Enhancing the Key Distribution Model in the RFID-Enabled Supply Chains

Sarah Abughazalah, Kostantinos Markantonakis, Keith Mayes
Smart Card Centre-Information Security Group (SCC-ISG)
Royal Holloway, University of London
Email: {Sarah.AbuGhazalah.2012, K.Markantonakis, keith.mayes}@rhul.ac.uk

***Abstract*—In this paper, we point out the use of secret sharing strategies as a promising solution for managing the key distribution and recovery in the Radio Frequency IDentification (RFID)-enabled supply chains. To this end, we designed a new model based on a secret sharing approach to solve the key distribution issue within the supply chains. We further proposed a secret key update protocol incorporating a resynchronisation capability to counter the disruptive effects of location tracking, replay attacks, and desynchronisation attacks. Compared with relevant approaches, our work demonstrates a number of advantages in terms of security and performance.**

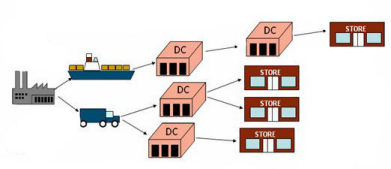
***Index Terms*—RFID; key management; secret sharing;**

I. Introduction

RFID is a wireless technology that uses radio signals to identify objects [1]. RFID is composed of three main components, namely a tag, a reader and a back-end server. The reader broadcasts a radio frequency (RF) signal to power and communicate with RFID tags without any physical contact; the RFID tag is attached to an item and transmits the stored information to nearby reader through the RF channel. The reader sends the tag's data to the backend server, which in turn stores data about the RFID tags it manages.

In this paper, we focus on the supply chain management, where thousands or millions of inbound and outbound products move from the manufacturer to the customers. These products should be correctly identified, verified and sorted at different points in the supply chain. Normally, there are three main parties within the supply chain: the manufacturer, the distributor centers (DC) and the retailer stores. The manufacturer dispatches the cases to different distributors who then send the products to the retailer stores, and in some instances the distributor may dispatch the cases to other distributor as shown in Fig. 1. We use the term "case" as the generic term for a collection of goods and the term "good" for the product itself.

Fig. 1. Supply chain parties



RFID has captured the attention of many leading supply chain companies to make this technology feasible. The RFID technology enables the supply chain to identify, track and verify the products automatically and in real time. By using the RFID technology, all the parties has to store the products' data in their database. The database and the product's tag should share a secret key for protecting the data during transmission. Thus, the secret key should be distributed among all the parties involved in the supply chain.

Basically, a secret key distribution must rely on secure channels established through pre-existing trust relationships. However, in supply chain practice, especially for ad hoc supply chain structures, there is often a lack of trust between the parties involved, as the products' owner may not know the next owner [2]. Moreover, the wireless channel between the reader and tags is vulnerable to possible attacks such as replay attacks, Denial of Service attacks (DoS), counterfeiting attacks and location tracking [3].

One solution to secure the communication channel is to encrypt the exchanged data with a secret key. Yet, the question remains regarding how to keep the secret key uncompromised. Research has been done concerning the possibility of distributing the secret key securely in the RFID supply chain via deploying the secret sharing approach.

Adi Shamir [4] proposed a secret sharing approach, where a secret can be divided into (n) parts that, individually, do not render any useful information about the secret. To reconstruct the secret, not all the parts are needed, any (k) of the parts are sufficient to reconstruct the original secret. This scheme is called a (k, n) threshold scheme.

The main contribution of this paper include:

1) A proposal has been introduced in [5], where the authors proposed to distribute one secret key securely among the supply chain parties by using a secret key sharing approach. Their model assumes that the distributor area is secure as the adversaries have no access. The reader can refer to Section III-A for more details.
   However, in this paper, unlike Jules et al. assumption that the distributer centers are secure, we assume that any place outside the manufacturing area is insecure, thus we designed a protocol for protecting the tags' data outside the manufacturing area. Beside that, we improve their model by generating two secret keys, one key for the cases that hold the products and the other key is for the products' tags. Hence, facilitating the cases identification for the distributer as it needs to read the cases' tags only not all the products' tags.
2) Cai et al. [6] found that Jules et al. model [5] is vulnerable to location tracking and counterfeiting attack. Thus, they designed a protocol for updating the secret key after recovering enough shares and authenticating the tags successfully. Therefore, the tag will respond with new data. The reader is referred to Section III-B for more details. However, we discovered that an attacker can desynchronise a tag without even compromising the internal data stored in the tag. As as result, the attacker

will be able to track the location of the tag and render the tag unusable in the next session. Hence, in this paper, we propose a new secret key update protocol in the RFID-enabled supply chains to prevent such attacks.
The operational environment of the proposed model is shown in Table I. In Table I, we assume that there are two distributors for clarification but it can be more.

The remainder of the paper is organised as follows: Section II lists the main goals of the proposed scheme. Section III summarises the main related work to key distribution in the RFID-enabled supply chain. In Section IV we explain the proposed scheme in detail. Section V is an analysis of our proposed scheme against the stated security requirements. Section VI discusses the parameterization issues. Finally, Section VII provides the concluding remarks.

