Group-based key array authentication protocol in radio frequency identification systems

Yi Jiang¹, Wei Cheng¹, Xiaojiang Du²

¹School of Electronics and Information, Northwestern Polytechnical University, Xi’an, People’s Republic of China
²College of Science and Technology, Temple University, Philadelphia, PA, USA
E-mail: jiangyiv88@nwpu.edu.cn

Abstract: For the purposes of information security and privacy between readers and tags, identity authentication is a significant issue for radio frequency identification (RFID) systems. In this study, the authors propose a novel security group-based key array authentication protocol, which is suitable for a large scale RFID environment. Based on a key array, this protocol can generate an authentication key for each pair of reader and tag with lower storage. Adding an identifier update phase, they design the authentication process passing the formal analysis from GNY. The security and performance analysis results show that the protocol they present can achieve better security than previous protocols in resisting external and internal attacks, with lower storage and acceptable communication and computation load.

1 Introduction

Radio frequency identification (RFID) systems is a contactless automatic identification system that provide contact-free communication between a reader and a tag over a radio link. Recently, the wide deployment of RFID systems in a variety of applications has raised many concerns about security and privacy. RFID systems must ensure the security of communication data as well as solve the identity authentication issue among entities.

To resolve the security issue successfully, several authentication protocols have been proposed, most of which focus on the threats from the external illegal attackers, but ignore the attacks from the internal legal entities. Hash chain protocol [1], which uses two different hash functions to confirm identity, can be used to achieve increased security. Challenge-respond based RFID authentication protocol (HIDVP) [2] keeps track of its session number and is designed to prevent eavesdropping and replay attacks by diversifying values. Moessner and Gul [3] offered a high level of security through the combination of a random key scheme with a strong cryptography. Zhou et al. [4] proposed a lightweight anti-desynchronisation privacy preserving authentication protocol which is suitable for the low-cost environment. However, all the previous papers [1–4] do not consider the internal attacks. The method suggested by Karthikeyan and Nesterenko [5], based on simple exclusive-or (XOR) operation and matrix operation, cannot resist the external attacks. It does not support the authentication of multiple readers. Another method, used by Yang et al. [6], in which tags only have hash function and exclusive-or operation, can improve the abilities of the forgery and anonymity attacks. However, it was pointed out that the scheme cannot protect privacy. Chien and Chen [7] proposed a mutual authentication scheme appropriate for low cost tags, but it cannot resist denial of service (DoS) attacks if the synchronisation between the tag and reader is lost. Kolias et al. [8] improved the protocol [7] by enabling readers and tags to communicate securely and by providing resistance against DoS attack. Ding et al. [9] proposed a method which shares a key from key array to resolve the internal attacks issue, but they do not provide a proof for correctness. Key array authentication protocol (KAAP) [10] is an extension to the protocol [9], performing informal security analysis, but it cannot solve the internal attack in the same group of readers or tags, and does not include an update operation.

In this paper, we propose a novel security authentication protocol GKAAP (group-based KAAP), which is suitable for large scale RFID systems. It has the best capacity to authenticate communication. To deal with the internal attacks, we design an authentication key generation method using a key array based on [11], which differs from the ones proposed in [9–10]. Using this method, we define the authentication process to prevent both the external and internal attacks. Then, to increase the ability to protect against attacks, the pseudorandom identifier update method is used. Considering the correctness of our protocol, we use GNY logic [12] to carry out the formal analysis. The security and performance analysis results show that the protocol we present outperforms previous protocols in security with lower storage and acceptable communication and computation load, while also resisting various attacks stemming from both the same group and between different groups.

The remainder of the paper is organised as follows. We describe our proposed protocol in Section 2. Section 3 analyses formally the correctness of our protocol with GNY logic.
logic. Section 4 analyses the performance of our protocol. Finally, our concluding remarks are stated in Section 5.

2 Proposed GKAAP protocol

In order to resolve such security and privacy problems from both the external and internal attacks, especially in the same group, the GKAAP protocol is proposed. It is an extension to the protocol KAAP [10] which has a key in the same group to resist the attacks among groups.

In our protocol, two types of keys are used for encryption: the shared key and the authentication key. A unique shared key \( k_w \) is given to legal readers and tags preventing external attacks. A key array \( D \) composed of authentication keys in the DB is used in security authentication between internal tags and readers.

