Ultralightweight RFID Reader-Tag Mutual Authentication Revisited

Yu-Chung Huang
Department of Computer Science and Information Engineering, National Central University
Jhongli City, Taiwan, R.O.C.
985402024@cc.ncu.edu.tw

Jehn-Ruey Jiang
Department of Computer Science and Information Engineering, National Central University
Jhongli City, Taiwan, R.O.C.
jrjiang@csie.ncu.edu.tw

***Abstract*—The RFID (Radio Frequency Identification) technology plays an important role of providing mobile services in Internet of Things (IoT) environments. In an RFID (Radio Frequency Identification) system, a tag with a unique ID is attached to an object and a reader can recognize the object by identifying the attached tag. With this identified tag ID, the reader can then retrieve the related information of the object from the backend server database and even access IoT-aware services associated with the object. Due to the nature of RF signals, the communication between the reader and tags is vulnerable to attacks. Typical attacks include the man-in-the-middle (MitM), replay, forward secrecy, denial of service (DoS), and impersonation attacks. Due to the extremely small memory and very limited computation power of tags, some RFID reader-tag mutual authentication schemes, like Huang and Jiang's scheme, Yi et al.'s scheme and Khedr's scheme, have been proposed to resist these attacks by using on-tag ultralightweight operations, such as the random number generation (RNG), the pseudo random number generator (PRNG), the cyclic redundancy check (CRC), the exclusive-or (XOR), and lightweight cryptographic hash function (LHash) operations. These schemes still have some flaws, though. This paper proposes an improved mutual authentication scheme using only ultralightweight operations to resist more attacks and/or achieve lower overheads in terms of communication, computation, storage occupancy and data updating.**

Keywords-Radio Frequency Identification (RFID); Internet of Things (IoT); hash; security; privacy; mutual authentication

I. INTRODUCTION

The RFID (Radio Frequency Identification) technology plays an important role of providing mobile services in Internet of Things (IoT) environments [1]. It is integrated into many kinds of mobile devices, such as smartphones, to endow them with the capability to access and manipulate objects in the physical world. RFID systems have attracted much attention and have been utilized in many applications, such as logistic control, supply chain management, asset tracking. An RFID system consists of tags, a reader and a backend server [2]. A tag with a unique ID, such as the Electronic Product Code (EPC), is usually attached to an object, and the reader can recognize the object by initiating the identification procedure (or interrogation procedure) to identify the tag ID through wireless communications between the reader and tags. With this identified tag ID, the related information of the object can be retrieved from the backend server database, and even IoT-aware services associated with the object can then be obtained.

In the identification procedure, a reader issues RF signals to command tags to respond with their IDs. Due to the nature of wireless communications, the identification procedure is susceptible to various latent attacks, such as the man-in-the-middle (MitM), replay, forward secrecy, denial of service (DoS) and impersonation attacks [3-5]. In most wireless applications, such attacks can be easily resisted by applying general cryptographic operations. However, RFID tags, such as the famous EPCglobal Class 1 Generation 2 (Gen2) tags [6], are usually very cheap and thus have extremely small memory and very limited computation power [6]. They cannot afford to run general cryptographic operations [7-8] and can run only ultralightweight operations, such as the random number generation (RNG), pseudo random number generator (PRNG), cyclic redundancy check (CRC), exclusive-or (XOR), and lightweight cryptographic hash function (LHash) [9] operations.

The RNG, PRNG, CRC and XOR operations are supported by common RFID tags, such as Gen2 tags. Among the operations, the PRNG operation is very useful, since it can play the role of a cryptographic one-way hash function, on which many RFID security schemes depend. However, the LHash operation, such as the QUARK lightweight hash function recently proposed in [9], consumes little memory and energy to run. It can then replace the PRNG operation and be used to construct security schemes for RFID systems.

Several RFID reader-tag mutual authentication schemes [10-14] have been proposed to resist attacks for RFID systems. By registering tags and readers in the backend server database, they allow a tag and a reader to authenticate each other. Some [10-11] of the schemes use heavy-weight operations on tags; they are thus unsuitable for low cost RFID tags. The other schemes [12-14] use only ultralightweight operations on tags; they can therefore be applied to low cost tags. Unfortunately, these ultralightweight schemes still suffer from security weaknesses and have high communication and/or computation overheads. This motivates us to design a low-overhead ultralightweight mutual authentication scheme to raise the security level of RFID systems.

This paper proposes an RFID reader-tag mutual authentication scheme using only ultralightweight operations,

namely the RNG, XOR, and LHash operations. As we will show, it nevertheless can resist all the aforementioned attacks. The proposed scheme is also compared with other related schemes to demonstrate its superiority in terms of the communication cost, the computation cost, and security.

The remainder of the paper is organized as follows. Some mutual authentication schemes are introduced in Section II. The proposed scheme is detailed in Section III. Security analysis and comparisons are presented in Section IV and Section V, respectively. Finally, some concluding remarks are drawn in Section VI.

II. Related Work

Many schemes [12-14] have been proposed to mitigate the security threats mentioned in Section I with the assumption that RFID tags have limited memory and computation power. These schemes thus use only ultralightweight operations, such as PRNG, CRC, and XOR that are suitable for low cost tags. Below, we describe in detail three of these schemes, namely, Huang and Jiang's scheme [12], Yi et al.'s scheme [13], Khedr's scheme [14], which are most related to our proposed scheme. Below in this paper, we use tag_i and $reader_j$ to denote the tag and the reader involved in the scheme. We also use $X \stackrel{?}{=} Y$ to denote a comparison (verification) function that verifies whether X equals (or matches) Y, where X and Y are values or expressions.