II. The main requirements and goals

The proposed model should meet the following requirements and goals:

- *Privacy:* The designed scheme must achieve two important notions related to privacy namely:
  - *Tag information privacy:* RFID tags should provide a mechanism for preventing the tag information from being revealed by any malicious reader. For example, encrypting the tag's reply will only allow an authorised reader to decrypt it.
  - *Untraceability:* If the data being sent from the tag to the reader is static or linked to data sent previously, the tag's holder location can be tracked without his knowledge. Therefore, the RFID tag's data should be anonymous and unlinkable.
- *Security:* The designed protocol should resist the following attacks:
  - *Replay attack:* The adversary can eavesdrop on the communication between a reader and tag, reuse the data and send it repeatedly.
  - *Desynchronisation attack:* The adversary can eavesdrop on the communication between a reader and tag, and prevent messages from reaching their target, thus causing a desynchronisation attack.
  - *Tag and reader impersonation attack:* The attacker sends a message to the reader that claims to come from a legitimate tag, and this message fabrication enables the attacker to masquerade as a legitimate tag. The same applies to the reader impersonation.
  - *Secret key secrecy:* We require that the adversary cannot recover the secret key unless he can get access to at least (k) shares..
- *Mutual authentication:* The scheme should provide a mutual entity authentication, where the communication should take place between valid tags and readers and provide assurance to the receiver (reader) about the identity of the sender (tag) and vice versa.
- *Flexibility:* The proposed model should allow all the participants in the supply chain to change the threshold parameters to process the cases and goods that have been divided into smaller or larger quantities.

III. Related Work

Most of the proposed privacy enhanced RFID mutual authentication protocols such as [3], [14]–[18] are based on storing the secret key used for authentication in a database. This may fail on delivering the secret key to the correct parties especially when a large scale of RFID tags move along ad hoc supply chains.

A practical solution to the key distribution problem is the secret sharing approach. Some research have been proposed to distribute the secret key securely based on the notion of secret key sharing [2], [5]–[7].

To begin, Langheinrich et al. [7] proposed the "Shamir tag", which is the first proposal based on using secret sharing approach in the RFID systems to protect the transmission of the tag's ID. This approach splits the ID of a tag into multiple shares based on Shamir secret sharing scheme [4], and stores all the shares on the tag itself. These shares are concatenated to form the new ID of a tag. Upon a reader's inquiry, an initial set of random bits from the new ID is released, followed by subsequent throttled single-bit releases. Once the entire new ID is released, the reader can compute the original ID. In this scheme, an RFID reader requires several minutes to recover the ID, which is not practical in the supply chain, where a large number of tags need to be processed [6].

Li et al. [2] proposed a new scheme called "Resilient Secret Sharing (RSS)". In this paper, the authors designed a secure and practical key distribution system between three parties (A, B, C) in the supply chain. Each tag stores two shares; one share belongs to the secret key (x) between A and B, and the other share is intended for the secret key (y) between A and C. The secret key (x) is divided into multiple shares that are stored in (r) tags, and the remaining shares (n-k-r) are stored in the database, where n is the number of shares, and k is the number of shares required to reconstruct the secret key. The same process is done on the secret key (y). Although this scheme is secure, the key shares are still distributed to a database, which does not solve the key distribution problem.

In this paper we focus on two research papers [5] and [6] as they are relevant to our main goals. These papers are discussed further in the following section.

A. Review of Jules et al. unidirectional RFID key distribution

In [5], the author proposed a secret key sharing approach to be used within the RFID-enabled supply chain to protect the transmission of the tag's ID. This approach does not require any computations on the tag side, the tag just stores two values and sends them to the reader.

The authors suggested the following:

- A group of tags share one secret key (K).
- Using a threshold scheme (k, n) where k $\leq$ n, the secret key is divided into (n) shares, only (k) shares are needed to reconstruct the secret key.
- The ith tag T_i stores two values namely share (S_i) and a symmetrically encrypted information ($E_K(EPC_i)$), where the Electronic Product Code (EPC) [8] is a universal identifier that provides a unique identity to every tag.
- Each ith good's tag sends (S_i, $E_K(EPC_i)$) to any distributer/retailer reader who queries it.
- When the reader receives k shares from the goods'tags, it recovers the secret key and obtains the EPC value for each tag.

The authors claim that their protocol is secure due to the following reasons:

TABLE I
THE OPERATIONAL ENVIRONMENT OF THE PROPOSED MODEL

	Responsibility	Aim
Manufacturer	1- Use two types of RFID tags. A tag attached to each case that holds the products, and a tag attached to each product in the case. 2- Generate two secret keys, one key is for the cases that hold the products and the other key is for the products' tags, and divide them into multiple shares. 3- Specify the secret key threshold to recover the secret keys. 4- Assign one share of the cases' secret key to each case's tag along with the encrypted tag's ID and any other values. 5- Assign one share of the products' secret key to each product's tag along with the encrypted tag's ID and any other values. 6- Lock all the products' tags with different PINs.	Facilitate the cases identification process within the distributer center. Encrypt the tags' unique IDs and protect the tag's identity from being revealed. Protect the goods' tags from revealing their data to any entity.
Distributer 1	Stage 1: Read all the cases' tags and collect enough shares to recover the cases' secret key. Stage 2: 1- Update the secret key and generate new shares. 2- Dispatch the products to the next distributer.	Authenticate the cases' tags. 1- Distribute the cases to downstream distributer(s) with new secret key threshold. 2- Resist replay attack, desynchronisation attack and impersonation attack.
Distributer 2	1- Read all the cases' tags and collect enough shares to recover the cases' secret key. 2- Dispatch the products to the retailer.	Authenticate the cases' tags.
Retailer	1- Retrieve a list of PINs (from step 6 above) associated to each product's tag from the manufacturer. 2- Unlock the tags. 3- Read all the products tags and collect enough shares to recover the products' secret key.	Allow the retailer's reader to unlock the tag. Authenticate the products' tags.

