IEICE TRANS. INF. & SYST., VOL.E91–D, NO.5 MAY 2008

PAPER Special Section on Information and Communication System Security

A Low Cost Key Agreement Protocol Based on Binary Tree for EPCglobal Class 1 Generation 2 RFID Protocol

Albert JENG[†a)], *Nonmember*, **Li-Chung CHANG**[††b)], *Member*, *and* **Sheng-Hui CHEN**[††], *Nonmember*

SUMMARY There are many protocols proposed for protecting Radio Frequency Identification (RFID) system privacy and security. A number of these protocols are designed for protecting long-term security of RFID system using symmetric key or public key cryptosystem. Others are designed for protecting user anonymity and privacy. In practice, the use of RFID technology often has a short lifespan, such as commodity check out, supply chain management and so on. Furthermore, we know that designing a long-term security architecture to protect the security and privacy of RFID tags information requires a thorough consideration from many different aspects. However, any security enhancement on RFID technology will jack up its cost which may be detrimental to its widespread deployment. Due to the severe constraints of RFID tag resources (e.g., power source, computing power, communication bandwidth) and open air communication nature of RFID usage, it is a great challenge to secure a typical RFID system. For example, computational heavy public key and symmetric key cryptography algorithms (e.g., RSA and AES) may not be suitable or over-killed to protect RFID security or privacy. These factors motivate us to research an efficient and cost effective solution for RFID security and privacy protection. In this paper, we propose a new effective generic binary tree based key agreement protocol (called BKAP) and its variations, and show how it can be applied to secure the low cost and resource constraint RFID system. This BKAP is not a general purpose key agreement protocol rather it is a special purpose protocol to protect privacy, un-traceability and anonymity in a single RFID closed system domain.

key words: *RFID security and privacy, key agreement, binary tree, limited resource tag, low cost*

1. Introduction

RFID [1]–[4] is an automatic identification method, relying on storing and remotely retrieving data using devices called RFID tags or transponders. An RFID tag is an object that can be attached to or incorporated into a product, animal, or person for the purpose of identification using radio waves. Chip-based RFID tags contain silicon chips and antennas. Passive tags require no internal power source, whereas active tags require a power source. RFID tags can be used in passports, transportation payments, product tracking, automotive, animal identification, inventory systems, human implants and libraries using its wireless identification characteristic.

Manuscript received July 17, 2007.
Manuscript revised December 11, 2007.

[†]The author is with the Department of Computer Science and Information Engineering, Jinwen University of Science and Technology, Taiwan.
[††]The authors are with the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan.
a) E-mail: albertjeng@hotmail.com
b) E-mail: lichung@mail.ntust.edu.tw

DOI: 10.1093/ietisy/e91–d.5.1408

A common concern with RFID is privacy risk. Many privacy groups are concerned about the ability to identify and track people when RFID is deployed in real life environment. Another key issue in RFID is security risk. People who use devices that carry personal financial information, such as credit card, do not want others to access their personal information. This is because the radio waves are transmitted into public environment, and this public environment may be monitored or eavesdropped. An adversary can utilize the unique identifier (UID) or electronic product code (EPC) for tracking the object action or obtaining sensitive information. These security problems in low cost RFID systems are raised by R. Damith, and C. Peter in [5]. We were inspired by [5] to survey and analyze existing proposed solutions to secure low cost RFID and found out that the key establishment techniques are the foundation to solve the above security problems [6]. Key establishment techniques are also the basis of cryptography based entity authentication protocols [6]. Key agreement protocol (KAP) is one of the major key establishment techniques. Some researchers have proposed key exchange protocols to protect the privacy and security of RFID [7]–[9]. However, those protocols do not take into consideration all the important factors in RFID technology such as user privacy, security, cost, limited resources and performance at the same time. Thus, we use existing functions of the EPC C1G2 RFID [10] to develop a generic KAP based on binary tree which considers these total factors for the RFID environment.

In this paper, we propose a generic binary tree based key agreement protocol (BKAP) for EPC Class-1 Generation-2 (C1G2) RFID systems. This BKAP is not a general purpose key agreement protocol rather it is a special purpose protocol to protect privacy, un-traceability and anonymity in a single RFID closed system domain. For example, if there is a consumer buys VIAGRA or Cialis, he does not want this privacy information to be disclosed to other people. One can use BKAP to protect his privacy in such cases. However, since the VIAGRA reader used in BKAP to protect the privacy of buying VIAGRA could not be used to exchange keys with the Cialis tags. Similarly, a Cialis reader could not exchange key with VIAGRA tags either. Therefore, BKAP is only good to protect privacy in a single RFID closed system domain. Furthermore, if the UID or EPC code responded by a tag to a reader's query is not varied, then it could be used to track the user's identity and violate the anonymity concern. Since BKAP takes the above factors into consideration, therefore, RFID tags

implemented with this BKAP protocol can be used to provide privacy as well as un-traceability and anonymity of the users. Afterward we will enhance the BKAP (EBKAP) to make it more efficient. These two binary tree based protocols do not use cryptographic function such as symmetric or asymmetric encryption function or hash function. Therefore, it will not require additional resources of the existing RFID system to implement these two protocols. Finally, we will analyze the privacy, security and performance of EBKAP to prove they are both secure and low cost.

