SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2016; **9**:4182–4191
Published online 14 August 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1597

RESEARCH ARTICLE

HEADA: a low cost RFID authentication technique using homomorphic encryption for key generation

Mohanad Dawoud* and D. Turgay Altilar

Department of Computer Engineering, Istanbul Technical University, Istanbul 34469, Turkey

ABSTRACT

The limited computational and memory resources available in the Radio Frequency Identification (RFID) tags constitute the essential challenge to find a technique that satisfies high security requirements. In this paper, security and privacy requirements for an RFID authentication system are defined. Although some of the conventional cryptographic operations provide these requirements partially, they are not considered as suitable solution for RFID applications because operations cost is high especially for RFID tags. An unconventional use of homomorphic encryption is proposed to provide low-cost security and privacy in this research. HEADA is proposed as a novel authentication technique, which consists of deployment and authentication process stages. The homomorphic encryption is used solely in the generation of keys during deployment stage. These keys are used by RFID tags to generate anonymous authentication keys during the authentication process using only integer addition operations. Moreover, some of the conventional approaches have to use a brute–force search in the server side to identify a tag. Unlike these techniques, HEADA enables the server to identify a tag only by a binary search. It is shown that HEADA is the only technique to satisfy all security and privacy requirements using low-cost operations in both tag and server sides. Copyright © 2016 John Wiley & Sons, Ltd.

KEYWORDS
RFID; authentication; security; low cost; homomorphic encryption

***Correspondence**
Mohanad Dawoud, Department of Computer Engineering, Istanbul Technical University, Istanbul 34469, Turkey.
E-mail: dawoud@itu.edu.tr

1. INTRODUCTION

Radio Frequency Identification (RFID) technology plays an important role in the automatic identification of objects such as cargos, vehicles, goods, and many different assets. A typical RFID system consists of three essential entities: servers, readers, and tags (Figure 1). The servers process data received from readers using stored databases. Readers are connected to the servers through fast and secure connection and used to transfer data between servers and tags. Tags are limited resources devices that store a unique ID as well as different data according to the application [1]. Readers communicate with the tags via wireless networks, which give them the capability of accessing data embedded in the tags without requiring direct contact or line-of-sight. The efficiency of using the RFID systems arose new requirements, such as security and privacy, which should be suitable to the architecture of the RFID technology, especially the limited resources in the tags.

1.1. The model for secure RFID authentication

The aim of the RFID authentication process is to identify and authenticate the tag in the server through insecure channels. Although the authentication process can be considered as an initial stage for a server to proceed with further communications to trace, read from, or write to the tag, our focus within the scope of this paper is only the authentication process of the tag in the server. The authentication protocol works as follows: As soon as a tag arrives at a proximity of a reader, it receives a query from that reader and responds with an authentication request. The reader forwards the request to the server to check it through fast and secure communication channel. Server authenticates the tag and sends a verification code to the tag through the reader. The tag verifies the server and sends a verification code to the server. Once the verification code from the tag to the server is received, the server verify the

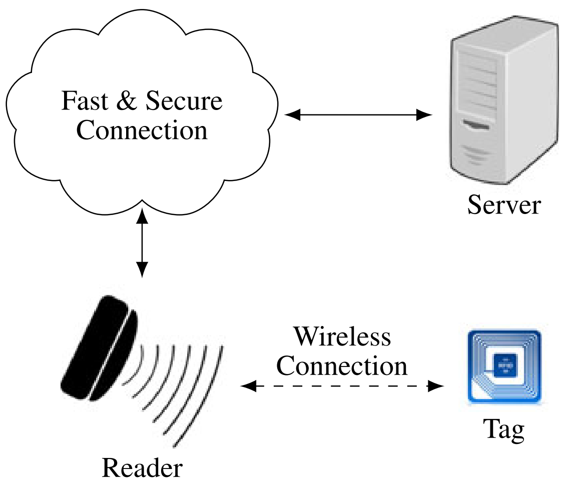


Figure 1. A typical model of radio frequency identification system.