- This scheme provides an efficient solution for a tag ownership transfer, as there is no need to distribute the secret key in the supply chain databases.
- The adversary will not be able to recover the secret key when he obtains the two values (S_i, (E_K(EPC$_i$))) from a customer's tag as he needs to collect k shares to recover the secret key.
- The authors assumed that the manufacturer and distributer centers are secure where the adversary does not have access, only the legitimate parties can collect k shares.

Cai et al. [6] shows that the tag's response can be tracked as (S_i, E_K(EPC$_i$)) are fixed and sent to any reader who queries it. Also, an adversary can obtain the fixed tag's reply, and thus he is able to counterfeit the tag.

In addition, from our point of view, it is inefficient for the distributer to scan all the goods' tags to find the secret key. The distributor's concern is to make sure that all the cases containing goods are delivered safely.

B. Review of Cai et al. secret key update protocol

Cai et al. [6] proposed an enhanced protocol for Jules et al. scheme [5] to avoid tag tracking and counterfeiting attacks. They presented a protocol for updating the secret key (K) and shares (S) after recovering the secret key and authenticating the tag successfully, thus the tag will respond with new values.

The authors proposed to store a new data called (c) to serve as a secret value and to authenticate the reader before updating the data. This value is stored in the tag i.e. c_i=h(K $\|$ S_i), where (h) is a hash function, h:$\{0,1\}^* \rightarrow \{0,1\}^L$, and L is the security parameter. The tag responds to the reader query with three values namely (S_i, c_i, M_i), where M_i=E_K(EPC$_i$).

During manufacturing, the manufacturer assigns the initial data (S_i, c_i, M_i) to the tag. The protocol uses simple cryptographic functions such as hash function, XOR operator ($\oplus$) and concatenation operator ($\|$). A $\leftarrow$ B means that the value of A is updated to that of B.

The protocol in [6] is described as follows:

- Tag $\rightarrow$ Reader: The tag T_i sends (S_i, c_i, M) to the reader.
- Reader: After receiving k shares and recovering the secret key, the reader calculates c_i=h(K $\|$ S_i) to find a match with the received c_i, if there is a match, it authenticates the tag.
- Reader: If the reader successfully authenticated the tag T_i, the reader generates a new secret key K$'$, divides it into (n) new shares, calculates M_i' and generates a new threshold(k$'$, n$'$).
- Reader: For the i^{th} tag T_i, the reader calculates:
  - c_i'=h(K$'$ $\|$ S_i')
  - A=(S_i' $\|$ M_i') $\oplus$ h('0' $\|$ c_i)
  - B=c_i' $\oplus$ h(1 $\|$ c_i)
  - C=h(c_i $\|$ S_i' $\|$ M_i' $\|$ c_i')
- Reader $\rightarrow$ Tag: The reader sends (A,B,C) to T_i.
- Tag: After receiving (A,B,C) from the reader, T_i computes:
  - (S_i' $\|$ M_i')=A $\oplus$ h('0' $\|$ c_i)
  - c_i'=B $\oplus$ h(1 $\|$ c_i)

  If C=hash(c_i $\|$ S_i' $\|$ M_i' $\|$ c_i'), the reader is authenticated. Then T_i updates its values to:
  - $S_i \leftarrow S_i'$
  - $M_i \leftarrow M_i'$
  - $c_i \leftarrow c_i'$

Cai et al. claim that their protocol is immune against location tracking, as the tag data are updated after a successful key recovery process. Thus, the tag will reply with new data every time the reader queries it. However, we deduce three attacks over their protocol namely desynchronisation attack, location tracking, and failure tag authentication. The attacks

work as follows:

1) *Desynchronisation attack:*
   Cai et al. protocol does not provide resistance to desynchronisation attack. An adversary can easily cause a synchronisation failure by intercepting the communication between the reader and the tag. If the reader's messages sent to x tags, where x $<$ k, are blocked, the next owner's reader will not be able to recover the new secret key and complete the authentication, causing a desynchronisation attack.
2) *Location tracking:*
   Since the tag does not receive any data from the reader as a result of a desynchronisation attack, it will reply with the same data (S_i, c_i, M_i) for every query it receives, and hence permits location tracking.
3) *Authentication failure:*
   The reader will not authenticate the compromised tag anymore due to desynchronising the tag's data, hence rendering the tag unusable.

IV. A KEY DISTRIBUTION IN AN RFID-ENABLED SUPPLY CHAINS SCHEME

In order to solve the security and flexibility problems found in previous works [2], [5]–[7] while maintaining their merits, we propose a flexible and secure key management and recovery model as an enhancement of the Jules et al. model and Cai et al. secret key update protocol. The details of our work are illustrated below.