A. Huang and Jiang's Scheme

We first describe the registration steps of Huang and Jiang's scheme [12]. Initially, the server sends (EPC_i, N_i, K_i, PID_i) to tag_i and stores (EPC_i, N_i^{old}, K_i^{old}, PID_i^{old}, N_i^{new}, K_i^{new}, PID_i^{new}) in the database to register tag_i, where EPC_i is the EPC number, N_i is the communication key, K_i is the authentication key, and PID_i is the pseudonym (pseudo identity) of tag_i. Note that the server stores two versions of N_i, K_i and PID_i, that is, the current version N_i^{new}, K_i^{new} and PID_i^{new}, and the old version N_i^{old}, K_i^{old} and PID_i^{old}. At the beginning, $N_i^{old} = N_i^{new}$, $K_i^{old} = K_i^{new}$, and $PID_i^{old} = PID_i^{new}$. The server also sends RID_j to $reader_j$ and stores RID_j in the database to register $reader_j$, where RID_j is the pseudonym of $reader_j$.

Below we describe the authentication and key update steps of Huang and Jiang's scheme.

Step 1: Before $reader_j$ queries tag_i about its tag ID, it generates a random number r_1 and sets $V_R = H(RID_j \oplus r_1)$, where H is a hash function. Then $reader_j$ sends a request message (r_1) to tag_i.

Step 2: Upon receiving (r_1), tag_i generates a random number r_2 and uses N_i, K_i and EPC_i to calculate $M_1 = N_i \oplus r_2$ and $M_2 = P(EPC_i||r_1||r_2) \oplus K_i$, where P stands for the PRNG operation. After that, it responds to $reader_j$ with (M_1, M_2, PID_i).

Step 3: After receiving the response message from tag_i, $reader_j$ appends r_1 and V_R to this message to form an authentication request ($M_1, M_2, PID_i, r_1, V_R$) to send to the backend server.

Step 4: Upon receiving the authentication request (M_1, M_2, PID_i, r_1, V_R) from $reader_j$, the server authenticates $reader_j$ by verifying $V_R \stackrel{?}{=} H(RID_j \oplus R_1)$. If the verification is successful, the server uses PID_i to find (N_i^{old}, N_i^{new}, K_i^{old}, K_i^{new}, EPC_i) in the backend database. Note that PID_i may be PID_i^{old} or PID_i^{new}; this can be decided by checking which of $PID_i \stackrel{?}{=} PID_i^{old}$ and $PID_i \stackrel{?}{=} PID_i^{new}$ is successful. The server then verifies $M_2 \stackrel{?}{=} P(EPC_i||r_1||r_2) \oplus K_i^{old}$ and $M_2 \stackrel{?}{=} P(EPC_i||r_1||r_2) \oplus K_i^{new}$ by calculating $r_2 = M_1 \oplus N_i^{old}$ and $r_2 = M_1 \oplus N_i^{new}$. If either of the above verifications is successful, the server sets x= *old* (if K_i^{old} passes the verification) or x= *new* (if K_i^{new} passes the verification), and calculates $M_3 = P(EPC_i||r_2||N_i^x) \oplus K_i^x$ and $Info = D_i \oplus RID_j$ for forwarding the message (M_3, *Info*) to $reader_j$. Moreover, if x=*new*, then the server performs the following updates: $PID_i^{old} = PID_i^{new}$, $PID_i^{new} = P(PID_i \oplus r_2)$, $N_i^{old} = N_i^{new}$, $N_i^{new} = P(N_i \oplus r_2)$, $K_i^{old} = K_i^{new}$ and $K_i^{new} = P(K_i \oplus r_2)$.

Step 5: After receiving the message (M_3, *Info*), $reader_j$ calculates $D_i = info \oplus RID_j$ and forwards M_3 to tag_i.

Step 6: Upon receiving M_3 from $reader_j$, tag_i verifies $M_3 \stackrel{?}{=} P(EPC_i||r_2||N_i) \oplus K_i$. If the above verification succeeds, tag_i performs the following updates: $PID_i = P(PID_i \oplus r_2)$, $N_i = P(N_i \oplus r_2)$, and $K_i = P(K_i \oplus r_2)$.

As shown in [12], Huang and Jiang's scheme can resist several attacks. However, the server needs to store and update many data for tags. For example, for tag_i, the server needs to store EPC_i, N_i^{old}, K_i^{old}, PID_i^{old}, N_i^{new}, K_i^{new}, PID_i^{new} and update N_i^{new}, K_i^{new}, PID_i^{new} for every successful identification session of tag_i.

B. Yi et al.'s Scheme

Yi et al.' scheme [13] uses only the PRNG and the CRC operations. Below we first describe the registration steps. Initially, the backend server randomly selects an initial authentication key K_i^0 and an initial access key P_i^0 for tag_i, which has a unique EPC number EPC_i. The two keys are stored on tag_i and will be updated after each successful authentication session.

The server database maintains a six-field record (EPC_i, K_i^{old}, P_i^{old}, K_i^{new}, P_i^{new}, $DATA_i$) for tag_i. In the record, K_i^{old} (K_i^{new}) is the old (new) authentication key for tag_i and it is set to K_i^0 initially; meanwhile, P_i^{old} (P_i^{new}) denotes the old (new) access key and is set to P_i^0 initially; the last one, $DATA_i$, denotes the full information about the tagged object.

The authentication and key update steps are explained as follows.

Step 1: To query tag_i, $reader_j$ sends tag_i a random number N_1 as a challenge.

Steps 2 and 3: On receiving N_1, tag_i generates a random number N_2 and then calculates $M_1 = N_2 \oplus K_i$ and $M_2 = CRC(K_i || EPC_i || N_1 || N_2) \oplus K_i$. The values M_1 and M_2 are sent back to $reader_j$, which in turn sends (M_1, M_2, N_1) as an authentication request to the backend server.