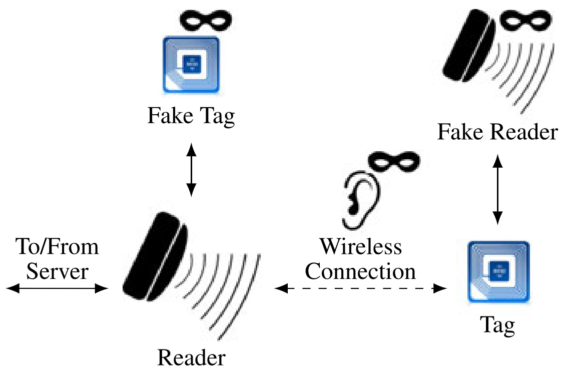


Figure 2. A typical security attacking model of radio frequency identification system.

tag and the protocol terminates. However, it is required to show that no attacker can identify or trace the tag at any time. Given a typical security attacking model for RFID system (Figure 2), an attacker is considered to be able to mimic a tag or a reader. It can also simply eavesdrop the communications between a legitimate tag and reader. The attacker could be either a single entity consisting of a fake tag, a fake reader and an eavesdropping module, or multiple entities encapsulating one or more modules. Therefore, following security requirements are considered as the minimum requirements to satisfy a high level of privacy and security in any RFID authentication protocol [1–4]:

(1) **Untraceability**: Only the authorized parties are allowed to identify or trace a tag at any time.
(2) **Resistance to replay attack**: The messages of any previous valid or invalid authentication processes cannot be used to gain a valid authentication for unauthorized party.
(3) **Resistance to DoS attack**: Modification, forging, blocking, or delaying of message from/to any party of the system is unable to make the tag or the server unreachable later under proper conditions.
(4) **Mutual authentication**: Both the server and the tag verify the proper identity of each other in the authentication process.
(5) **Tag forgery resistance**: Generating, predicting, or reusing a valid authentication key is infeasible without a valid authentication data.
(6) **Server forgery resistance**: Generating, predicting, or reusing a valid server verification key is infeasible without a valid authentication data.
(7) **Data recovery**: Any flaw in the synchronization or the status of any of the parties at a moment is recoverable.

1.2. Low Cost Operations

Radio Frequency Identification tags are renowned for being cheap, small, and made of sustainable materials with limited storage and processing capabilities. Chien [5] suggested categorizing the RFID tags into four categories according to the operations in the tag during the authentication process. The four categories are ultralight, lightweight, simple, and high cost tags. Therefore, efficiency requirements in this paper are defend as follows:

(1) **Low cost operations in the tag**: Tags belong to one of the first two categories, that is, ultralight or lightweight, which means including only OR, AND, XOR, CRC, or/and PRNG functions.
(2) **Low cost operations in the server**: No brute–force operations on the tags data (or part of it) during the authentication protocol.

Along with the seven security requirements, these two efficiency requirements will be noted as the 7+2 requirements in the rest of this paper.

1.3. Related works

There have been a number of techniques proposed to address some of the 7 + 2 requirements defined in Sections 1.1 and 1.2 . However, to the best of our knowledge, all of them fail to satisfy the complete set of the them. Weis *et al.* [6] and Kim [7] proposed two hash-based authentication techniques for RFID systems. The techniques suffer from eavesdropping, spoofing, and replay attacks. Moreover, the servers need to apply brute–force operations on all, or part, of the users to identify the communicated tags. Bringer *et al.* [8] proposed an extensions and improvements of Juels and Weis [9] technique. Both techniques use binary inner product to simplify the processes in the tag. However, they do not consider the server authentication, and they suffer from the traceability attacks as well. Yi *et al.* [4] proposed an improved technique to overcome some weaknesses in Chien and Chen [10] technique. However, the server in Yi *et al.* technique needs to apply brute–force operations to identify the communicated tags in each query. The same problem existed in the technique proposed in [11] where a variable c_i is updated in each authentication process and used as index to avoid the brute–force search in the server. For each tag, the brute–force search is supposed to be applied once in the first authentication process where c_i is initialized by zero. However, an attacker can continuously keep sending requests with $c_i = 0$, which may evolve into DoS attack. Moreover, c_i is sent in a clear format, which can be used by an attacker to trace a tag. The efficiency of the technique depends on the implementation of the Chebyshev polynomials in the tag [12]. Karthikeyan and Nesterenko [13] use matrix multiplication to generate the authentication messages between the server and the tag. The key of the tag is updated after each completed authentication process. The attacker can forge the identity of the tag by sending a query to it when it is not in the range of any reader. The tag replies by an authentication message as if the request came from a legitimate reader. If no response is received by the tag, it returns back to listening stage without updating the key. The attacker may either use the authentication message, which is not used yet in the server to authenticate himself there, or it may send another request to trace the tag because the tag replies by the same authentication message.