2. Preliminaries

In this section, we introduce the terms and the functions for binary tree based KAP (BKAP).

- n: A binary tree has n nodes.
- B_n: A n-node binary tree.
- $PT(tree)$: To traverse a B_n in postorder.
- B_n^{pre}: A set of B_n, and every B_n has the same preorder sequence and different postorder sequence.
- Tag_i: The i-th tag, and every Tag_i is assigned a B_n, and the B_n is selected from the B_n^{pre}.
- B_{n_i}: The B_n of the Tag_i.
- Ps_i: The postorder sequence of the B_{n_i}.
- Km_i: The Master key of the B_{n_i}.
- K_s: Session key.
- $Derive(postorder)$: To read the node content with the Ps_i order.
- $PRNG(seed)$: Pseudo-random number generator.
- $B_n^{pre-post}$: A set of B_n, and every B_n has the same preorder and postorder sequences.
- RIS_t: Random inorder sequence generated from the $B_n^{pre-post}$ by a tag.
- $Secret[n]$: An array of secret data, which has n elements.
- RN_r: Random number generated by a reader.
- $BTree(preorder, inorder)$: To build a unique B_n with a preorder and an inorder sequence.
- NN_s: The total number of singly linked nodes of a B_n.

3. System Model

3.1 Overview

Our BKAP concept is very simple as illustrated in Fig. 1. We divide the memory of readers and tags into several sections and store a pre-defined secret information in the content of the memory. Assume both the tag and the reader will store the same secret data in their memory contents to begin with. Suppose we divided the memory into 4 sections, and we store secret data A, C, D, B into memory sections 1 through 4 respectively. When a reader queries a tag, the tag will respond with the right order of a sequence to read out the contents of those memory sections, and then the reader will read its own memory sections with this order to learn

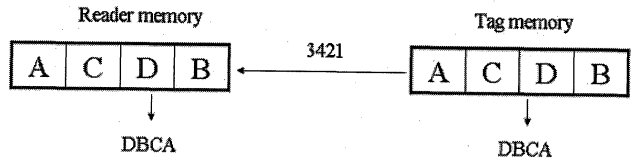


Fig. 1 BKAP concept.

the secret session key. In Fig. 1, a tag responds with a sequence order of 3421 to a reader, the reader learns the correct session key is "DBCA" by reading the contents of its memory sections with this 3421 order. Therefore, the transmitted information in the air is only a sequence order without involving any secret information either in plaintext or in ciphertext.

The advantage of this scheme is that an adversary has to launch a physical attack on a reader's or a tag's memory to learn the secret session key. To launch a physical attack needs a very high class technology and very precise equipments. The high class technology and precise equipments can only be afforded by a national or a large company laboratory. These attacking requirements will jack up the threshold of breaking our proposed generic key agreement protocol based on binary tree.

But this protocol still has some security risks. For example, a tag in this protocol always responds to the query of the reader without prior authentication of the reader. Furthermore, the correct sequence order responded by a tag, if not varied can be used to track the user's identity and violate the anonymity concern. Therefore, we add a binary tree node traversal based technology into this protocol to provide privacy as well as the un-traceability and anonymity of the RFID tags. A tag now can exchange a secret session key with a reader securely with this binary tree based key agreement protocol.

3.2 BKAP

Horowitz [11] describes that a B_n can be uniquely defined with a preorder and an inorder sequences. So we use this characteristic to establish the BKAP which only transmits node order without sensitive data between a reader and a tag for exchanging session key. Now we detail the BKAP protocol.

The BKAP is divided into two phases, creation phase and key agreement phase. The creation phase includes those steps that must be implemented when a reader and a tag are manufactured. The key agreement phase is executed when a reader exchanges a session key with a tag. The following procedure 1 describes the creation phase of BKAP.

Procedure 1 (Creation phase procedure of the BKAP)

(1) Service provider selects a preorder sequence and generates an array $Secret[n]$ which has n elements.
(2) Readers stores the selected preorder sequence and the $Secret[n]$. It is shown in Fig. 2.
(3) Service provider generates a unique B_{n_i}.

