly, the malicious IoT node can derive the counterpart IoT node's next output of the PUF if it eavesdrops on the message transmitted to the server. These defects have led to our insider attacks on Roy et al.'s protocol. Moreover, in the cases of the N2N communication and the N2S communication, we must point out that once the attacker stole the session keys, i.e., T_{AB} or $R_{B(i+1)}$, he can always impersonate not only the IoT node to cheat other IoT nodes or the server but also the server to cheat the IoT nodes in the subsequent sessions. The secure communications between the server and the corresponding IoT nodes are also possibly under the attacker's surveillance. This means that Roy et al.'s protocol fails to provide the known key security. Our security analysis results indicate that designing PUF-based authentication and key exchange protocols for IoT remains a challenging task.

Author Contributions: Conceptualization, D.-Z.S.; methodology, D.-Z.S.; validation, D.-Z.S. and Y.T.; formal analysis, D.-Z.S. and Y.-N.G.; investigation, D.-Z.S.; writing—original draft preparation, D.-Z.S. and Y.-N.G.; writing—review and editing, D.-Z.S. and Y.T.; supervision, D.-Z.S.; funding acquisition, D.-Z.S. All authors have read and agreed to the published version of the manuscript.

Funding: The work of Da-Zhi Sun was supported in part by the National Natural Science Foundation of China under Grant No. 61872264. The APC was funded by the National Natural Science Foundation of China under Grant No. 61872264.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data was created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: The authors would like to thank the editors and the reviewers for their valuable suggestions and comments.