Symmetric and asymmetric encryption algorithms are used for authentication in many techniques [14–16]. In spite of the security capabilities of the encryption algorithms, they are considered costly on tags, which have limited computational resources.

The rest of the paper is organized as follows. Section 1.4 states the problem statement. Section 2 describes the proposed techniques in details. The security of this technique is discussed and compared with other techniques in Section 3. Section 4 concludes this paper.

1.4. The problem statement and contribution

The proposed technique is based on finding a set of sub-keys for each tag; these sub-keys are partitioned into groups and stored in the tag in the deployment course. The tag selects a number of combinations (combination, from now on, refers to a set of sub-keys composed of a single sub-key from each group) from these groups and adds them to generate a different authentication key in each authentication process. The server uses a reversibility property found in the sub-keys to reconstruct the used combinations from the authentication key to identify and authenticate the tag. Based on these assumptions, to achieve the seven security requirements defined in Section 1.1, the following conditions need to be achieved:

(1) **Uniqueness of sub-keys:** For all the tags, sub-keys are unique.
(2) **Uniqueness of authentication keys:** The summation of any combination of sub-keys should be unique.
(3) **Irreversibility:** Reversibility property should be restricted only to the server using constant time complexity algorithm and key data.
(4) **Randomness of the authentication key**: Authentication keys need to be selected randomly without duplication based on a key, known only to the tag and the server, and a simple algorithm.

Therefore, the contributions of this paper are defined to be:

(1) Developing a new methodology for finding a huge series of unique numbers using a relatively very small set of integers and a constant time algorithm (by utilizing integer addition). Moreover, only the parties that have a specific key data have to be able to identify the generator of each number using a constant time algorithm and simple binary search. Note that such numbers can be efficiently utilized in many applications such as privacy-preserving identification and temporary password generation.
(2) Developing a constant time algorithm for random selection without duplication from a set of elements.
(3) Applying the two methodologies given earlier to derive an RFID anonymous authentication technique that satisfies all the 7 + 2 requirements.

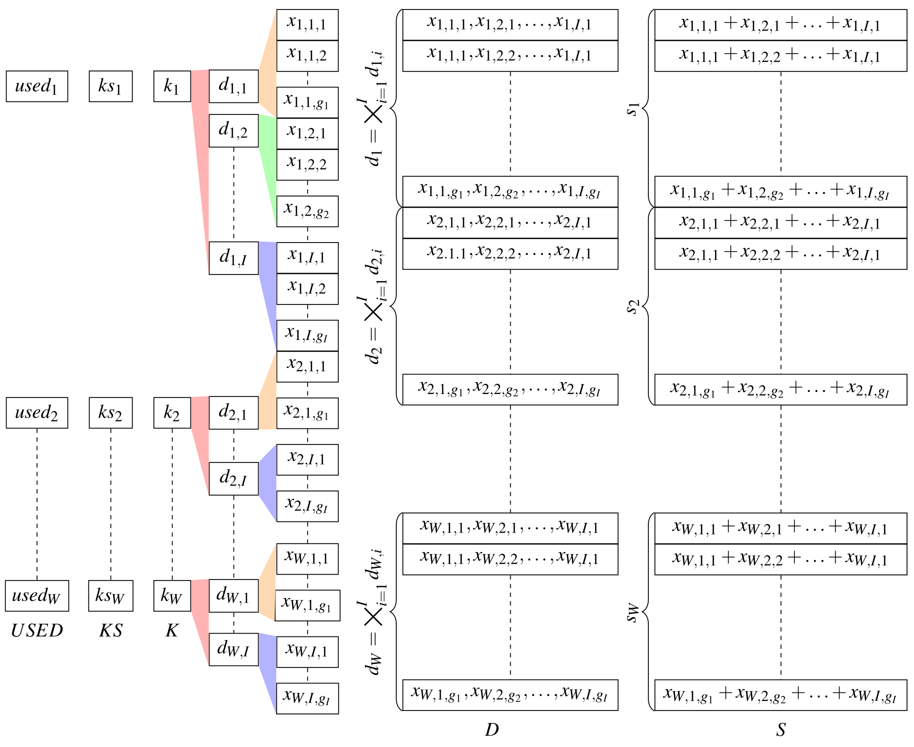
2. HEADA, THE PROPOSED TECHNIQUE

In order to explain the technique, the key components of the HEADA are explained in details in Section 2.1. Sections 2.2 and 2.3 explain how these components are generated to achieve these properties and requirements. The HEADA protocol is explained in Section 2.4.