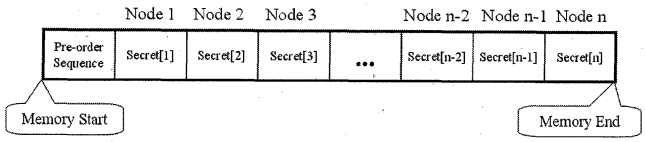


Fig. 2 A reader memory organization.

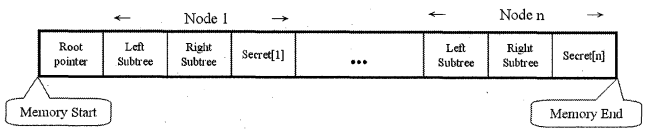


Fig. 3 A tag memory organization of BKAP.

	Reader	Air	Tag
Step 1	Broadcasts Query command	Query →	
Step 2			Calculates $Ps_i = PT(B_{n\_i})$ Derives $Km_i = Derive(Ps_i)$ Generates $RIS_t \in B_n^{pre\text{-}post}$
		← RIS_t	
Step 3	Calculates $Ps_i = PT(BTree(preorder, RIS_t))$ Derives $Km_i = Derive(Ps_i)$ Generates $RN_r = PNRG(time)$		
		RN_r →	
Step 4	Calculates $K_s = PNRG(RN_r \ominus Km_i)$		Calculates $K_s = PNRG(RN_r \oplus Km_i)$

Fig. 4 Key agreement phase of BKAP.

(4) The Tag_i stores the unique B_{n_i} and the $Secret[n]$. It is shown in Fig. 3.

In our BKAP, the key agreement phase consists of 4 primary steps, and it was shown in Fig. 4:

Step1. A reader broadcasts query command.

Step2. A Tag_i executes $PT(tree)$ with B_{n_i} to get Ps_i after receiving query command. Then the Tag_i executes $Derive(postorder)$ with Ps_i to read the $Secret[n]$ of node contents, and then it derives Km_i. Afterward the Tag_i generates RIS_t and transmits it to the reader.

Step3. After the reader receives the RIS_t, it executes $BTree(preorder, inorder)$ with the pre-stored preorder sequence in the creation phase and the received RIS_t to uniquely define B_{n_i}. Then the reader executes $PT(tree)$ with the B_{n_i} to get the Ps_i. The reader executes $Derive(postorder)$ to read the $Secret[n]$ of node contents, and then it derives the Km_i. The reader executes $PRNG(seed)$ to generate a pseudo-random number RN_r and transmits it to the Tag_i.

Step4. The reader uses the RN_r and Km_i as the seed of $PRNG(seed)$ to generate K_s. The Tag_i also uses the RN_r and the Km_i from step 2 as the seed of $PRNG(seed)$ to generate K_s.

3.3 An Example of BKAP

Figure 5, Fig. 6, and Fig. 7 show an example with a 4-node binary tree. They illustrate the whole process of BKAP. The following procedure 2 describes the creation phase of BKAP.

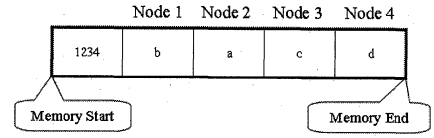


Fig. 5 An example of a reader memory in EBKAP.

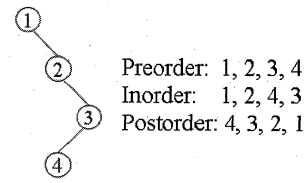


Fig. 6 An example of a B_{n_i}.

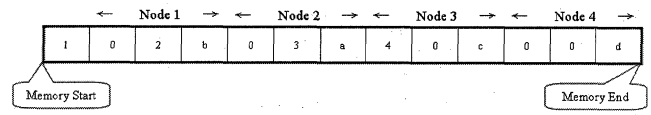


Fig. 7 An example of a tag memory in EBKAP.

Procedure 2 (An example of creation phase procedure of BKAP)

(1) Service provider selects a preorder sequence, 1234, and generates an array $Secret[n]$ which has 4 elements, where $Secret[1] = b$, $Secret[2] = a$, $Secret[3] = c$, and $Secret[4] = d$.

(2) A reader stores the selected preorder sequence and the $Secret[n]$. It is shown in Fig. 5.

(3) Service provider generates a unique B_{n_i} shown in Fig. 6.

(4) The Tag_i stores the unique B_{n_i} and the Km_i. It is shown in Fig. 7.

In this example, the key agreement phase is described as follows:

Step1. A reader broadcasts query command.

Step2. A Tag_i executes $PT(tree)$ with B_{n_i} to get the Ps_i after receiving query command. In this example, the Ps_i is 4321. Then the Tag_i executes $Derive(postorder)$ with Ps_i to read the $Secret[n]$ of node contents which are dcab. The Tag_i use the dcab as Km_i. Afterward the Tag_i generates RIS_t and transmits it to the reader. The RIS_t can be a 4321, 3421, 1432, 1342, 2431, 2341, 1243, and 1234 inorder sequence, we choose 1342 inorder sequence.