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

References

1. Idriss, T.; Idriss, H.; Bayoumi, M. A lightweight PUF-based authentication protocol using secret pattern recognition for constrained IoT devices. *IEEE Access* **2021**, *9*, 80546–80558. [CrossRef]
2. Mukhopadhyay, D.; Chakraborty, R.S.; Nguyen, P.H.; Sahoo, D.P. Physically Unclonable Function: A Promising Security Primitive for Internet of Things. In Proceedings of the 2015 IEEE 28th International Conference on VLSI Design (VLSID), Bangalore, India, 3–7 January 2015; IEEE Computer Society: Los Alamitos, CA, USA, 2015; pp. 14–15.
3. Goncu, E.; Yalcin, M.E. A design of cellular automata-based PUF and its implementation on FPGA. *Int. J. Circuit Theory Appl.* **2020**, *48*, 1244–1255. [CrossRef]
4. Yoon, S.; Kim, B.; Kang, Y.; Choi, D. Security Enhancement for IoT Device Using Physical Unclonable Functions. In Proceedings of the 10th International Conference on Information and Communication Technology Convergence (ICTC)-ICT Convergence Leading the Autonomous Future, Jeju, Republic of Korea, 16–18 October 2019; IEEE: New York, NY, USA, 2019; pp. 1457–1459.
5. Yilmaz, Y.; Gunn, S.R.; Halak, B. Lightweight PUF-Based Authentication Protocol for IoT Devices. In Proceedings of the 3rd IEEE International Verification and Security Workshop (IVSW), Catalonia, Spain, 2–4 July 2018; IEEE: New York, NY, USA, 2018; pp. 38–43.
6. Chatterjee, U.; Govindan, V.; Sadhukhan, R.; Mukhopadhyay, D.; Chakraborty, R.S.; Mahata, D.; Prabhu, M.M. Building PUF based authentication and key exchange protocol for IoT without explicit CRPs in verifier database. *IEEE Trans. Dependable Secur. Comput.* **2019**, *16*, 16424–16437. [CrossRef]
7. Chuang, Y.H.; Lei, C.L. PUF based authenticated key exchange protocol for IoT without verifiers and explicit CRPs. *IEEE Access* **2021**, *9*, 112733–112743. [CrossRef]
8. Li, S.N.; Zhang, T.K.; Yu, B.; He, K. A provably secure and practical PUF-based end-to-end mutual authentication and key exchange protocol for IoT. *Sensors* **2021**, *21*, 5487–5501. [CrossRef]
9. Chaterjee, U.; Mukhopadhyay, D.; Chakraborty, R.S. 3PAA: A private PUF protocol for anonymous authentication. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 756–769. [CrossRef]
10. Siddiqui, Z.; Gao, J.C.; Khan, M.K. An improved lightweight PUF-PKI digital certificate authentication scheme for the internet of things. *IEEE Internet Things J.* **2022**, *9*, 19744–19756. [CrossRef]
11. Harishma, B.; Mathew, P.; Patranabis, S.; Chatterjee, U.; Agarwal, U.; Maheshwari, M.; Dey, S.; Mukhopadhyay, D. Safe is the new smart: PUF-based authentication for load modification-resistant smart meters. *IEEE Trans. Dependable Secur. Comput.* **2022**, *19*, 663–680. [CrossRef]
12. Qureshi, M.A.; Munir, A. PUF-RAKE: A PUF-based robust and lightweight authentication and key establishment protocol. *IEEE Trans. Dependable Secur. Comput.* **2022**, *19*, 2457–2475. [CrossRef]
13. Aman, M.N.; Chua, K.C.; Sikdar, B. Mutual authentication in IoT systems using physical unclonable functions. *IEEE Internet Things J.* **2017**, *4*, 1327–1340. [CrossRef]
14. Roy, S.; Das, D.; Mondal, A.; Mahalat, M.H.; Roy, S.; Sen, B. PUF Based Lightweight Authentication and Key Exchange Protocol for IoT. In Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT), Online, 6–8 July 2021; pp. 698–703.
15. Qureshi, M.A.; Munir, A. PUF-IPA: A PUF-Based Identity Preserving Protocol for Internet of Things Authentication. In Proceedings of the IEEE 17th Annual Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2020; pp. 1–7.
16. Lounis, K.; Zulkernine, M. T2T-MAP: A PUF-based thing-to-thing mutual authentication protocol for IoT. *IEEE Access* **2021**, *9*, 137384–137405. [CrossRef]
17. Nimmy, K.; Sankaran, S.; Achuthan, K. A novel lightweight PUF based authentication protocol for IoT without explicit CRPs in verifier database. *J. Ambient Intell. Humaniz. Comput.* **2023**, *14*, 6227–6242. [CrossRef]
18. Zheng, Y.; Chang, C.H. Secure Mutual Authentication and Key-Exchange Protocol between PUF-Embedded IoT Endpoints. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Republic of Korea, 22–28 May 2021; IEEE: New York, NY, USA, 2021; pp. 1–5.
19. Wang, Z.Y.; Li, S.Q.; Guo, Y.; Zeng, J.P. A lightweight key sharing protocol for IoT devices based on PUF. *J. Hunan Univ. Nat. Sci.* **2022**, *49*, 103–110.
20. Ebrahimabadi, M.; Younis, M.; Karimi, N. A PUF-based modeling-attack resilient authentication protocol for IoT devices. *IEEE Internet Things J.* **2022**, *9*, 3684–3703. [CrossRef]
21. Zerrouki, F.; Ouchani, S.; Bouarfa, H. PUF-based mutual authentication and session key establishment protocol for IoT devices. *J. Ambient Intell. Humaniz. Comput.* **2022**. *early access*. [CrossRef]
22. Sun, D.Z.; Tian, Y. Security of a PUF mutual authentication and session key establishment protocol for IoT devices. *Mathematics* **2022**, *10*, 4310. [CrossRef]
23. Wang, H.Y.; Meng, J.; Du, X.L.; Cao, T.F.; Xie, Y. Lightweight and anonymous mutual authentication protocol for edge IoT nodes with physical unclonable function. *Secur. Commun. Netw.* **2022**, *2022*, 1203691. [CrossRef]
24. Park, K.; Park, Y. IAKA-CIOT: An improved authentication and key agreement scheme for cloud enabled internet of things using physical unclonable function. *Sensors* **2022**, *22*, 6264. [CrossRef]
25. Aseeri, A.O.; Chauhdary, S.H.; Alkatheiri, M.S.; Alqarni, M.A.; Zhuang, Y. Application of physical unclonable function for lightweight authentication in internet of things. *Comput. Mater. Contin.* **2023**, *75*, 1901–1918. [CrossRef]

26. Prosanta, G.; Biplab, S. Lightweight and privacy-preserving two-factor authentication scheme for IoT devices. *IEEE Internet Things J.* **2019**, *6*, 6580–6589.
27. Mostafa, A.; Lee, S.J.; Peker, Y.K. Physical unclonable function and hashing are all you need to mutually authenticate IoT devices. *Sensors* **2020**, *20*, 4361. [CrossRef]
28. Uysal, E.; Akgun, M. P/Key: PUF based second factor authentication. *PLoS ONE* **2023**, *18*, e0280181. [CrossRef]
29. Roy, S.; Das, D.; Mondal, A.; Mahalat, M.H.; Sen, B.; Sikdar, B. PLAKE: PUF based secure lightweight authentication and key exchange protocol for IoT. *IEEE Internet Things J.* **2023**, *10*, 8547–8559. [CrossRef]