2.1. Key components of the HEADA

The following are the main components of the HEADA along with their notations used in this paper (Figure 3):

- $T = \{t_1, t_2, \ldots, t_W\}$: The set of W tags considered in the algorithm.
- $K = \{k_1, k_2, \ldots, k_W\}$: The set of security keys of the tags in T.
- M: The number of sub-keys in a key k_n.
- I: The number of groups of sub-keys of a key k_n.
- $G = \{g_1, g_2, \ldots, g_I\}$: The sizes of the I groups of sub-keys of a key k_n. The set of G is the same for all the tags.
- $x_{n,i,j}$: The jth sub-key of the ith group of the key k_n.
- $d_{n,i}$: The set of sub-keys of the ith group of the key k_n.



$used_1$
ks_1
k_1
$d_{1,1}$
$d_{1,2}$
$d_{1,I}$
$x_{1,1,1}$
$x_{1,1,2}$
$x_{1,1,g_1}$
$x_{1,2,1}$
$x_{1,2,2}$
$x_{1,2,g_2}$
$x_{1,I,1}$
$x_{1,I,2}$
x_{1,I,g_I}
$x_{2,1,1}$
$x_{2,1,g_1}$
$x_{2,I,1}$
x_{2,I,g_I}
$used_2$
ks_2
k_2
$d_{2,1}$
$d_{2,I}$
$used_W$
ks_W
k_W
$d_{W,1}$
$d_{W,I}$
$x_{W,1,1}$
$x_{W,1,g_1}$
$x_{W,I,1}$
x_{W,I,g_I}
USED
KS
K
$d_1 = \times_{i=1}^{I} d_{1,i}$
$d_2 = \times_{i=1}^{I} d_{2,i}$
$d_W = \times_{i=1}^{I} d_{W,i}$
$x_{1,1,1}, x_{1,2,1}, \ldots, x_{1,I,1}$
$x_{1,1,1}, x_{1,2,2}, \ldots, x_{1,I,1}$
$x_{1,1,g_1}, x_{1,2,g_2}, \ldots, x_{1,I,g_I}$
$x_{2,1,1}, x_{2,2,1}, \ldots, x_{2,I,1}$
$x_{2,1,1}, x_{2,2,2}, \ldots, x_{2,I,1}$
$x_{2,1,g_1}, x_{2,2,g_2}, \ldots, x_{2,I,g_I}$
$x_{W,1,1}, x_{W,2,1}, \ldots, x_{W,I,1}$
$x_{W,1,1}, x_{W,2,2}, \ldots, x_{W,I,1}$
$x_{W,1,g_1}, x_{W,2,g_2}, \ldots, x_{W,I,g_I}$
D
s_1
s_2
s_W
$x_{1,1,1} + x_{1,2,1} + \ldots + x_{1,I,1}$
$x_{1,1,1} + x_{1,2,2} + \ldots + x_{1,I,1}$
$x_{1,1,g_1} + x_{1,2,g_2} + \ldots + x_{1,I,g_I}$
$x_{2,1,1} + x_{2,2,1} + \ldots + x_{2,I,1}$
$x_{2,1,1} + x_{2,2,2} + \ldots + x_{2,I,1}$
$x_{2,1,g_1} + x_{2,2,g_2} + \ldots + x_{2,I,g_I}$
$x_{W,1,1} + x_{W,2,1} + \ldots + x_{W,I,1}$
$x_{W,1,1} + x_{W,2,2} + \ldots + x_{W,I,1}$
$x_{W,1,g_1} + x_{W,2,g_2} + \ldots + x_{W,I,g_I}$
S

Figure 3. Key preparations and details for the HEADA.