Step3. After the reader receives the RIS_t, 1342, it executes $BTree(preorder, inorder)$ with the pre-stored preorder sequence, 1234, in the creation phase and the received RIS_t, 1342, to uniquely define B_{n_i}. Then the reader executes $PT(tree)$ with B_{n_i} to get Ps_i. In

this example, the Ps_i is 4321. The reader executes *Derive*(*postorder*) to read the *Secret*[*n*] of node contents, and the reader derives Km_i, dcab. The reader executes *PRNG*(*seed*) to generate a pseudo-random number RN_r and transmits it to the Tag_i.

Step4. The reader uses RN_r and Km_i, dcab, as the seed of *PRNG*(*seed*) to generate K_s. The Tag_i also uses RN_r and the Km_i, dcab, as the seed of *PRNG*(*seed*) to generate K_s.

3.4 EBKAP

From Fig. 4, we can easily find that the Km_i derived by a Tag_i is the same at every key agreement phase. So we can modify a tag memory organization for removing the two functions *PT*(*tree*) and *Derive*(*postorder*) of the step 2 in the key agreement phase to reduce a tag computing resources and enhance the BKAP performance. The modified tag memory organization is shown in Fig. 8. The following procedure 3 describes the creation phase of EBKAP. It is similar to BKAP.

Procedure 3 (Creation phase procedure of EBKAP)

(1) Service provider selects a preorder sequence and generates an array *Secret*[*n*] which has *n* elements.
(2) A reader stores the selected preorder sequence and the *Secret*[*n*]. It is shown in Fig. 2.
(3) Service provider generates a unique B_{n_i}.
(4) The Tag_i stores the unique B_{n_i} and the Km_i. It is shown in Fig. 8.

In the EBKAP, the key agreement phase also consists of 4 primary steps, and it was shown in Fig. 9:

Step1. A reader broadcasts query command.
Step2. A Tag_i generates RIS_t and transmits it to the reader after receiving query command.
Step3. After the reader receives the RIS_t, it executes *BTree*(*preorder*, *inorder*) with the pre-stored preorder sequence in the creation phase and the received RIS_t to uniquely define B_{n_i}. Then the reader executes *PT*(*tree*) with B_{n_i} to get Ps_i. The reader executes *Derive*(*postorder*) to read the *Secret*[*n*] of node contents, and it derives the Km_i. The reader executes *PRNG*(*seed*) to generate a pseudo-random number RN_r and transmits it to the Tag_i.
Step4. The reader uses the RN_r and derived the Km_i as the seed of *PRNG*(*seed*) to generate K_s. The Tag_i also uses the RN_r and the pre-stored Km_i as the seed of *PRNG*(*seed*) to generate K_s.

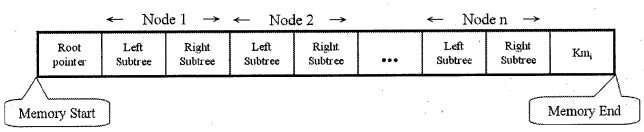


Fig. 8 A tag memory organization of EBKAP.

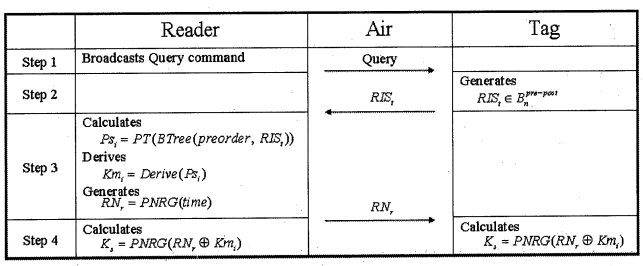


Fig. 9 Key agreement phase of EBKAP.

4. Analysis

The example of BKAP in Sect. 3.2 shows that a tag responds with a variant data (i.e., inorder sequence) to a reader in key agreement phase. This can resist the attack when an adversary uses a constant data to launch tracking attack and eavesdropping attack [5].

4.1 Analysis of Eavesdropping Attack

Figure 4 shows that an adversary in key agreement phase can eavesdrop RIS_t and RN_r from the air interface. The RIS_t is a random inorder sequence and is not a constant data, and the RN_r is also a random number. This means an adversary can not lunch such a tracking attack with random RIS_t and RN_r. In addition, if the adversary wants to get the K_s, he must know the Ps_i and the memory content of a tag or the reader in BKAP. The adversary can get the Ps_i only from brute-force search with RIS_t, and the time of brute-force searching is the well-known Catalan number.

$$C_n = \frac{1}{n+1}\binom{2n}{n} \cong O\left(\frac{4^n}{n^{3/2}}\right) \quad (1)$$

If a computer needs one microsecond for trying a postorder sequence, and the n is 32, the adversary need about 1,760 years for searching the Ps_i. In the above analysis, even though the adversary finds the Ps_i, he does not derive the Km_i without the tag or reader memory content. The adversary only lunches physical attack to get the memory content for analyzing the K_s.