2.1 Generation of the authentication key based on group

Suppose that tags are expressed as \( T_i \), \( i \in \{0, 1, 2, \ldots, m\} \) and the readers are expressed as \( R_j \), \( j \in \{0, 1, 2, \ldots, n\} \). Then we define \( D \) as the key generation array, \( A \) as the generation array of tags with size \( m \times n \) and \( G \) as the generation array of readers with size \( n \times n \). Suppose that \( D = A \times G \), where the dimension of array \( D \) is \( m \times n \). The expressions of \( A \), \( G \) and \( D \) are

\[
A = \begin{bmatrix}
1 & a_1 & a_2 & \cdots & a_{n-1} \\
1 & a_2 & a_2 & \cdots & a_{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & a_m & a_m & \cdots & a_{n-1}
\end{bmatrix}
\]

\[
G = \begin{bmatrix}
g_1 & g_2 & g_3 & \cdots & g_n \\
g_2 & g_2 & g_3 & \cdots & g_n \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
g_{n-1} & g_{n-1} & g_n & \cdots & g_n \\
g_n & g_n & g_n & \cdots & g_n
\end{bmatrix}
\]

\[
D = \begin{bmatrix}
k_{i1} & k_{i2} & k_{i3} & \cdots & k_{in} \\
k_{i2} & k_{i2} & k_{i3} & \cdots & k_{in} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
k_{in} & k_{in} & k_{in} & \cdots & k_{in}
\end{bmatrix}
\]

\( T_i \) stores the \( i \)th row in \( A \) and \( R_j \) stores the \( j \)th column in \( G \). The authentication key is \( k_i \) in \( D \). As the generation of rows in \( A \) is regular, \( T_i \) can only store \( a_i \), which can decrease the storage space greatly. It is the same as \( R_j \) which stores \( g_j \). In database DB is used to prove the correctness of the authentication key by computing between \( T_i \) and \( R_j \). If some tags and readers are not permitted to communicate, the corresponding key in \( D \) is null. The generation of the authentication key between \( T_i \) and \( R_j \) is described by the following equation

\[
k_{ij} = 1 + a_i g_j + (a_i g_j)^2 + \cdots + (a_i g_j)^{n-1} \quad (1)
\]

Based on the different functions, all tags and readers are divided into \( S \) and \( T \) groups, respectively, in which \( T_i \) belongs to the \( i \)th tag group \( A_s \), \( s \in \{1, 2, \ldots, S\} \) and \( R_j \) belongs to the \( j \)th read group \( G_r \), \( t \in \{1, 2, \ldots, T\} \). Different read groups own relative independent authorities for each tag groups, using different authentication key array \( D_{st} \) in DB. If one of the reader groups is not permitted to access a tag group, the corresponding key array is non-existent in DB, so the authentication cannot be carried out.

Overall, the different tags in the same group have different authentication keys with the different readers in the same group, which can defend against the internal attacks in groups. By group classification, it is fit for multiple applications against unauthorised access. The authentication key must be computed with the corresponding \( a \) and \( g \) stored in the tag and reader, which can prevent losing keys if an attacker captures a tag and obtains all of its sensitive message, including the authentication key.

2.2 Group-based key array authentication protocol

In the initialisation of GKAAP, it will allot an ID to each tag and reader in order to make communication between the two more convenient. The true ID can be utilised by replay and tracking attacks, so we use pseudorandom identifier preID = CRC(ID) to substitute it.

In order to describe the authentication phases between a tag and a reader, we use \( T_i \) and \( R_j \) to illustrate the process of message exchanges. They are as follows:

1. Phase 1: \( R_j \) generates a random number \( r_{R_j} \), then concatenates \( r_{R_j}, \) preID \( \text{RJ} \), \( G_r \) and \( g_j \) to form \( r_{R_j} \parallel \text{preID}_R \parallel G_r \parallel g_j \). Encrypting using the shared key \( k_w \), \( R_j \) sends the message \( \{r_{R_j} \parallel \text{preID}_R \parallel G_r \parallel g_j \}_k_w \) to \( T_i \) as an initial query.