- $d_n = d_{n,1} \times d_{n,2} \times \cdots \times d_{n,I}$: The set of all the combinations generated using the key k_n, where $\times$ indicates the Cartesian product.
- s_n: The set of summations of the combinations in d_n.
- $D = \{d_1, d_2, \ldots, d_W\}$.
- $S = \{s_1, s_2, \ldots, s_W\}$.
- $KS = \{ks_1, ks_2, \ldots, ks_W\}$: The set of selection keys of the tags in T. Selection key is used to determine the order of combinations selection in the tag t_n as shown in Section 2.3.
- $USED = \{used_1, used_2, \ldots, used_W\}$: The set of used combinations lists for the tags in T. $used_n$ list is used to store the used combinations for the tag t_n.

Assume that $W = 3$, $M = 8$, $I = 2$, and $G = \{4, 4\}$; Table I shows example values of K and KS and D, S, ID_n values.

In order to run the HEADA properly, the server keeps K, KS, ID, and K_H secret. Each tag t_n keeps k_n, ks_n, and φ (which is discussed in Section 2.3) secret. M, I, $G = \{g_1, \ldots, g_I\}$, m, and W are kept in both the server and the tags; they are also considered as public information, available even to attackers.

2.2. Key generation

Each key $k_n \in K$ consists of M sub-keys partitioned into I groups. The generation of these sub-keys is carried out in four steps as follows:

- **Step 1: Selecting parameters:** Parameters are selected to satisfy the needed level of security and efficiency.
- **Step 2: Raw sub-keys generation:** An algorithm that uses the concept of Segment:Offset is used to generate non-overlapped segments. Once segments are produced, each segment is re-partitioned into non-overlapped sub-segments recursively until sufficient number of groups of integers, that is, $A_1, \ldots, A_I$, can be produced. Given that in $(A{:}B)$, A indicates the segment and B the offset. Therefore, the range of the summations is partitioned into non-overlapped segments as $A_1{:}(A_2{:}(A_3{:}(\ldots{:}A_I)))$, where A_i is the set of raw integers that can be used in the ith group. Therefore, $A_1, \ldots, A_I$ can be generated in a recursive way as shown in Figure 4. Note that $Q_1, \ldots, Q_I$ are common divisors for the values in $A_1, \ldots, A_I$, respectively, which means that any value $\in A_i$ is a multiplication of Q_i. Moreover, all the values $\in A_{i+1}, \ldots, A_I$ as well as the summations of any combination composed of one number from each set are less than Q_i. For example, applying the algorithm in Figure 4 using the example in Section 2.1, where $W = 3$, $I = 2$, and $G = \{4, 4\}$ gives A_1 and A_2 as follows:

$$A_1 = \{13, 26, 39, 52, 65, 78, 91, 104, 117, 130, 143, 156\}$$
$$A_2 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$$

Given W, M, and the set $G = \{g_1, \ldots, g_I\}$, then, for any summation value of raw sub-keys combination $s = x_1 + x_2 + \ldots + x_I$, it can be reversed to get the raw sub-keys in two steps: (1) calculating the values of $Q_1, \ldots, Q_I$ as follows:

$$Q_i = \begin{cases} 1 & \text{for } i = I \\ Q_{i+1}\,(1 + (W^* g_{i+1})) & \text{for } 1 \leq i < I \end{cases} \quad (1)$$

Then, calculating $x_1, \ldots, x_I$ as follows:

$$x_i = \begin{cases} \left\lfloor \frac{s}{Q_i} \right\rfloor {}^* Q_i & \text{for } i = 1 \\ \left\lfloor \frac{s \bmod Q_1 \ldots \bmod Q_{i-1}}{Q_i} \right\rfloor {}^* Q_i & \text{for } 1 < i \leq I \end{cases} \quad (2)$$

For example, knowing that $W = 3$, $M = 8$, $G = \{4, 4\}$, suppose that the two raw sub-keys 117 and 2 are used, which means $s = 117 + 2 = 119$, then Q_1, Q_2, x_1, and x_2 are calculated as follows:

$$Q_2 = 1$$
$$Q_1 = 1^*(1 + (3^*4)) = 13$$
$$x_1 = \left\lfloor \frac{119}{13} \right\rfloor {}^*13 = 117$$
$$x_2 = \left\lfloor \frac{119 \bmod 13}{1} \right\rfloor {}^*1 = 2$$

Therefore, raw sub-keys satisfy the uniqueness of sub-keys and summations but do not satisfy irreversibility.