A. Our approach

The proposed scheme is described below:

1) Manufacturer initialisation process:
The main goal of this process is to protect the secret keys namely (K_C and K_T) from being revealed during the transmission to the next owner. The manufacturer does the following:

1) The manufacturer generates two random numbers (R1 and R2).
2) The manufacturer calculates K_C=h(R1) and K_T=h(R2) where h is a one-way hash function. K_C is the secret key for the cases and K_T is the secret key for the goods.
3) The manufacturer splits K_C into n shares and stores one share (S_i) in the i^{th} case's tag memory. Similarly, K_T is divided into n′ shares, where each share (S_i') is stored in the i^{th} good's tag memory, as shown in Fig. 2.
4) The manufacturer specifies the threshold (k, n) for recovering (K_C) secret key and (k′, n′) for recovering (K_T) secret key. We assume that the number of shares (k, k′) is large enough to prevent the attacker from reading k or k′ shares.
5) For each tag T_i in the cases, the manufacturer computes C_i=h($K_C \oplus S_i$) and stores C_i in the case tag's memory for authenticity purpose.
6) To ensure privacy of the tag's data, such as EPC during transmission, the manufacturer encrypts each case tag's EPC value and stores it in the tag's memory, i.e. EPC-Case$_i$=$E_{K_C}\{EPC_i\}$, and similarly encrypts the EPC value of each good's tag and stores it in the tag's memory, i.e. EPC-Tag$_i$=$E_{K_T}\{EPC_i\}$, where E_{K_C} and E_{K_T} represent a symmetric key encryption using K_C and K_T, respectively.
7) The manufacturer locks only the goods' tags with an access-PIN, thus the tag will only send its data to the intended retailer(s) if it receives the correct access-PIN.
8) The manufacturer puts the goods inside the cases and then sends these cases to the next distributer.

2) Distributor key recovery:
In this section we discuss the key recovery process when all the cases reach the distributor. The distributor uses the threshold scheme to recover the secret key (K_C), decrypt the encoded-EPC (EPC-Case$_n$) and obtain the EPC values.

- When the distributor ensures that all the expected cases have arrived, the reader scans the cases' tags.
- Each tag sends (S_i, C_i, EPC-Case$_i$) to the reader.
- The reader collects k shares to obtain the secret key (K_C).
- The reader decrypts EPC-Case for each case's tag using the recovered secret key (K_C) to retrieve the EPC value.
- The reader re-calculates C_i=h($K_C \oplus S_i$) for every cases' tags to authenticate the cases. If there is a match, the distributer looks for the next owner:
  - If the next owner is another distributor the secret key update process is performed as shown in Section IV.A.3.
  - In case that the next owner is a retailer, the distributor updates the threshold of the goods' tags (k_{new}', n_{new}') to fit the new dispatched items as shown in Section IV.A.4. In addition, the distributor should destroy the cases' tags as they will not be used anymore; hence avoid tampering or tracking the cases' tags.

3) Distributer secret key update process:
This process occurs only if the next owner of the RFID tags is another distributor. The main goals of updating the secret key and the threshold are to prevent location tracking and counterfeiting attacks. If K_C, S_i and EPC-Case$_i$ are fixed, the attacker will be able to trace the location of the tag and/or obtain such data to counterfeit a legitimate tag. The secret key update process and resynchronisation process is shown in Table II.

- *Distributor key initialisation process:*
  Once the reader recovered the secret key, and obtained EPC_i for a specific tag T_i, it does the following:
  1) The reader generates a random number (R).
  2) The reader calculates K_C'=h(R). K_C' is the new secret key for the cases.
  3) For each tag, the reader computes EPC-Case2$_i$=$E_{K_C}'\{EPC_i\}$ for the i^{th} tag in the system.
  4) The reader divides K_C' into n new shares (S_n').
  5) The reader specifies the new threshold parameters (k_{new}, n_{new}) for recovering (K_C').
  6) The reader calculates C_i'=h($K_C' \oplus S_i'$).
  7) To distribute the new values of S_i', and C_i' securely for the i^{th} tag, the reader does as follows:
     - Generates a random number R1.
     - Calculates M1= h($EPC_i \oplus$ EPC-Case2$_i \oplus$ R1 $\oplus$ C_i) $\oplus$ S_i'.

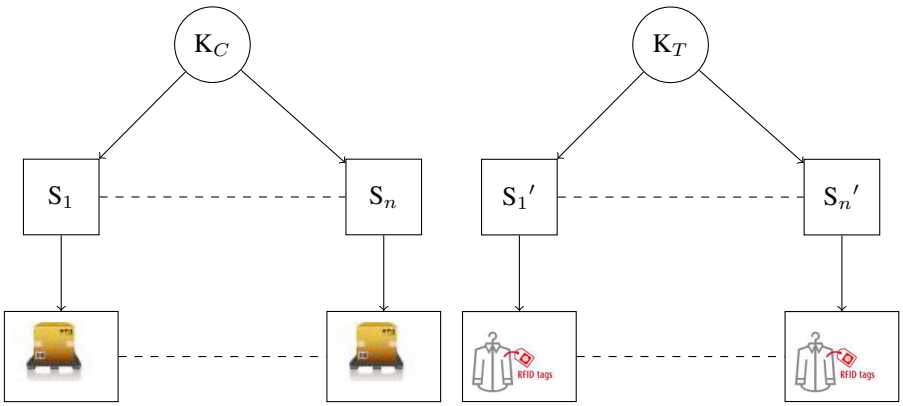


Fig. 2. Key splitting process

- Calculates M2= h(S_i' ⊕ EPC_i ⊕ R1) ⊕ C_i'.
- Sends M1, M2, EPC-Case2$_i$ and R1 to the tag.