2. Phase 2: On receiving the query, \( T_i \) decrypts the ciphertext \( \{r_{R_j} \parallel \text{preID}_R \parallel G_r \parallel g_j \}_k_w \) using \( k_w \). When \( T_i \) obtains \( r_{R_j} \parallel \text{preID}_R \parallel G_r \parallel g_j \), it will do several jobs as follows:
   - \( T_i \) verifies \( R_j \) by checking \( G_r \) in the list of permitted access reader groups which is prestored in its memory. If it is valid, \( T_i \) will compute the authentication key \( k_{ij} \). Otherwise, it will stop the authentication process with an error code.
   - \( T_i \) uses \( a_i \) belonging to the tag group \( A_s \) and \( g_j \) to compute the authentication key \( k_{ij} \) with formula (1).
   - \( T_i \) generates a random number \( r_{T_i} \), and encrypts \( r_{T_i} \parallel r_{R_j} \) by \( k_{ij} \) to obtain \( \{r_{T_i} \parallel r_{R_j} \}_k_{ij} \). Then, \( T_i \) concatenates \( \text{preID}_T \parallel A_s \) and \( \{r_{T_i} \parallel r_{R_j} \}_k_{ij} \) to encrypt by \( k_w \).

\( T_i \) sends the message \( \{\text{preID}_T \parallel A_s \parallel \{r_{T_i} \parallel r_{R_j} \}_k_{ij} \}_k_w \) to \( R_j \) as an an response.

3. Phase 3: When \( R_j \) receives the response, it extracts \( \text{preID}_T \) and \( A_s \) from \( \{\text{preID}_T \parallel A_s \parallel \{r_{T_i} \parallel r_{R_j} \}_k_{ij} \}_k_w \) with \( k_w \). Then, \( R_j \) forwards \( \text{preID}_T \parallel A_s \) to DB for verifying the identity of \( T_i \) and obtains \( k_{ij} \).

4. Phase 4: When receiving \( \text{preID}_T \parallel A_s \), DB first extracts \( \text{preID}_T \) to verify whether the message was generated from a legal tag of the corresponding \( A_s \). If \( \text{preID}_T \) is correct, DB will deliver the corresponding authentication key \( k_{ij} \) to \( R_j \) from the key array \( D_{st} \).

When \( R_j \) receives \( k_{ij} \), it will check whether the current computed \( r_{R_j} \) equals the previous generated \( r_{R_j} \) in Phase 1. If the two values are identical, \( T_i \) will be successfully authenticated by \( R_j \). Otherwise, it will stop the authentication process with an error code.

Phase 5: Reversing \( r_{T_i} \), we obtain \( \text{preID}_T \parallel r_{T_i} \) encrypts \( \text{preID}_T \parallel r_{T_i} \) by \( k_{ij} \) and forwards the ciphertext \( \{r_{T_i} \parallel r_{R_j} \}_k_{ij} \) to \( T_i \). To avoid being intercepted directly by attackers from the latter part of \( \{r_{T_i} \parallel r_{R_j} \}_k_{ij} \) which will be used for spoofing, it chooses \( \text{preID}_T \) to transmit.
When receiving $\{r_T\}_{k_j}$, $T_i$ extracts $r_T$ by decrypting and reversing. $T_i$ checks whether the current computed $r_T$ equals the previous generated $r_T$ in Phase 2. If the two values are identical, $R_j$ will be successfully authenticated by $T_i$. Otherwise, it will stop the authentication process with an error code.

The authentication process is illustrated in Fig. 1, and the phases can be described as follows

$$P1(R_j \rightarrow T_i): \left\{ r_{R_j} \| \text{preID}_{R_j} \| G_i \| g_i \right\}_{k_i}$$

$$P2(T_i \rightarrow R_j): \left\{ \text{preID}_{T_i} \| A_s \| \left\{ r_{T_i} \| r_{R_j} \right\}_{k_i} \right\}$$

$$P3(R_j \rightarrow DB): \text{preID}_{T_i} \| A_s$$

$$P4(DB \rightarrow R_j): \left\{ k_{ij}, \left( r_{R_j} \right)_{\text{com}} \triangleq \left( r_{R_j} \right)_{\text{ori}} \right\}$$

$$P5(R_j \rightarrow T_i): \left\{ r_{T_i} \right\}_{k_{ij}} \| \left\{ r_{T_i} \right\}_{\text{com}} \triangleq \left( r_{T_i} \right)_{\text{ori}}$$

### 2.3 Updating for pseudorandom identifier

Using the same preID to replace a same tag ID or a same reader ID for long time is unsafe for RFID systems. Tags and readers can easily suffer from replay, tracking and forgery attacks. We will update preID for every tag and reader when they authenticate with each other. To maintain consistency, DB will store the new preID and the old preID simultaneously, which can prevent the DoS attack from not updating the preID simultaneously between a tag (reader) and DB. If an error occurred during updating the preID to a tag or a reader, DB will use the old preID to attend the authentication process.