Step 4: The server retrieves every record and checks if M_2" or M_2' matches M_2, where $M_2" = CRC(K_i^{old} || EPC_i || N_1 || N_2) \oplus K_i^{old}$ and $M_2' = CRC(K_i^{new} || EPC_i || N_1 || N_2) \oplus K_i^{new}$. The check is repeated until a match is found or the end of the database is reached. If a match is found, it implies that tag_i has been successfully authenticated; otherwise, an authentication failure message is sent to $reader_j$ and the authentication step stops.

For the case that tag_i is authenticated successfully, the server calculates $M_3 = CRC(EPC_i || N_2) \oplus P_i^{old}$ or $M_3 = CRC(EPC_i || N_2) \oplus P_i^{new}$ depending on which of K_i^{old} and K_i^{new} leads to the match in the database. It also updates authentication key K_i and access key P_i by setting $K_i = P(K_i^{new} \oplus N_2)$ and $P_i = P(P_i^{new} \oplus N_2)$.

Step 5: The server sends (M_3, $DATA_i$) to $reader_j$, where $DATA_i$ is the information of the object to which tag_i is attached. $Reader_j$ in turn passes M_3 to tag_i.

Step 6: Upon receiving M_3, tag_i has to verify $M_3 \oplus P_i \stackrel{?}{=} CRC(EPC_i || N_2)$. If the verification is successful, it updates its authentication key K_i and access key P_i by setting $K_i^{old} = K_i^{new}$, $K_i^{new} = P(K_i^{new} \oplus N_2)$, $P_i^{old} = K_i^{new}$ and $P_i^{new} = P(P_i^{new} \oplus N_2)$.

In Yi et al.'s scheme, tag_i shares random number (N_1, N_2) and some private information, such as EPC_i, authentication key K_i and access key P_i with the server, where the information is used to build messages M_1 and M_2 in order to prove its authenticity. Unfortunately, since the communication channel between tag_i and $reader_j$ is insecure, an adversary can monitor and modify messages exchanged between them. As shown by Safkhani et al. in [15], Yi et al.'s scheme cannot resist the replay, DoS, forward secrecy and impersonation attacks. Additionally, upon receiving the authentication request (M_1, M_2, N_1) from $reader_j$, the server needs to retrieve every data record in the database and verify if M_2" or M_2' matches M_2 for every record. This will lead to massive computation overheads. For a database of n registered tags, this will cause $n/2$ such verifications in average.

C. Khedr's Scheme

Khedr's scheme (SRFID) [14] adopts the hash and the increment operations that can be implemented on low-cost tags. We first describe the registration steps of the scheme. Initially, the backend server randomly selects an initial hidden ID IDH_i^0 with an initial sequence number SQN_i of an arbitrary value for tag_i. The two values are stored on tag_i and will be updated after each successful authentication session.

The server database maintains a five-field record (IDH_i^{old}, IDH_i^{new}, ID_i^{old}, ID_i^{new}, SQN_i) for tag_i. In the record, IDH_i^{old} (IDH_i^{new}) is the old (new) hidden tag identification number for tag_i and it is set to IDH_i^0 initially; meanwhile, ID_i denotes the tag's current ID and is set to ID_i^0. At the beginning, $IDH_i^{old} = IDH_i^{new}$ which is initial hidden ID of tag_i.

The authentication and key update steps of the Khedr's scheme is described as follows.

Step 1: Before $reader_j$ queries tag_i, it generates a random number R. Then $reader_j$ sends a request message (R) to tag_i.

Step 2: Upon receiving (R), tag_i uses IDH_i and SQN_i to calculate $ID_i = H(IDH_i || SQN_i)$ and $M_1 = H(IDH_i || R)$, where $H(\cdot)$ is a lightweight hash function. After that, it responds to $reader_j$ with (ID_i, M_1).

Step 3: After receiving the response message from tag_i, $reader_j$ appends R to this message as an authentication request (ID_i, M_1, R) and forwards it to the backend server.

Step 4: Upon receiving the authentication request (ID_i, M_1, R) from $reader_j$, the server uses ID_i as the index to obtain information associated with tag_i from the database for verifying $M_1 \stackrel{?}{=} H(IDH_i || R)$. If the verification is successful, the server sets $SQN_i = INC(SQN_i)$, where $INC(SQN_i)$ is a function returning the value of SQN_i plus a pre-specified fixed value. It then sets $IDH_i^{old} = IDH_i$ and $IDH_i^{new} = H(SQN_i || IDH_i)$. The server afterwards calculates $M_2 = H(IDH_i^{new} || SQN || R)$ and sends the message (ID_i, R, M_2) to $reader_j$. The server further updates the current tag ID for the next authentication session by setting $SQN_i = INC(SQN_i)$, $ID_i^{old} = ID_i$ and $ID_i^{new} = H(IDH_i^{new} || SQN_i)$.

Step 5: After receiving the message (ID_i, R, M_2) from the server, $reader_j$ just forwards it to tag_i.

Step 6: Upon receiving the message (ID_i, R, M_2) from $reader_j$, tag_i calculates $SQN_i = INC(SQN_i)$ and $IDT_i = H(IDH_i || SQN_i)$ and $M_2' = H(IDT_i || SQN_i || R)$ and verify $M_2 \stackrel{?}{=} M_2'$. If the verification is successful, tag_i increases the sequence number again by setting $SQN_i = INC(SQN_i)$, and updates IDH_i by setting $IDH_i = IDT_i$; otherwise, the tag just resets the sequence number by setting $SQN_i = DEC(SQN_i)$, where $DEC(SQN_i)$ is a function returning the value of SQN_i minus a fixed value.

As will be shown later, the overheads of Khedr's scheme are not high. However, as shown by Seyed et al. in [16]. Khedr's scheme cannot resist the replay, forward secrecy and impersonation attacks.