8) When the tag receives the messages, it obtains S_i' and C_i' by calculating:
S_i'= M1 ⊕ h(EPC_i ⊕ EPC-Case2$_i$ ⊕ R1 ⊕ C_i)
C_i'= M2 ⊕ h(S_i' ⊕ EPC_i ⊕ R1)
Now, the tag guarantees that the reader has successfully recovered the secret key, decrypted EPC-Case$_i$ and obtained EPC_i. The tag updates its values to:
$S_i \leftarrow S_i'$
EPC-Case$_i$ ← EPC-Case2$_i$
$C_i \leftarrow C_i'$
9) To inform the reader that the tag has received the new data, it does the following operations:
   - The tag generates a random number R2.
   - The tag calculates M3= H(EPC_i ⊕ S_i ⊕ C_i ⊕ R1 ⊕ R2), where S_i and C_i are the updated values, and sends M3 to the reader, along with R2.
10) The reader then recalculates M3. If there is a match, the reader guarantees that the tag has updated its values successfully.
11) The distributor dispatches the cases to the next distributor. However, if the reader did not receive M3 from the tag, the reader starts the resynchronisation process as shown in the next section.

- *Resynchronisation process:*
If the reader did not receive M3 from the tag, then the reader does the following:

1) To re-distribute (S_i', C_i') securely for the i^{th} tag, the reader does as follows:
   - Generates a new random number (R3).
   - Computes M4= h(EPC_i ⊕ EPC-Case2$_i$ ⊕ R3) ⊕ S_i'.
   - Computes M5= h(S_i' ⊕ EPC_i ⊕ R3) ⊕ C_i'.
   - Sends a resynchronization request with M4, M5, EPC-Case2$_i$ and R3 to the tag.
2) When the tag receives a resynchronisation request, M4, M5, EPC-Case2 and R3, it re-computes M4 and M5 as in step 8 above.
   a) If the received values of S_i', EPC-Case2 and C_i' are equal to the current values, the tag assumes that M3 did not reach the reader and keeps the values the same without update.
   b) If the received values of S_i', EPC-Case2 and C_i' are not equal to the current values of S_i, EPC-Case and C_i, the tag updates its values to:
   $S_i \leftarrow S_i'$
   EPC-Case$_i$ ← EPC-Case2$_i$
   $C_i \leftarrow C_i'$
3) The tag generates a random number R4.
4) The tag calculates M6= H(EPC_i ⊕ S_i ⊕ C_i ⊕ R3 ⊕ R4) and sends it to the reader along with R4 to inform the reader that it received the new values. The process is iterated until the reader assures that the tag has updated its values.
5) The distributor dispatches the cases to the next distributor.

4) Retailer key recovery process:
At this stage, the retailer wants to authenticate all the goods' tags and retrieve their EPC values. The reader needs to unlock the tag first and then recover the secret key K_T to obtain the EPC values form the tags.

The good's tag data is both read and write locked so everyone cannot read or modify the tag's data, but only the legitimate reader with the correct access-PIN. We assume that the retailer contacts the manufacturer to retrieve a list of the access-PIN values attached with the tag's sequence number to unlock the tag, and stores them in a database. The authentication and authorisation between the retailer and manufacturer is beyond the scope of this paper.

The purpose of locking the tag is to prevent it from sending the values of S_i and EPC-Tag$_i$ to any reader query, which in turn permits location tracking and impersonation attacks. Thus, in order to obtain the tag's data, the reader has to have the correct access-PIN.

The retailer key recovery process is shown in Table III and summerised below:

- To unlock the tag, the reader sends a random number R1 to the tag as a challenge. The tag responses with another random number R2 and the tag's sequence number.
- The reader retrieves the access-PIN based on the received sequence number from the database, and calculates PIN=h(R1 ⊕ R2 ⊕ access-PIN), and sends PIN to the tag.
- The tag re-calculates PIN. If the received PIN is correct, the tag calculates M1= h(access-PIN ⊕ R1) ⊕ S_i and sends it to the reader along with EPC-Tag$_i$.

- The reader re-calculates M1 $\oplus$ h(access-PIN $\oplus$ R1) to obtain S_i. After unlocking all the tags, the reader collects k' shares to recover K_T.
- Finally, the reader decrypts EPC-Tag$_i$ based on the re-covered K_T and retrieves EPC$_i$ for every i^{th} tag.