4.2 Analysis of the Size of $B_n^{pre-post}$

We know RIS_t is generated from $B_n^{pre-post}$. The size of $B_n^{pre-post}$ provides the number of the variances in RIS_t transmitted from a tag to a reader. The larger the size of $B_n^{pre-post}$, the more difficult an adversary can track a tag. From [12], we learn that the preorder and postorder sequences of two B_n are the same, but their inorder sequences can be different if a B_n changes its subtree of a given node with a single chain topology from left to right or from right to left. So, if B_{n_i} has a singly linked node, the number of RIS_t of Tag_i are $2^1 = 2$. If B_{n_i} has two singly linked nodes, the number of RIS_t of Tag_i are $2^2 = 4$, and so on. As we mentioned in Sect. 2, we denote that the total singly linked nodes of a binary tree is NN_s. Therefore the number of RIS_t of a Tag_i are 2^{NN_s}.

4.3 Analysis of the Size of B_n^{pre}

Because B_{n_i} is selected from B_n^{pre}, we must find how many B_n can be assigned to a tag. The problem is similar to counting the number of binary trees (B_n) can be built from a given set of n nodes. Figure 10 shows the organization of a binary tree. Let a_n be the size of B_n^{pre} and $a_0 = 0$. If the left subtree has k nodes, then the right subtree has $n - k - 1$ nodes, where $k = 0, 1, \ldots, n-1$. From [11] we know that B_n could have the number of variant binary trees as shown in Eq. (1). But we must subtract the number of B_n which have the same postorder sequences from Eq. (1). The number of B_n with the same postorder sequence is $a_{n-1}a_0$ [12]. So the size of the B_n^{pre} can be derived as follows:

$$\begin{aligned} a_n &= a_0a_{n-1} + a_1a_{n-2} + \ldots + a_{n-3}a_2 + a_{n-2}a_1 \\ &= \sum_{i=0}^{n-2} a_ia_{n-i-1} \end{aligned} \tag{2}$$

Because Eq. (2) has no close form, we use matlab to calculate the value of the a_n, and Fig. 11 shows the result.

4.4 Performance Analysis of EBKAP

4.4.1 Single Tag Environment

In EBKAP, the computational complexity needed in a tag depends on the procedure of generating RIS_t. It can be divided into two tasks. The first task is randomly changing the subtree of every singly linked node from left subtree to right subtree or right subtree to left subtree. The program

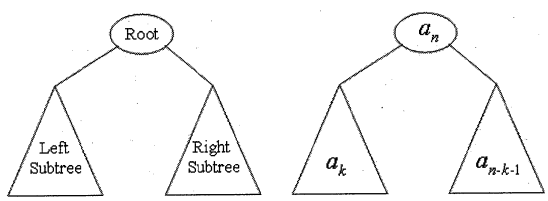


Fig. 10 The definition of a binary tree.

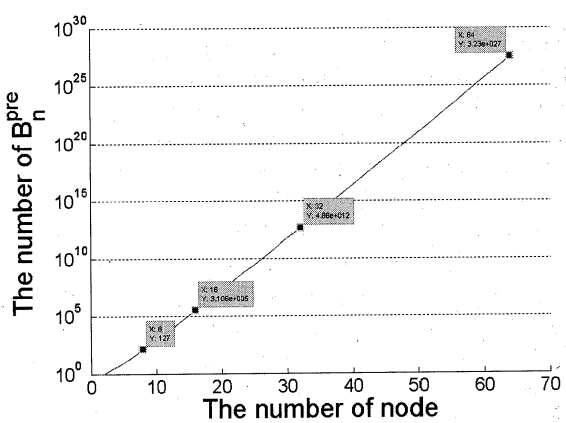


Fig. 11 The size of the B_n^{pre}.

is shown in Fig. 12. We can easy know its time complexity is $O(n)$. The second task is to traverse a B_n via inorder sequence. Its time complexity is also $O(n)$ [11].