III. Proposed Scheme

This section elaborates the proposed mutual authentication scheme, which has two phases: (1) the register phase, (2) the mutual authentication phase and. Similar to the schemes mentioned in Section II, the proposed scheme assumes the communication between the reader and the tags is insecure, but the communication between the reader and the backend server is secure. Notations used in the proposed scheme are described in Table I, and the detailed steps of the proposed scheme are shown in Fig. 1. Note that $INC(\cdot)$ (resp., $DEC(\cdot)$) used in the scheme is a function taking the sequence number SQN_i as the input to return the value of SQN_i plus (resp., minus) a pre-specified fixed value.

TABLE I. NOTATIONS

K_i	The authentication key shared between tag_i and the server
$\oplus$	The exclusive-or operation
$H(\cdot)$	A lightweight hash function like QUARK [9]
r_i	A random number generated by the server or tag_i
$X \stackrel{?}{=} Y$	A function verifying whether X matches Y
$\|\|$	The concatenation operation
EPC_i	The 96-bit EPC (Electronic Product Code) of tag_i
SQN_i	Sequence number
$INC(\cdot)$	A function returning the sequence number plus a fixed value
$DEC(\cdot)$	A function returning the sequence number minus a fixed value

A. Registration Phase

Initially, the server sends (h_i, EPC_i, K_i, SQN_i) to tag_i and stores (h_i, EPC_i, K_i^{old}, K_i^{new}, SQN_i) in the database to register tag_i, where SQN_i is the sequence number of an arbitrary initial value, K_i is the authentication key, EPC_i is the EPC number, and h_i is the search index of tag_i calculated according to Eq. (1). The calculation in Eq. (1) is based on a lightweight one-way hash function H like QUARK [9] taking EPC_i and SQN_i as input parameters. With h_i as the index, the server can use the binary search to locate the information of tag_i in the database for the purpose of authenticating tag_i in the authentication phase, as will be described later. Note that the server stores two versions, the current (or new) version K_i^{new}, and the old version K_i^{old}, of K_i, where $K_i^{old} = K_i^{new}$ at the beginning.

$$h_i = H(EPC_i || SQN_i) \tag{1}$$

B. Mutual Authentication Phase

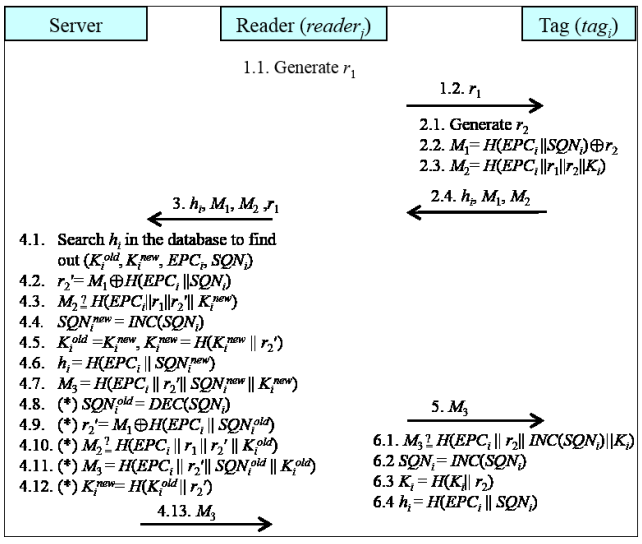

Fig 1. The mutual authentication phase steps of the proposed scheme

The detailed steps of the mutual authentication phase are depicted in Fig. 1 and described as follows.

Step 1: Before $reader_j$ begins to query tag_i, it generates a random number r_1 and then sends a message (r_1) as a challenge to tag_i.

Step 2: Upon receiving (r_1), tag_i generates a random number r_2 and uses EPC_i, SQN_i and K_i to calculate M_1 and M_2 according to Eqs. (2)-(3) and then sends (h_i, M_1, M_2) to $reader_j$.

$$M_1 = H(EPC_i || SQN_i) \oplus r_2 \tag{2}$$

$$M_2 = H(EPC_i || r_1 || r_2 || K_i) \tag{3}$$

Step 3: After receiving (h_i, M_1, M_2), $reader_j$ appends r_1 to this message as an authentication request and forwards (h_i, M_1, M_2, r_1) to the backend server.

Step 4: Upon receiving the authentication request (h_i, M_1, M_2, r_1) from $reader_j$, the server uses the binary search with h_i as the index key to find out (K_i^{old}, K_i^{new}, EPC_i, SQN_i) in the database to calculate r_2' based on EPC_i and SQN_i according to Eq. (4).

$$r_2' = M_1 \oplus H(EPC_i || SQN_i) \tag{4}$$

The server then executes the following verification according to Eq. (5).

$$M_2 \stackrel{?}{=} H(EPC_i || r_1 || r_2' || K_i^{new}) \tag{5}$$

If the verification in Eq. (5) is successful, then tag_i is authenticated. The server afterwards updates the information of tag_i and calculates M_3 according to Eqs. (6)-(9). After that, the server forwards the message (M_3) to $reader_j$.

$$SQN_i^{new} = INC(SQN_i) \tag{6}$$

$$K_i^{old} = K_i^{new},\ K_i^{new} = H(K_i^{new} || r_2') \tag{7}$$

$$h_i = H(EPC_i || SQN_i^{new}) \tag{8}$$

$$M_3 = H(EPC_i || r_2' || SQN_i || K_i^{new}) \tag{9}$$

But if the verification in Eq. (5) fails, tag_i is not authenticated. It is probably that tag_i is illegal or tag_i just does not update its information properly. The server then performs the second authentication by executing the actions marked with asteroids depicted in Fig. 1. Note that the second authentication depends on the pervious sequence number and pervious authentication key to authenticate tag_i. The server actions are explained below. The server first obtains the pervious sequence number and recalculates r_2' according to Eqs. (10)-(11).

$$SQN_i^{old} = DEC(SQN_i) \tag{10}$$

$$r_2' = M_1 \oplus H(EPC_i \| SQN_i^{old}) \quad (11)$$

The server then performs the re-verification shown in Eq. (12).