V. Analysis

The proposed model also provides protection against the following attacks:

- *Reader impersonation attack:* An attacker may attempt to generate a fake random number and send it with the messages (M1, M2, EPC-Case2$_i$) to the tag in order to modify the tag with incorrect values. However, the attacker cannot calculate M1 and M2, as they involve a secret value namely (EPC$_i$), which is 144 bits length, that is only known to the tag and legitimate reader. Moreover, the attacker cannot impersonate a retailer reader as he needs first to be authenticated by the manufacturer to retrieve a list of access-PINs.
- *Tag impersonation attack:* The attacker cannot send M3 on behalf of the tag to the reader, as it involves three secret values unknown to the attacker (EPC$_i$, S_i, C_i) and it is a result of a one-way hash function that is pre-image resistant.
  The attacker cannot impersonate a good's tag once he obtained M1,and EPC-Tag$_i$ from a previous compromised session as he needs to know the access-PIN value to accomplish such attack.
- *Desynchronisation attack:* The new secret update protocol addresses the realistic scenario, where messages might not reach their intended recipient due to accidental or malicious interference. The following scenario needs to be considered:
  - If the reader does not receive M3 from the tag within a specific timeout, the reader will assume that the tag did not receive M1 and M2, or M3 was lost during transmission, so the reader will start the resynchronisation process and send M4, M5, EPC-Case2 and R3. Similarly, if the resynchronisation messages are blocked or M6 was lost, after timeout the reader will restart the resynchronisation process.

  Eventually, when the jamming stops, the tag and reader will resynchronise and will be able to authenticate each other. In summary, as long as the desynchronise attack is not continuous, the tag and reader will eventually be able to synchronise. All the attacker can gain by jamming the messages is to delay the synchronisation process.
- *Tag information privacy:* The aim of the proposed model is to protect the identity of the tags (EPC). The value of the EPC is protected via encrypting it with a secret key. This key is not openly distributed and is not stored in the participants' database. Also, the tag does not maintain the secret key in its memory. Hence, the attacker cannot obtain the value of EPC unless he collects k shares. It will take up to (2^{144}) attempts to guess the EPC value, which is conceptually strong. Moreover, the access-PIN is not sent in clear, it is protected via the hash function.
- *Untraceability:* The tag's secret key K_C, S, EPC-Case and C values are updated after each successful key recovery process, so the tag's responses will be different for every reader query. Hence, the attacker cannot track the tag's location.
  Resynchronisation process plays an important role in preventing location tracking as the reader keeps resending the new updated data until it confirms that the tag has successfully changed its data.
  Furthermore, the attacker will not be able to track the good's tag purchased by a customer as he needs a correct access-PIN to read the tag's data.
- *Replay attack:* The intruder can eavesdrop on a session, obtain M1 and M2 and resend the messages to the tag. Then, the tag authenticates the intruder, changes its data and sends M3. However, the proposed protocol utilises a challenge-response scheme, where each party maintains a set of random numbers it has seen from previous protocol run to avoid repeatable random numbers.
  In the retailer store, the attacker cannot replay the PIN to the tag because it involves fresh random numbers. As highlighted in the previous paragraph the tag maintains a set of random numbers it has seen from previous protocol run to avoid replay attack.
- *Forward security:* In the proposed protocol, the cases' tag is updated with new values (S, EPC-Case, C), which are totally independent from its previous values.
- *Mutual authentication:* The proposed protocol allows the distributor's reader to access the tag and update the tag's data by sending authentication messages M1 and M2, which confirm that the reader has successfully recovered the secret key and has obtained the right value of (EPC$_i$). Similarly, the tag also sends M3, which confirms to the reader that the legitimate tag has successfully changed the values of S_i, EPC$_i$ and C_i, which can only be obtained by a legitimate tag.
  Similarly, the reader in the retailer shop cannot access the tag until it authenticates itself to the tag via sending the correct access-PIN, which can only be obtained by the legitimate reader (authenticated by the manufacturer). Also, the tag is authenticated to the reader via the access-PIN which is previously assigned by the manufacturer.
- *Flexibility:* In our model, any legitimate party in the supply chain can generate new secret sharing parameters (k_{new},n_{new}). Thus, a downstream party may choose $k_{new} \leq k$ and $n_{new} \leq n$ to process small cases of tags, or choose $k_{new} \geq k$ and $n_{new} \geq n$ to process large cases of tags. Beside that, the distributer needs only to read the cases' tags for an identification purpose instead of scanning thousands or millions of the products' tags.

As shown in Table IV, Cai et al [6] has several issues that this paper attempts to address.

VI. Parameterization

In real-world implementation, the μ-Chip Hibiki tag can be employed [9]. This kind of tags has the capability to lock all the banks in the tag's memory via utilising a PIN (called access-PIN in this paper) to lock the tag's memory.

An EPC tags memory is logically divided into four banks namely, reserved memory for storing the kill and access passwords, UII memory for storing the EPC code, TID memory for storing the tag identifier value, and user memory which allows user-specific data storage.