In tag memory of the EBKAP, we need two ROMs for pre-storing data. The first ROM is used for pre-storing B_{n_i}, the ROM needs $(2n + 1)\lceil \log_2 n \rceil$ bits or $O(n \log n)$ space, where "⌈x⌉" is a ceiling function which denotes the smallest integer greater than or equal to x. We use 288 logic gates, 12 FlipFlops, and 20 IO Buffer realizing the logic of pre-storing B_{n_i}, where n is 32 as shown in the simulation result of Fig. 13. The second ROM is used for pre-storing Km_i, the ROM needs the k bits or $O(k)$ space, where the k is the length of Km_i. We use 4 logic gates, 2 FlipFlops, and 130 IO Buffer for realizing the logic of pre-storing Km_i, where k is 128 as shown in the simulation result of Fig. 14. In addition, when the EBKAP protocol traverses a B_n in a tag via inorder sequence, it needs a stack with a height d [11]. This mean

```
char* Inorder_Permutation(char * tree)
{
    for(i = 0; i < n; i++) {
        if (tree -> leftsubtree == empty  && tree -> rightsubtree != empty || tree
                -> leftsubtree != empty && tree -> rightsubtree == empty )
            if(rand() mod 2 == 1)
                swap(tree -> leftsubtree, tree -> rightsubtree );
    }
    return tree;
```

Fig. 12 Randomly select sdsd B_n from $B_n^{pre-post}$.

```
================================================================================
*                              Final Report                                    *
================================================================================
Final Results
RTL Top Level Output File Name      : rom.ngr
Top Level Output File Name          : rom
Output Format                       : NGC
Optimization Goal                   : Speed
Keep Hierarchy                      : YES
Target Technology                   : Automotive 9500XL
Macro Preserve                      : YES
XOR Preserve                        : YES
Clock Enable                        : YES
wysiwyg                             : NO

Design Statistics
# IOs                               : 20

Cell Usage :
# BELS                              : 288
#      AND2                         : 60
#      AND3                         : 3
#      AND6                         : 1
#      GND                          : 1
#      INV                          : 117
#      OR2                          : 104
#      OR3                          : 2
# FlipFlops/Latches                 : 12
#      FD                           : 12
# IO Buffers                        : 20
#      IBUF                         : 8
#      OBUFE                        : 12
================================================================================
```

Fig. 13 The simulation result of pre-storing B_{n_i} using Xilinx ISE 9.1i.

```
Design Statistics
# IOs                               : 130

Cell Usage :
# BELS                              : 4
#      GND                          : 1
#      INV                          : 2
#      VCC                          : 1
# FlipFlops/Latches                 : 2
#      FD                           : 2
# IO Buffers                        : 130
#      IBUF                         : 2
#      OBUFE                        : 128
================================================================================
```

Fig. 14 The simulation result of pre-storing Km_i using Xilinx ISE 9.1i.

```
Design Statistics
# IOs                              : 18

Cell Usage :
# BELS                             : 443
#      AND2                        : 153
#      AND3                        : 16
#      AND4                        : 10
#      AND6                        : 1
#      AND7                        : 1
#      AND8                        : 2
#      GND                         : 1
#      INV                         : 173
#      OR2                         : 71
#      OR3                         : 2
#      XOR2                        : 13
# FlipFlops/Latches                : 77
#      FD                          : 4
#      FDCE                        : 73
# IO Buffers                       : 18
#      IBUF                        : 10
#      OBUFE                       : 8
=========================================================================
```

Fig. 15 The simulation result of implementing STACK using Xilinx ISE 9.1i.

```
Design Statistics
# IOs                              : 33

Cell Usage :
# BELS                             : 750
#      AND2                        : 484
#      AND3                        : 55
#      AND4                        : 6
#      AND6                        : 7
#      GND                         : 1
#      INV                         : 125
#      OR2                         : 12
#      OR5                         : 12
#      OR8                         : 48
# FlipFlops/Latches                : 421
#      FDCE                        : 421
# IO Buffers                       : 33
#      IBUF                        : 21
#      OBUFE                       : 12
=========================================================================
```

Fig. 16 The simulation result of implementing RAM using Xilinx ISE 9.1i.

the tag needs a $O(d \log n)$ stack space. The simulation result in Fig. 15 shows that we use 443 logic gates, 77 FlipFlops, and 18 IO Buffer for realizing the logic of the STACK operations, where d is equal to 10. The EBKAP protocol also needs RAM memory to temporarily store data. The simulation result in Fig. 16 shows that we use 750 logic gates, 421 FlipFlops, and 33 IO Buffer for realizing the logic of the RAM processing.

Figure 17 shows the design summary results of realizing the entire EBKAP protocol, which are automatically generated in Xilinx ISE 9.1i environment. It is clear to know that the EBKAP with 32-node binary tree requires 3,474 logic gates, 790 FlipFlops, 31 tri-states, and 9 IO Buffer to implement in digital logic.