$$M_2 \stackrel{?}{=} H(EPC_i \| r_1 \| r_2' \| K_i^{old}) \quad (12)$$

If the authentication fails, then tag_i is not authenticated and the authentication phase stops abnormally. Otherwise, tag_i is authenticated, and the server then calculates M_3 according to Eq. (13).

$$M_3 = H(EPC_i \| r_2' \| SQN_i^{old} \| K_i^{old}) \quad (13)$$

After that, the server forwards the message (M_3) to $reader_j$. Moreover, the server updates the information of tag_i according to Eq. (14).

$$K_i^{new} = H(K_i^{old} \| r_2') \quad (14)$$

Step 5: After receiving the transmission message (M_3) from the server, $reader_j$ forwards (M_3) to tag_i.
Step 6: Upon receiving message (M_3) from $reader_j$, tag_i performs the verification shown in Eq. (15).

$$M_3 \stackrel{?}{=} H(EPC_i \| r_2 \| INC(SQN_i) \| K_i) \quad (15)$$

If the verification is successful, tag_i updates its information according to Eqs. (16)-(18). But if the verification fails, $reader_j$ is not authenticated and tag_i aborts its authentication phase.

$$SQN_i = INC(SQN_i) \quad (16)$$

$$K_i = H(K_i \| r_2) \quad (17)$$

$$h_i = H(EPC_i \| SQN_i) \quad (18)$$

IV. SECURITY ANALYSES

In this section, the security of the proposed scheme is analyzed. Note that T, R, and S respectively represent tag_i, $reader_j$ and the server in the following context.

A. MitM Attack Analysis

When $reader_j$ interrogates tag_i, an adversary initiates the MitM attack to intercept the message sent between $reader_j$ and tag_i. Afterwards, the adversary pretends to be a legal reader (resp., tag_i) to forward tampered messages to tag_i (resp., $reader_j$) to pass the authentication and deliver some forged information so that the server and tag_i lose key synchronization and cannot authenticate each other properly in the next run.

Because the server and tag_i first perform the authentication and then update their authentication key (K_i) according some information securely embedded in the authentication information (M_1, M_2, M_3), it is impossible for an adversary to inject or modify information to pass the authentication and then affect the update of keys. The proposed scheme can thus resist the MitM attack.

B. Replay Attack Analysis

If an adversary obtains the information (h_i, M_1, M_2) transmitted between tag_i and $reader_j$, and then initiates the replay attack to spoof the server by transmitting previously obtained information to pass the authentication. However, the adversary cannot pass the authentication. This is because that r_2, h_i, SQN_i, K_i are updated after each authentication to be r_2' , h_i' , SQN_i', K_i^{new} in the next round, and thus the legitimate M_1, M_2 in the next round (denoted by M_1' and M_2' respectively) should be $M_1' = H(EPC_i \| SQN_i) \oplus r_2'$ and $M_2' = H(EPC_i \| r_1' \| r_2' \| K_i^{new})$. Therefore, the adversary cannot replay the obtained information (h_i, M_1, M_2) to pass the authentication.

C. Forward Secrecy Attack Analysis

In the forward secrecy attack, an adversary compromises keys shared by tag_i and $reader_j$ and then tries to calculate previous keys to reveal information transmitted earlier between tag_i and $reader_j$.

Suppose that the adversary has compromised SQN_i and K_i shared by tag_i and server. Since SQN_i and K_i are calculated by evoking the increment function and the hash function, no previous versions of SQN_i and K_i can be obtained even when they are compromised at some instance. The proposed scheme can thus resist the forward secrecy attack.

D. DoS Attack Analysis

In the DoS attack, an adversary can intercept the message (M_3) sent from $reader_j$ to tag_i, where $M_3 = H(EPC_i \| r_2' \| SQN_i \| K_i^{new})$. Such an adversary prevents tag_i from updating the shared keys and makes the shared keys stored on the server different from those stored on tag_i. Therefore, the server (and hence $reader_j$) and tag_i cannot communicate properly henceforth.

To resist the DoS attack, the new and the old keys (K_i^{old}, K_i^{new}) are all stored on the server. In the case that tag_i fail to update the keys, the server can still allow tag_i to pass the authentication and resynchronizes the keys with tag_i for later communication. Therefore, the proposed scheme can resist the DoS attack.

E. Impersonation Attack Analysis

To initiate an impersonation attack, an adversary can pretend to be a legitimate $reader_j$ (i.e., server) or tag_i to pass the authentication verification of $M_3 \stackrel{?}{=} H(EPC_i \| r_2' \| SQN_i \| K_i)$ and $M_2' \stackrel{?}{=} H(EPC_i \| r_1 \| r_2 \| K_i^{new})$ after eavesdropping on communication messages between $reader_j$ and tag_i. Below we explain why the proposed scheme can resist the impersonation attack.

The adversary can easily get the information (h_i, M_1, M_2, M_3) from the following messages transmitted between tag_i and $reader_j$. However, the adversary cannot get the private information (EPC_i, SQN_i, K_i) stored in the server or the information (r_2, SQN_i, K_i) stored in tag_i, because the above-mentioned information is not transmitted between tag_i and $reader_j$, and between $reader_j$ and the server. Moreover, r_2, SQN_i, K_i are updated after each authentication. Therefore, the adversary cannot calculate the correct communication parameters $M_3=H(EPC_i||r_2'||SQN_i||K_i^{new})$ and $M_2 = H(EPC_i ||r_1||r_2||K_i)$ from the intercepted messages to pass the authentication of $M_3 \stackrel{?}{=} H(EPC_i||r_2'||SQN_i||K_i)$ and $M_2' \stackrel{?}{=} H(EPC_i ||r_1||r_2||K_i^{new})$.

V. COMPARISONS

This section shows the comparisons of the proposed scheme with related schemes, namely Huang and Jiang's [12], Yi et al.'s [13], Khedr's [14], in terms of communication, computation, storage, and data updating overheads. This section also shows security comparisons.