TABLE II
DISTRIBUTER SECRET KEY UPDATE PROCESS

Reader		Tag$_i$ [EPC$_i$, S$_i$, EPC-Case$_i$, C$_i$]
	$\longleftrightarrow$ $Successfully recover the key$	
1- Generates a random number R 2- Calculates K$_C$′=h(R) 3- Encrypts EPC-Case2$_i$=E$_{K_C}$′{EPC$_i$} 4- Divides the new secret K$_C$′ into new n shares 5- Generates a random number R1 6- Calculates M1= h(EPC$_i$ ⊕ EPC-Case2$_i$ ⊕ R1 ⊕ C$_i$) ⊕ S$_i$′, M2= h(S$_i$′ ⊕ EPC$_i$ ⊕ R1) ⊕ C$_i$′		
7- Sends M1, M2, EPC-Case2$_i$ and R1	$\xrightarrow{M1,M2,EPC-Case2_i,R1}$	8- Calculates S$_i$′=M1⊕ h(EPC$_i$ ⊕ EPC-Case2$_i$ ⊕ R1 ⊕ C$_i$), C$_i$′=M2 ⊕ h(S$_i$′ ⊕ EPC$_i$ ⊕ R1) 9- Updates its values to: S$_i$ ← S$_i$′ EPC-Case$_i$← EPC-Case2$_i$ C$_i$← C$_i$′ 10- Generates a random number R2 11- Calculates M3= h(EPC$_i$ ⊕ S$_i$ ⊕ C$_i$ ⊕ R1 ⊕ R2)
12- Recalculates M3	$\xleftarrow{M3,R2}$	
Resynchronisation process: 1- Generates R3 2- Computes M4= h(EPC$_i$ ⊕ EPC-Case2$_i$ ⊕ R3) ⊕ S$_i$′ M5= h(S$_i$′ ⊕ EPC$_i$ ⊕ R3) ⊕ C$_i$′		
	$\xrightarrow{Resynchronisation request,M4,M5,EPC-Case2_i,R3}$	
		3- Re-computes M4 and M5 4- If S$_i$′ == S$_i$, EPC-Case2 == EPC-Case and C$_i$′ == C$_i$: no update else update: S$_i$ ← S$_i$′ EPC-Case$_i$← EPC-Case2$_i$ C$_i$← C$_i$′ 5- Generates a random number R4
The process is iterated until the tag updates its values successfully	$\xleftarrow{M6,R4}$	6- Calculates M6= h(EPC$_i$ ⊕ S$_i$ ⊕ C$_i$ ⊕ R3 ⊕ R4)

In the proposed scheme, the case's tag stores four values in a rewritable flash memory namely (EPC, S, EPC-Case, and C). The size of each value is 144 bits length. The EPC value is stored in the UII memory, and the values of (S, EPC-Case and C) are stored in the user memory.

The good's tag stores five variables namely (EPC, S, EPC-Tag, access-PIN, and sequence) each of which is 144 bits length. The access-PIN is stored in the reserved memory. The EPC value is stored in the UII memory, and the S, EPC-Tag and sequence values are stored in the user memory.

The designed scheme is compatible with μ-Chip Hibiki tag chips that can store 272 bits in the EPC memory bank and 1536 bits in the user memory bank. Furthermore, additional tag memory is necessary to store a list of random numbers received from previous queries, for example by adding extended on-chip non-volatile memory on the RFID tags.

Due to the limited computational power in the low cost RFID tags, we propose to use the Keccak which has been selected as the winner of the NIST SHA-3 competition in 2012 [10]. The design of the SHA-3 algorithm takes into account the hardware implementation, so it has some properties of lightweight hash function [11]. Kavun et al. [12] shows that Keccak[400], which takes an input of size 144 bits, is suitable for the RFID low cost tag. According to [12], the area of the SHA-3 algorithm is around 5K gate equivalents (GE) which confirms to the requirement of low-cost RFID tags (4-5K gates for security modules).

Distributor scenario: We suppose there are a total of 100 cases (small numbers for more clarity). A manufacturer randomly generates a 144 bits for instance, hashing it with SHA-3, and then take the output as the secret key (K_C) for the cases. K_C is then divided into 100 shares. The manufacturer assigns exactly one share to each case. The manufacturer employs, for example, a (80, 100) secret sharing threshold, so the distributor needs to collect at least 80 shares to recover the cases' secret key. We chose the threshold to be 80 to maximally tolerate (up to 20) reading errors.

Retailer scenario: Similarly, suppose we have 200 goods. A manufacturer generates another random number 144 bits, hashing it with SHA-3, and then takes the output as the secret key (K_T) for the goods. K_T is then divided into 200 shares. The manufacturer assigns exactly one share to each good's tag. The distributor de-packs the goods and re-packs them into smaller 80 goods and delivers them to retailer A, for example. They employ a new (60, 80) secret sharing threshold, so retailer A needs to collect at least 60 shares to recover the

TABLE III
RETAILER KEY RECOVERY PROCESS

Reader		Tag_i [EPC_i, S_i, EPC-Tag_i, access-PIN_i, sequence_i]
1- Generates a random number R1	$\xrightarrow{R1}$	
		2- Generates a random number R2
3- Retrieves the access-PIN based on sequence_i	$\xleftarrow{R2, sequence_i}$	
4- Calculates PIN=h(R1 $\oplus$ R2 $\oplus$ access-PIN)		
	$\xrightarrow{PIN}$	5- Confirms the correctness of the PIN If the received PIN is correct, the tag calculates M1= h(access-PIN $\oplus$ R1) $\oplus$ S_i
6- Obtains S_i by calculating M1 $\oplus$ h(access-PIN $\oplus$ R1)	$\xleftarrow{M1, EPC-Tag_i}$	
7-Collects k' shares to recover K_T		
8- Decrypts EPC-Tag_i based on the recovered K_T		
9- Retrieves EPC_i for every i^{th} tag		