The generation of the new preID of a tag is described by (2), and the generation of the new preID of a reader is described by the following equation

$$\text{preID}_{R_j(new)} = \text{PRNG} \left( \text{preID}_{R_j} \right)$$

PRNG is a unidirectional random number generator, which is irreversible. DB stores $\text{preID}_{R_j(new)}$, $\text{preID}_{T_i}$, $\text{preID}_{R_j(new)}$ and $\text{preID}_{R_j}$ simultaneously.

The new preID can be transmitted and verified during Phases 4 and 5. The authentication process of updating is described as follows

- **Phase 4**: DB will compute the $\text{preID}_{R_j(new)}$ and $\text{preID}_{R_j(new)}$. It will concatenate the corresponding authentication key $\text{preID}_{R_j(new)}$ and $\text{preID}_{R_j(new)}$. Then, DB will deliver $\left\{ \text{preID}_{R_j(new)} \| \text{preID}_{R_j(new)} \right\}_{k_{ij}}$ to $R_j$. $R_j$ checks whether the $\text{preID}_{R_j(new)}$ is updated. If $\left( \text{preID}_{R_j(new)} \right)_{\text{com}} \triangleq \text{PRNG} \left( \text{preID}_{R_j} \right)$, $R_j$ will update its preID.

- **Phase 5**: $R_j$ encrypts $\left\{ T_T \oplus \text{preID}_{R_j(new)} \right\}$ by $k_{ij}$ and forwards the ciphertext $\left\{ T_T \oplus \text{preID}_{R_j(new)} \right\}_{k_{ij}}$ to $T_i$. When receiving $\left\{ T_T \oplus \text{preID}_{R_j(new)} \right\}_{k_{ij}}, T_i$ extracts $\left( T_T \oplus \text{preID}_{R_j(new)} \right)$ by decryption and checks the correctness of the $T_T$ and its new preID. If $\left( T_T \oplus \text{preID}_{R_j(new)} \right)_{\text{com}} \triangleq \left( T_T \right)_{\text{ori}} \oplus \text{PRNG} \left( \text{preID}_{R_j} \right)$, $R_j$ will be successfully authenticated by $T_i$. Otherwise, it will stop the authentication process with an error code. $T_i$ will then update its preID by formula (2). DB, readers and tags store the same PRNG function.

The authentication processes 4 and 5 are illustrated in Fig. 2, and the Phases 4 and 5 can be described as follows

$$P4(DB \rightarrow R_j): \left\{ \text{preID}_{R_j(new)} \| \text{preID}_{R_j(new)} \| k_{ij} \right\}$$

$$(r_{R_j})_{\text{com}} \triangleq (r_{R_j})_{\text{ori}}, \left( \text{preID}_{R_j(new)} \right)_{\text{com}} \triangleq \text{PRNG} \left( \text{preID}_{R_j} \right)$$

![Fig. 1 Authentication process of GKAAP](image-url)
3 Formal analysis of authentication protocol with GNY logic

In this section, GNY logic is applied to analyse the design correctness of GKAAP. We will analyse our protocol using four steps: proposing the initial assumptions, setting up an ideal model, confirming the security goals and verifying by GNY logic rules [12]. Table 1 shows notations to facilitate the formal descriptions.

3.1 Initial assumptions

We assume that the following holds at the beginning of every run of the protocol.