As shown in Table II, the communication overhead (i.e., the number of bits transmitted) between tag_i and $reader_j$ are first examined. In Table II, L_{HELO}, L_K, and L_{ID} stand for, respectively, the length (128 bits) of the hello message, the key and the tag identity. Furthermore, L_{RNG} and L_H stand for, respectively, the length (128 bits) of the key and LHash output. Furthermore, L_{CK}, L_{HK} and L_{PK} stand for, respectively, the length (128 bits) of the XOR operation result of a key and a CRC output, the XOR operation result of a key and a LHash function output, and the XOR operation result of a key and a PRNG output. By Table II, the communication costs of Huang and Jiang's, Yi et al.'s, and Khedr's schemes are respectively $1L_{HELO}+ 2L_{RNG}+ 1L_{ID}+ 2L_{PK}$ (=768 bits), $1L_{RNG}+ 1L_K + 2L_{CK}$ (= 512 bits) and $2L_{RNG}+ 2L_{ID} + 2L_{HK}$ (=768 bits). We can observe that the proposed scheme has a lower communication cost, which is $2L_{RNG} + 3L_H$ (= 640 bits), than Huang and Jiang's and Khedr's.

TABLE II. COMMUNICATION COST COMPARISONS

Schemes	Communication costs
Huang and Jiang's [12]	$1L_{HELO} + 2L_{RNG} + 1L_{ID} + 2L_{PK}$ (=768 bits)
Yi et al.'s [13]	$1L_{RNG}+ 1L_K + 2L_{CK}$ (= 512 bits)
Khedr's [14]	$2L_{RNG}+ 2L_{ID} + 2L_{HK}$ (=768 bits)
Proposed Scheme	$2L_{RNG} + 3L_H$ (= 640 bits)

*Note that L_{HELO}, L_H, L_{RNG}, L_{CK}, L_{HK}, L_{PK}, L_K and L_{ID} are the bit lengths of the hello message, LHash function output, random number generator output, XOR result of a key with a CRC output, XOR result of a key with a LHash output, XOR result of a key with a PRNG output, key and identity, respectively.

Table III shows the comparisons of the proposed scheme with related ones in terms of the tag and the server computation costs during the authentication phase. In Table III, n stands for the number of tags; T_{XOR}, T_{PRNG}, T_{CRC}, T_H, T_{INC}, T_{DEC} and T_{COMP} stand for the computation cost (or time complexity) for the XOR, PRNG, CRC, increment, decrement and comparison (COMP) operations, respectively. Note that the XOR and the COMP operations have very low computation costs; the computation costs of other operations are higher and higher in the ascending order: T_{INC}, T_{DEC}, T_{PRNG}, T_{CRC} and T_H. Note that the CRC and LHash have almost the same communication costs [14]. As to T_{VERI}, it stands for the computation cost of the verification procedure, which varies with schemes and consists of many operations. However, it should be noted that T_{VERI} is much larger than T_{COMP}. We also assume the server database utilizes the heap tree data structure to achieve (log n) search time complexity to locate out of n records a proper record associated with a given pseudonym in Huang and Jiang's scheme, Khedr's scheme and our proposed scheme.

TABLE III. COMPUTATION COST COMPARISONS

Schemes	Computation costs	
	Tag_i	Server
Huang and Jiang's [12]	$6T_{XOR}+ 5T_{PRNG}+ 1T_{COMP}$	$1T_H+1T_{COMP}$ + (log n)T_{COMP}+ $8T_{XOR}+ 4T_{PRNG}+ 1T_{VERI}$ ($T_{VERI} = 2T_{XOR}+ 2T_{PRNG}+ 2T_{COMP}$)
Yi et al.'s [13]	$5T_{XOR}+ 2T_{PRNG}+ + 2T_{CRC}+1T_{COMP}$	$3T_{XOR}+ 1T_{CRC}+ 2T_{PRNG}+$ (n/2)T_{VERI} ($T_{VERI}= 2T_{XOR}+ 2T_{CRC}+ 2T_{COMP}$)
Khedr's [14]	$4T_H+ 2T_{INC}+ 1T_{COMP}$	(log n)T_{COMP}+ $3T_H+ 2T_{INC}+ 1T_{VERI}$ ($2T_H+ 2T_{COMP}$)
Proposed Scheme	$1T_{XOR}+ 5T_H+ 1T_{INC}+ 1T_{COMP}$	(log n)T_{COMP}+ $7T_H+ 1T_{INC}+ 1T_{DEC}+ 2T_{XOR}$ + $1T_{VERI}$ ($2T_H+ 2T_{COMP}$)

*Note that n stands for number of tags: T_{XOR}, T_{PRNG}, T_{CRC}, T_H, T_{INC}, T_{DEC}, T_{VEFI} and T_{COMP} are the computation costs of the XOR, PRNG, CRC, LHash function, increment function, decrement function, verification and comparison operations/procedures, respectively.

In Huang and Jiang's scheme [12], when tag_i receives the message (r_1, M_3), it takes $3T_{XOR}+ 2T_{PRNG}$ computation cost to calculate M_1, M_2 and M_3, and it takes $1T_{COMP}$ computation cost to compare the calculated M_3 with the received M_3. If the calculated M_3 equals to the received M_3, tag_i spends a cost of $3T_{XOR}+ 3T_{PRNG}$ to calculate N_i, K_i and PID_i. The total computation cost of tag_i is thus $6T_{XOR}+ 5T_{PRNG}+ 1T_{COMP}$. When the server receives (M_1, M_2, PID_i, r_1, V_R), it takes $1T_{XOR}+1T_H$ computation cost to calculate $V_R=H(RID_j \oplus r_1)$, and it takes $1T_{COMP}$ computation cost to compare the calculated V_R with the received V_R. If the calculated V_R equals to the received V_R, the server spends a cost of (log n)T_{COMP} to find a record of PID_i in the backend database, and spends a cost of $2T_{XOR}$ to calculate r_2 and spends a cost $T_{VERI}= 2T_{XOR}+ 2T_{PRNG}+ 2T_{COMP}$ to verify if M_2 matches M_2. If the verification succeeds, the server spends a cost of $4T_{PRNG}+ 5T_{XOR}$ to calculate M_3, N_i, K_i, $Info$ and PID_i. The total computation cost of the server is thus $1T_H+1T_{COMP}$ + (log

$n)T_{COMP}$+ $8T_{XOR}$+ $4T_{PRNG}$+ $1T_{VERI}$ (T_{VERI} = $2T_{XOR}$+ $2T_{PRNG}$+ $2T_{COMP}$).