TABLE IV
SECURITY FEATURES COMPARISON

	Cai et al [6]	Section IV
Reader impersonation	$\surd$	$\surd$
Tag impersonation	$\surd$	$\surd$
Desynchronisation attack	$\times$	$\surd$
Tag information privacy	$\surd$	$\surd$
Untraceability	$\times$	$\surd$
Replay attack	$\times$	$\surd$
Forward security	$\surd$	$\surd$
Mutual authentication	$\times$	$\surd$
Goods' tag access control	$\surd$	$\surd$
Flexibility	$\surd$	$\surd$

goods' secret key.

VII. CONCLUSION

In this paper, we proposed an improved version of a key distribution and recovery model in the RFID-enabled supply chain. Firstly, our scheme distributes the secret key securely in the RFID-enabled supply chain via deploying the secret sharing approach. Secondly, it updates the secret key after each successful key recovery, and thus eliminates the threats associated with location tracking. Thirdly, the proposed protocol avoids replay attack using fresh random numbers generated by the reader and tags. Fourthly, to counter the disruptive effects of desynchronisation attacks, the protocol has a resynchronisation phase that is initiated by the reader whenever it suspects a desynchronisation with the tag. Fourthly, the proposed scheme permits the distributor to change the threshold based on the dispatched items, also it facilitates the tags identification process for the distributer. Finally, the proposed protocol meets the main requirements of the low cost RFID tags in terms of storage and computational costs.

VIII. ACKNOWLEDGMENT

This research was supported by the Ministry of Higher Education and King Khaled University in Saudi Arabia.

REFERENCES

[1] K. Finkenzeller, *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*, 2nd ed. New York, NY, USA: John Wiley & Sons, Inc., 2003.

[2] T. Li, Y. Li, and G. Wang, "Secure and practical key distribution for RFID-enabled supply chains," in *Security and Privacy in Communication Networks*. Springer, 2012, pp. 356–372.

[3] B. Song, "RFID authentication protocols using symmetric cryptography," Ph.D. dissertation, Royal Holloway, University of London, 2009.

[4] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[5] A. Juels, R. Pappu, and B. Parno, "Unidirectional key distribution across time and space with applications to RFID security," in *Proceedings of the 17th conference on Security symposium*. USENIX Association, 2008, pp. 75–90.

[6] S. Cai, T. Li, C. Ma, Y. Li, and R. H. Deng, "Enabling secure secret updating for unidirectional key distribution in RFID-enabled supply chains," in *Information and Communications Security*. Springer, 2009, pp. 150–164.

[7] M. Langheinrich and R. Marti, "Practical minimalist cryptography for RFID privacy," *Systems Journal, IEEE*, vol. 1, no. 2, pp. 115–128, 2007.

[8] EPCglobal, "EPC radio-frequency identity protocols class-1 generation-2 UHF RFID protocol for communications at 860 MHz-960 MHz," 2008.

[9] A. Honzawa, "Secure RFID project spread use for product lifecycle management," Jun. 2008. [Online]. Available: http://www.grifs-project.eu/data/File/AHonzawa_Hitachi_UHF_RFID.pdf

[10] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family main document," *Submission to NIST (Round 2)*, vol. 3, 2009.

[11] P. Pessl and M. Hutter, "Pushing the limits of SHA-3 hardware implementations to fit on RFID," in *Cryptographic Hardware and Embedded Systems-CHES 2013*. Springer, 2013, pp. 126–141.

[12] E. B. Kavun and T. Yalcin, "A lightweight implementation of Keccak hash function for radio-frequency identification applications," in *Radio Frequency Identification: Security and Privacy Issues*. Springer, 2010, pp. 258–269.

[13] T. B. Pedersen, Y. Saygın, and E. Savaş, "Secret charing vs. encryption-based techniques for privacy preserving data mining," 2007.

[14] T. Dimitriou, "A lightweight RFID protocol to protect against traceability and cloning attacks," in *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*. IEEE, 2005, pp. 59–66.

[15] H. Chien and C. Chen, "Mutual authentication protocol for RFID conforming to EPC Class 1 Generation 2 Standards," *Computer Standards Interfaces*, vol. 29, no. 2, pp. 254–259, 2007.

[16] D. Duc, H. Lee, and K. Kim, "Enhancing security of EPCglobal Gen-2 RFID against traceability and cloning," in *Symposium on Cryptography and Information Security*. The Institute of Electronics, Information and Communication Engineers, 2006.

[17] E. Yoon, "Improvement of the securing RFID systems conforming to EPC Class 1 Generation 2 Standard," *Expert Systems with Applications*, vol. 39, no. 1, pp. 1589–1594, 2012.

[18] T. Yeh, Y. Wang, T. Kuo, and S. Wang, "Securing RFID systems conforming to EPC Class 1 Generation 2 Standard," *Expert Systems with Applications*, vol. 37, no. 12, pp. 7678–7683, 2010.