1. The assumptions for $T_i$

   - $r_{T_i} \in T_i$, $\text{preID}_{T_i} \in T_i$, $A_s \in T_i$, $T_i = \Diamond \text{preID}_{R_i}$
   - $k_u \in T_i$, $T_i = \Diamond k_u$, $k_o \in T_i$, $T_i = \Diamond k_o$
   - $T_i = T_i \leftrightarrow R_j$, $T_i = T_i \leftrightarrow DB$

   These expressions indicate that: $T_i$ possesses $r_{T_i}$, $\text{preID}_{T_i}$, $A_s$ and $k_u$; $T_i$ believes that $k_u$ and $k_o$ are fresh and $T_i$ is entitled to believe that $\text{preID}_{R_i}$ is fresh; $T_i$ believes $k_u$ and $k_o$ are suitable secrets for $T_i$ and $R_j$; $T_i$ believes $k_o$ is a suitable secret for $T_i$ and DB.

2. The assumptions for $R_j$

   - $r_{R_j} \in R_j$, $\text{preID}_{R_j} \in R_j$, $G_t \in R_j$, $g_j \in R_j$, $R_j = \Diamond \text{preID}_{T_i}$
   - $k_u \in R_j$, $R_j = \Diamond k_u$, $R_j = R_j \leftrightarrow T_i$

   These expressions indicate that: $R_j$ possesses $r_{R_j}$, $\text{preID}_{R_j}$, $G_t$ and $g_j$; $R_j$ believes that $k_u$ is fresh and $R_j$ is entitled to believe that $\text{preID}_{T_i}$ is fresh; $R_j$ believes that $k_u$ and $k_o$ are suitable secrets for $T_i$ and $R_j$.

3. The assumptions for DB

   - $DB = \Diamond \text{preID}_{T_i}$, $DB = \Diamond A_s$
   - $k_o \in DB$, $DB = \Diamond k_o$, $DB = DB \leftrightarrow T_i$

   These expressions indicate that: DB possesses $k_o$; DB believes that $A_s$ and $k_o$ are fresh, and DB is entitled to believe that $\text{preID}_{T_i}$ is fresh; DB believes that $k_o$ is a suitable secret for DB and $T_i$.

3.2 Ideal model

According to the authentication phases, the formal messages delivered between $T_i$, $R_j$, and DB can be expressed as follows:

1. $T_i : \langle r_{T_i} \rangle_{k_s}, \langle \text{preID}_{R_i} \rangle_{k_s}, \langle G_t \rangle_{k_s}, \langle g_j \rangle_{k_s}$
2. $R_j : \langle \text{preID}_{T_i} \rangle_{k_s}, \langle A_s \rangle_{k_o}, \langle r_{T_i} \rangle_{k_o}, \langle k_o \rangle_{k_o}$
3. DB : $\langle \text{preID}_{T_i} \rangle_{k_s}, \langle A_s \rangle_{k_o}$
4. $R_j : \langle k_o \rangle_{k_o}, \langle \text{preID}_{T_i} \rangle_{k_o}, \langle \text{preID}_{R_i} \rangle_{k_o}$
5. $T_i : \langle r_{T_i} \rangle_{k_s}, \langle \text{preID}_{T_i} \rangle_{k_o}$
3.3 Security goals

The security goals for \( T_i, R_j \) and DB can be expressed as follows

1. \( T_i = R_j \iff r_{Ti} = r_{Rj} \), Applying rule T1: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\). From the assumptions for \( T_i \), it has \( r_{Ti} \in T_i \). Applying rule T3: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\). Then, applying rule P1: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\), we obtain \( r_{Ti} \in T_i \). As a consequence, \( T_i \) believes that \( r_{Ti} \in T_i \).

2. \( R_j \rightarrow T_i \rightarrow r_{Ti} \), Applying rule T4: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\). Then, applying rule P1: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\), we obtain \( r_{Ti} \in T_i \). As a consequence, \( T_i \) believes that \( r_{Ti} \in T_i \).

3. DB believes that \( R_j \rightarrow \preID_T \), Applying rule T5: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\). Then, applying rule P1: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\), we obtain \( r_{Ti} \in T_i \). As a consequence, \( T_i \) believes that \( r_{Ti} \in T_i \).

3.4 Verification

The security goals can be verified by the initial assumptions and GNY logic rules in the ideal model. The process is similar to the protocol [10].

1. For C(1): \( T_i = R_j \iff r_{Ti} \), From the assumptions for \( T_i \), it has \( r_{Ti} \in T_i \). Applying rule T1: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\). From the assumptions for \( T_i \), it has \( r_{Ti} \in T_i \). Applying rule T3: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\). Applying rule P1: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\), we obtain \( r_{Ti} \in T_i \). As a consequence, \( T_i \) believes that \( r_{Ti} \in T_i \).