In Yi et al.'s scheme [13], when tag_i receives the message (N_1, M_3), it takes $3T_{XOR}$+ $2T_{CRC}$ computation cost to calculate M_1, M_2 and M_3, and it takes $1T_{COMP}$ computation cost to compare the calculated M_3 with the received M_3. If the calculated M_3 equals to the received M_3, tag_i spends a cost of $2T_{XOR}$+ $2T_{PRNG}$ to calculate K_i and P_i. The total computation cost of tag_i is thus $5T_{XOR}$+ $2T_{PRNG}$+ + $2T_{CRC}$+$1T_{COMP}$. When the server receives (M_1, N_1, M_2), it retrieves every database record and spends a cost T_{VERI}= $2T_{XOR}$+ $2T_{CRC}$+ $2T_{COMP}$ to verify if $M_2^{''}$ or $M_2^{'}$ matches $CRC(K_i||EPC_i||N_1||N_2) \oplus K_i$. The average time to finish the verification is thus $(n/2)(2T_{XOR}+ 2T_{CRC}+ 2T_{COMP})$, where n is the number of registered tags whose information is stored in the database. If the verification succeeds, the server spends a cost of $3T_{XOR}$+ $1T_{CRC}$+ $2T_{PRNG}$ to calculate M_3, K_i and P_i. The total computation cost of the server is thus $3T_{XOR}$+ $1T_{CRC}$+ $2T_{PRNG}$+ $(n/2)T_{VERI}$ (T_{VERI}= $2T_{XOR}$+ $2T_{CRC}$+ $2T_{COMP}$).

In Khedr's scheme [13], when tag_i receives the message (R, ID_i, M_2), it takes $4T_H$+ $1T_{INC}$ computation cost to calculate ID_i, M_1, SQN and M_2, and it takes $1T_{COMP}$ computation cost to compare the calculated M_2 with the received M_2. If the calculated M_2 equals to the received M_2, tag_i spends a cost of $1T_{INC}$ to calculate SQN_i. The total computation cost of tag_i is thus $4T_H + 2T_{INC} + 1T_{COMP}$. When the server receives (ID_i, M_1, R), it spends a cost of (log $n)T_{COMP}$ to find a record of ID_i in the backend database, and spends a cost T_{VERI}= $2T_H$+ $2T_{COMP}$ to verify if M_1 matches M_1. If the calculated M_1 equals to the received M_1, the server spends a cost of $3T_H$+ $2T_{INC}$ to calculate IDH_i, M_2, ID_i and SQN_i. The total computation cost of the server is thus (log $n)T_{COMP}$+ $3T_H$+ $2T_{INC}$+ $1T_{VERI}$ ($2T_H$+ $2T_{COMP}$).

In the proposed scheme, when tag_i receives the message (r_1, M_3), it takes $1T_{XOR}$+ $3T_H$+ $1T_{INC}$ computation cost to calculate M_1, M_2 and M_3, and it takes $1T_{COMP}$ computation cost to compare the calculated M_3 with the received M_3. If the calculated M_3 equals to the received M_3, tag_i spends a cost of $2T_H$ to calculate K_i and h_i. The total computation cost of tag_i is thus $1T_{XOR}$+ $5T_H$+ $1T_{INC}$+ $1T_{COMP}$. When the server receives (h_i, M_1, M_2, r_1), it spends a cost of (log $n)T_{COMP}$ to find a record of h_i in the backend database, and spends a cost of $1T_{XOR}$ + $1T_H$ to calculate r_2 and spends a cost T_{VERI}= $1T_H$+ $1T_{COMP}$ to verify if M_2 matches M_2. If the calculated M_2 equals to the received M_2, the server spends a cost of $3T_H$+ $1T_{INC}$ to calculate M_3, K_i and h_i. But if $M_2 \neq M_2$, the server spends a cost of $1T_{DEC}$ + $1T_{XOR}$ + $1T_H$ to calculate r_2 and spends a cost T_{VERI}= $1T_H$+ $1T_{COMP}$ to re-verify if M_2 matches M_2. If the calculated M_2 equals to the received M_2, the server spends a cost of $2T_H$ to calculate M_3 and K_i. The total computation cost of the server is thus (log $n)T_{COMP}$+ $7T_H$+ $1T_{INC}$+ $1T_{DEC}$+ $2T_{XOR}$ + $1T_{VERI}$ ($2T_H$+ $2T_{COMP}$). By Table III, we can observe that only Khedr's scheme has lower computation cost than the proposed scheme.

Table IV shows the comparisons of schemes in terms of security. In summary, Yi et al.'s scheme cannot resist the replay, DoS and impersonation attacks and Khedr's scheme cannot resist the MitM, replay and impersonation attacks. However, only Huang and Jiang's scheme and the proposed scheme can resist the MitM, replay, forward secrecy, DoS and impersonation attacks.