4.4.2 Multiple Tags Environment

We assume BKAP and EBKAP will kick in to perform key agreement only after all the tags in the field have been individually identified using some kind of anti-collision technique of EPCglobal C1G2 based identification scheme by the same RFID reader [10]. During the key agreement phase of both BKAP and EBKAP, there is no need to pre-store all the tags' shared secrets in the server database except to store the single selected preorder sequence in the reader. Therefore, when both schemes are used in a closed domain multi-

```
Design Statistics
# IOs                              : 9

Cell Usage :
# BELS                             : 3474
#      AND2                        : 1418
#      AND3                        : 146
#      AND4                        : 58
#      AND5                        : 3
#      AND6                        : 45
#      AND7                        : 1
#      AND8                        : 7
#      GND                         : 5
#      INV                         : 1174
#      OR2                         : 505
#      OR3                         : 19
#      OR5                         : 12
#      OR8                         : 48
#      XOR2                        : 33
# FlipFlops/Latches                : 790
#      FD                          : 33
#      FDCE                        : 757
# Tri-States                       : 31
#      BUFE                        : 31
# IO Buffers                       : 9
#      IBUF                        : 3
#      OBUFE                       : 6
=========================================================================
```

Fig. 17 The design summary results of Xilinx ISE 9.1i.

ple tags environment, the complexity and performance measure of server-side and procedure is virtually no different than in the single tag environment.

4.5 Comparison with Other Existing Key Agreement Protocols in RFID Field

All other existing key agreement protocols in RFID field have to use either special "noisy tags" [7], or a hash function in addition to XOR operation, PRNG, etc [8], or require many interactions between the tag and the reader [9]. Our BKAP and EBKAP are the only binary tree based KAPs, which avoid using special tags, hash functions, or excessive interactions between the tag and the reader. Since none of the existing KAPs gave the cost of realizing their protocols in terms of the number of gate logics like us, therefore we can't make any meaningful quantitative comparison between their results and ours.

5. Conclusions and Future Works

Although RFID technology can be applied in many fields [13], such as asset management, tracking, matching, process control, access control and automated payment, these applications will result in more security and privacy risks. Many proposals have been proposed for solving these security and privacy risks, however, most of them are impractical. This is because some of them were designed for protecting long-term security using public key cryptosystem but do not consider the hardware cost [14], or some of them were designed using symmetric key cryptosystem but do not consider the privacy or the whole environment security [15]. In this paper, we propose a generic binary tree based KAP, and then apply it to RFID system for symmetric session key exchange.

The major advantages of our EBKAP are the low time complexity $O(n)$, and the low space complexity $O(n \log n)$. We also estimate the number of the logic gates required to implement EBKAP in a RFID tag by running a Xilinx ISE

9.1i Verilog Hardware Design Language (Verilog HDL) program. Our estimating result shows that it only takes 3,474 logic gates, 790 FlipFlops, 31 tri-states, and 9 IO Buffer to implement EBKAP. We believe the number of the logic gates can be optimized further to be less than these estimated figures. Furthermore, we do not require a backend EPC/master key database for implementing either BKAP or EBKAP. That is to say, we only make use of the existing resources in the RFID reader and tags to provide security and privacy to the low cost RFID systems. The EBKAP is different to the "One Time Codes" scheme in [15] because they need to pre-store the shared keys in their databases and they also need to worry about the key synchronization problem.

The BKAP and EBKAP, also have vulnerabilities. The number of the RIS_t depends on the NN_s, this may be weak when the NN_s is too small. So we suggest the service provider select B_{n_i} with a large NN_s. In addition, we also suggest that RFID systems implemented with BKAP or EBKAP should be developed in accordance with the National Institute of Standards and Technology (NIST) FIPS 140-2 Cryptographic Module Security Requirements [16] for protecting them from tampering and other physical attacks.

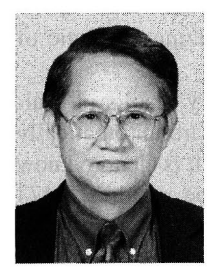
Although EBKAP is intended to protect the privacy and traceability when RFID is deployed in real life environment, the same concepts adopted in EBKAP can be extended and applied to develop two other similar protocols [17]. The first protocol is EPC retrieval protocol (ERP) which could be used to protect the privacy with user anonymity, when a reader tried to retrieve a tag's EPC code. The second is a binary tree based mutual authentication protocol called BMAP, which has a similar purpose like ERP to protect user's privacy with user anonymity, because in BMAP a tag only responds a random value to the query of a reader after a proper mutual authentication between the tag and the reader.

Acknowledgement

This work was supported in part by National Taiwan University of Science and Technology, and in part by the National Science Council, R.O.C., under Grants 94-2218-E-011-011 and 95-2218-E-011-008.