2. For C(1): \( T_i = R_j \iff r_{Ti} \), From the assumptions for \( T_i \), it has \( r_{Ti} \in T_i \). Applying rule T1: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\). Applying rule P1: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\), we obtain \( r_{Ti} \in T_i \). As a consequence, \( T_i \) believes that \( r_{Ti} \in T_i \).

3. For C(2): \( R_j = DB \), From the assumptions for DB, it has \( DB = DB \), and the communication channel between \( R_j \) and DB is secure, so \( R_j \equiv DB \). Applying rule J1: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\). Applying rule P1: \((\preID_{r_{Ti}}) \iff (\preID_{r_{Rj}})\), we obtain \( r_{Ti} \in T_i \). As a consequence, \( T_i \) believes that \( r_{Ti} \in T_i \).
5. **Internal attack resistance**: The protocol from [10] can prevent internal attack from different groups of readers or tags. Readers and tags of the internal system impersonate another reader and tag from other groups, which can be discovered by DB quickly, because different groups use different \( k_{ij} \). This method can resolve internal attack between groups, but it cannot solve the internal attack in groups, which can be solved by having a different \( k_{ij} \) for each reader and tag in the same group. Each tag and reader has a different authentication key in our protocol, so our protocol can resist internal attack, not only among the different groups but also within the same group.

6. **Mutual authentication**: Our protocol provides mutual authentication between the communicating entities. The preID of reader and \( r_t \) are authenticated by the tag in Phases 1 and 5. The preID of tag, \( r_R \) and \( k_{ij} \) are authenticated by the reader with the aid of DB in Phases 3 and 4. The shared key \( k_u \) is mainly used to provide the confidential authentication to protect against the external attacks.

Table 2 summarises and compares the security analysis of various recent protocols as well as our novel protocol presented in this paper. It can be concluded that the proposed protocol provides a much higher level of security than the previous presented protocols. The compared protocols are typical and interrelated with GKAAP. It can resist both the external and internal attacks effectively.

### 4.2 Performance analysis

During the performance analysis, the length of keys and random numbers are ignored for the sake of simplicity. In our protocol, the access list of tag includes the reader group ID, the number of which is \( T \). The tag does not store the authentication key, but only \( a_t \), the number of which is also \( T \). Then, the tag must store its identifier, the length of which is \( L \). The storage of a tag is \( 2T + L \). The value \( T \) is lower than \( L \) which is also defined as the length of the access list in KAAP. The communication load and computation load is similar to the protocol presented in paper [10], but the delivery message increase \( L \) to update from a reader to a tag. A reader connects with DB directly, so its ability to transmit additional data are very strong. Adding the updating part, tags and readers must carry out random number operation (PRNG) to update preID in our protocol, so the computation load is \( 2(R + E) \). We compare performance between GKAAP and KAAP because KAAP [10] is most closely related to our protocol. Table 3 shows the performance comparison between KAAP and our protocol. From it, we can see that our protocol has lower storage requirements, and slightly higher communication load and computation load. PRNG has less cost, so computation load is acceptable. Hence, GKAAP can be used in the applications requiring high security with proper cost. Thus, the generation method of authentication key by key array in our protocol did not increase tag consumption, but unexpectedly reduced the consumption.

### 5 Conclusions

We describe a novel security authentication protocol GAKKP, which is suitable for large scale RFID systems. To strengthen the resistance to internal attack in groups, we propose a method to generate authentication key by key array, which can reduce storage of tags effectively. During the course of identity authentication, we introduce the identifier updating phase, which can enhance the performance to prevent external attacks. From GNY analysis, the design correctness of our protocol is proved. The security and performance analysis results show that the protocol we present can achieve better security than previous protocols with proper cost. Based on the comprehensive analysis and comparison, our protocol can be used in applications requiring high security.

### 6 Acknowledgments

The authors acknowledge support from the Nature Science Foundation of China (nos. 61301092, 61202394) and the Nature Science Foundation of Shanxi Province of China (no. 2012JQ8005). The authors would also like to thank all of the anonymous reviewers for their comments and suggestions.

### 7 References

Second Int. Conf. on Security in Pervasive Computing (SPC 2005), (LNCS: 3450), Berlin, 2005, pp. 70–84