As shown in Tables II, III, and IV, Huang and Jiang's scheme and the propose scheme can resist the same number of attacks, while the proposed scheme has lower communication and computation overheads. Below we further compare the two schemes in terms of storage and data update overheads. Both schemes store 4-tuple information, i.e., (EPC_i, N_i, K_i, PID_i) and (h_i, EPC_i, K_i, SQN_i), on tag_i. In Huang and Jiang's scheme, the server database stores a 7-tuple (EPC_i, N_i^{old}, K_i^{old}, PID_i^{old}, N_i^{new}, K_i^{new}, PID_i^{new}) for tag_i, while the proposed scheme stores a 5-tuple (h_i, EPC_i, K_i^{old}, K_i^{new}, SQN_i). For every successfully identification session, Huang and Jiang's scheme updates N_i^{new}, K_i^{new}, and PID_i^{new} with the PRNG operation, while the proposed scheme updates K_i, and SQN_i with the LHash operation. The proposed scheme obviously has lower storage and update overheads.

TABLE IV. SECURITY COMPARISONS

Schemes / Attacks	Huang and Jiang's [12]	Yi et al.'s [13]	Khedr's [14]	Proposed scheme
Resisting MitM attack	Yes	Yes	Yes	Yes
Resisting replay attack	Yes	No	No	Yes
Resisting forward secrecy attack	Yes	No	No	Yes
Resisting DoS attack	Yes	No	Yes	Yes
Resisting impersonation attack	Yes	No	No	Yes

VI. CONCLUSION

This paper proposes an ultralightweight RFID reader-tag mutual authentication scheme to reduce communication and computation overheads and to resist various attacks, such as the MitM, replay, forward secrecy, DoS, and impersonation attacks. The proposed scheme uses only ultralightweight operations, like the RNG, XOR and LHash. Compared with related schemes, namely Huang and Jiang's scheme [12], Yi et al.'s scheme [13] and Khedr's scheme [14], the proposed method can resist more attacks and/or has lower communication, computation, storage, and update overheads.

In the future, we plan to design more efficient and more secure RFID reader-tag mutual authentication schemes using only ultralightweight operations. One direction of the design is to use the Rabin algorithm to encrypt (resp., decrypt) messages by executing one multiplication operation on a tag and to decrypt (resp., encrypt) messages by executing one square root operation on a reader. Since a reader has much more resources, such as memory, energy and computation power, than a tag, the asymmetric computation requirements demanded by the Rabin algorithm encryption and decryption are suitable for designing feasible and secure RFID reader-tag mutual authentication schemes.

REFERENCES

[1] S. Li, L. D. Xu, S. Zhao, "The Internet of Things: a Survey," Information Systems Frontiers, Vol 17, Issue 2, pp. 243-259, 2015.

[2] C. Aggarwal, J. Han, "A Survey of RFID Data Processing," Managing and Mining Sensor Data, Springer, pp. 349-382, 2013.

[3] G. N. Khan, G. Zhu, "Secure RFID Authentication Protocol with Key Updating Technique," in Proc. of the 22nd International Conference on Computer Communications and Networks (ICCCN), pp. 1-5, August 2013.

[4] Z. Y. Wu, S. C. Lin, T. L. Chen, C. Wang, "A Secure RFID Authentication Scheme for Medicine Applications," in Proc. of the Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 175-181, July 2013.

[5] J. S. Cho, S. C. Kim, "Hash-based RFID tag Mutual Authentication Scheme with Retrieval Efficiency," in Proc. of IEEE the 9th International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 324-328, May 2011.

[6] S. Sajal, Y. Atanasov, B. D. Braaten , "A low cost flexible passive UHF RFID tag for sensing moisture based on antenna polarization, " in Proc. of 2014 IEEE Electro/Information Technology (EIT), pp. 542-545, June 2014.

[7] C. C. Chang , W. Y. Chen, T. F. Cheng, "A Secure RFID Mutual Authentication Protocol Conforming to EPC Class 1 Generation 2 Standard," in Proc. of 2014 Tenth International Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), pp. 642-645, August 2014.

[8] M. Safkhani, N. Bagheri, A. Mahani, "On the security of RFID anti-counting security protocol (ACSP)," Journal of Computational and Applied Mathematics, pp. 512-521, 2014.

[9] J. P. Aumasson, L. Henzen, W. Meier, M. Naya-Plasencia, "Quark: a lightweight hash," Journal of Cryptology, vol. 26, p. 313-339, 2013.

[10] Y. Liao, C. Hsiao, "A secure ECC-based RFID authentication scheme integrated with ID-verifier transfer protocol," Ad Hoc Networks, Vol. 18, pp. 133-146, July 2013.

[11] Z. Li, R. Zhang, Y. Yang, Z. Li, "A Provable Secure Mutual RFID Authentication Protocol Based on Error-Correct Code," in Proc. of 2014 Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), pp. 73-78, October 2014.

[12] Y. C. Huang and J. R. Jiang, "An Ultralightweight Mutual Authentication Protocol for EPC C1G2 RFID Tags," in Proc. of 2012 International Symposium on Parallel Architectures, Algorithms and Programming (PAAP'12), pp. 133-140, December 2012.

[13] X. Yi, L. Wang, D. Mao, Y. Zhan, "An Gen2 Based Security Authentication Protocol for RFID," in Proc. of 2012 International Conference on Applied Physics and Industrial Engineering, Vol. 24, pp. 1385-1391, 2012.

[14] W. I. Khedr, "SRFID: A Hash-Based Security Scheme for Low Cost RFID Systems," Egyptian Informatics Journal, Vol. 14, No. 1, pp. 89-98, 2013.

[15] M. Safkhani, N. Bagheri, P. Peris-Lopez, A. Mitrokotsa, J. C. Hernandez-Castro, "Weaknesses in another Gen2-Based RFID Authentication Protocol." in Proc of 2012 IEEE International Conference on RFID-Technologies and Applications (RFID-TA), pp. 80-84, November 2012.

[16] M. A. Seyed, B. Karim, A. Behzad, R. A. Mohammad, "Traceability Analysis of Recent RFID Authentication Protocols," Wireless Personal Communications, March 2015.