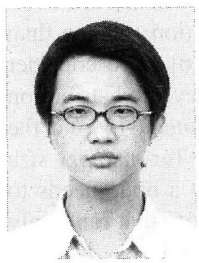
References

[1] R. Weinstein, "RFID: A technical overview and its application to the enterprise," IT Professional, vol.7, no.3, pp.27–33, 2005.
[2] K. Finkenzeller, RFID Handbook: Fundamentals and Application in Contactless Smart cards and Identification, 2nd ed., John Wiley & Sons, UK, 2003.
[3] M. Shimizu, M. Kobayashi, and M. Umehira, "Overview of RFID technologies for ubiquitous services," NTT Technical Review, vol.1, no.9, pp.12–18, Dec. 2003.
[4] M.J.B. Robshaw, "An overview of RFID tags and new cryptographic developments," Inf. Secur. Tech. Rep., vol.11, no.2, pp.82–88, 2006.
[5] R. Damith and C. Peter, "Security in low cost RFID," Technical Report Adelaide-AUTOID-WP-HARDWARE-027, Adelaide Auto-ID Center, Sept. 2006.
[6] W. Stallings, Cryptography and Network Security Principles and Practice, 3rd ed., Pearson Education, USA, 2004.
[7] C. Castelluccia and G. Avoine, "Noisy tags: A pretty good key exchange protocol for RFID tags," Int. Conf. Smart Card Research and Advanced Applications—CARDIS (Lecture Notes in Computer Science), Springer-Verlag, Berlin, Germany, 2006.
[8] L. He, Y. Gan, N.N. Li, and Z.Y. Cai, "A security-provable authentication and key agreement protocol in RFID system," WiCom 2007, pp.2078–2080, Sept. 2007.
[9] H. Chabanne and G. Fumaroli, "Noisy cryptographic protocols for low-cost RFID tags," IEEE Trans. Inf. Theory, vol.52, no.8, pp.3562–3566, 2006.
[10] EPCglobal Inc., "Class 1 generation 2 UHF air interface protocol standard version 1.09," Available at http://www.epcglobalinc.org/standards/uhfc1g2/
[11] E. Horowitz, S. Sahni, and D. Mehta, Fundamentals of Data Structures in C++, pp.246–329, Computer Science Press, New York, 1995.
[12] B.M. Mireille, "Sorted and/or sortable permutations," Discrete Mathematics, vol.225, no.1, pp.25–50, 2000.
[13] T. Karygiannis, B. Eydt, G. Barber, L. Bunn, and T. Phillips, "Guidance for securing radio frequency identification (RFID) systems (Draft)," Special Publication 800-98, Recommendations of the National Institute of Standards and Technology, Sept. 2006.
[14] A. Juels, R.L. Rivest, and M. Szydlo, "The blocker tag: Selective blocking of RFID tags for consumer privacy," Proc. ACM Conf. Computer Commun. Secur., pp.103–110, Oct. 2003.
[15] R. Ghosal, M. Jantscher, A.R. Grasso, and P.H. Cole, "One time codes," Technical Report Adelaide-Autoid-WP-Hardware-030, Adelaide Auto-ID Center, Sept. 2006.
[16] National Institute for Standards and Technology (NIST), "Security requirements for cryptographic modules," FIPS PUB 140-2, May 25, 2001.
[17] S.-H. Chen, A New Generic Binary Tree Based Key Authentication Protocol to Protect The Security and Privacy of the Low Cost RFID Systems, Master Thesis, Department of Electrical Engineering, National Taiwan University of Science and Technology, July 2007.



Albert Jeng received his B.S., M.S. and Ph.D. degrees all in Electrical Engineering from National Taiwan University, University of Idaho and University of Maryland respectively. He has served as an information security consultant in USA and several other Asia Pacific countries, concentrating on the protection of their national information infrastructure (NII) for more than 25 years. He has extensive theoretical, practical, and teaching experience in trusted computer, cryptography, IT security evaluation, certification and accreditation, as well as network security areas. Currently, he is a full professor with the department of Computer Science and Information Engineering (CSIE) at The Jinwen University of Science and Technology (JUST), as well as an adjunct professor with the CSIE department at The National Taiwan University of Science and Technology (NTUST) teaching and performing research in RFID and Wireless Communication Security, Common Criteria Evaluation and FIPS 140-2 Validation, PKI and Healthcare Applications, and Cryptology areas.

Li-Chung Chang received the B.S. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 1990, the M.S. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, in 1994, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, in 2003. Currently, he is an Assistant Professor with the Department of Electrical Engineering at National Taiwan University of Science and Technology, Taipei, Taiwan. His research interests include power amplifier predistortion for wireless applications, UWB system architecture, and RFID security.



Sheng-Hui Chen received the B.S. degree in electrical engineering from the National Taiwan University of Science and Technology (NTUST) in 2004. He has graduated with a Master degree specialized in Computer & Communication at NTUST, and his master thesis is on the security and privacy of radio frequency identification (RFID).