- **Step 3: Homomorphic encryption for irreversibility:** To make the reversibility property restricted only to the server, raw sub-keys are encrypted using homomorphic encryption with a key K_H. Homomorphic encryption is a form of encryption that allows computations to be applied on the encrypted data [17]. According to the homomorphic property, the summations of the combinations after encryption give a set of unique values if the original values do the same; this uniqueness is not guaranteed if the encryption algorithm does not have the addition homomorphic property. Therefore, the homomorphic property keeps the first two conditions satisfied. It also nonlinearly maps the raw sub-keys into an irreversible ones. For example, encrypting the values in A_1 and A_2 shown previously using the Algebra Homomorphic Encryption Scheme (AHEE) [17] using the AHEE homomorphic key set $\{p = 241, q = 197, x = 23, g = 13, r = 54, k = 68\}$ gives

$$A'_1 = \{88, 176, 23, 111, 199, 46, 134, 222, 69, 157, 4, 92\}$$
$$A'_2 = \{118, 236, 113, 231, 108, 226, 103, 221, 98, 216, 93, 211\}$$

Table I. Example values of K and KS and their D, S, and ID_n values for $W = 3$, $M = 6$, $I = 2$, and $G = \{3, 3\}$.

K			D	S	ID	KS
			{88, 236}	324		
			{88, 118}	206		
			{88, 98}	186		
		$x_{1,1,1} = 88$	{88, 221}	309		
		$x_{1,1,2} = 69$	{69, 236}	305		
	$d_{1,1}$	$x_{1,1,3} = 92$	{69, 118}	187		
		$x_{1,1,4} = 4$	{69, 98}	167		
k_1			{69, 221}	290	1	13
		$x_{1,2,1} = 236$	{92, 236}	328		
		$x_{1,2,2} = 118$	{92, 118}	210		
	$d_{1,2}$	$x_{1,2,3} = 98$	{92, 98}	190		
		$x_{1,2,4} = 221$	{92, 221}	313		
			{4, 236}	240		
			{4, 118}	122		
			{4, 98}	102		
			{4, 221}	225		
			{111, 226}	337		
			{111, 231}	342		
			{111, 108}	209		
		$x_{2,1,1} = 111$	{111, 216}	327		
		$x_{2,1,2} = 176$	{176, 226}	402		
	$d_{2,1}$	$x_{2,1,3} = 222$	{176, 231}	407		
		$x_{2,1,4} = 134$	{176, 108}	284		
k_2			{176, 216}	392	2	4
		$x_{2,2,1} = 226$	{222, 226}	448		
		$x_{2,2,2} = 231$	{222, 231}	453		
	$d_{2,2}$	$x_{2,2,3} = 108$	{222, 108}	330		
		$x_{2,2,4} = 216$	{222, 216}	438		
			{134, 226}	360		
			{134, 231}	365		
			{134, 108}	242		
			{134, 216}	350		
			{157, 93}	250		
			{157, 113}	270		
			{157, 211}	368		
		$x_{3,1,1} = 157$	{157, 103}	260		
		$x_{3,1,2} = 199$	{199, 93}	292		
	$d_{3,1}$	$x_{3,1,3} = 23$	{199, 113}	312		
		$x_{3,1,4} = 46$	{199, 211}	410		
k_3			{199, 103}	302	3	7
		$x_{3,2,1} = 93$	{23, 93}	116		
		$x_{3,2,2} = 113$	{23, 113}	136		
	$d_{3,2}$	$x_{3,2,3} = 211$	{23, 211}	234		
		$x_{3,2,4} = 103$	{23, 103}	126		
			{46, 93}	139		
			{46, 113}	159		
			{46, 211}	257		
			{46, 103}	149		

Suppose that the encrypted values are used, instead of the raw sub-keys in the same reversibility example shown previously. Note that the encryptions of 117 and 2 are 69 and 236, respectively, and $s = 69 + 236 = 305$. Applying the same reversibility process gives

```
Input: $W$, $I$, and $G = \{g_1, \ldots, g_I\}$
Output: $A_1, \ldots, A_I$

  $i \leftarrow 1$
  $\{Q_1, Q_2, \ldots, Q_I\} \leftarrow \{null, null, \ldots, null\}$
  $\{A_1, A_2, \ldots, A_I\} \leftarrow \{null, null, \ldots, null\}$
  $GETA(i)$
  return $A_1, \ldots, A_I$

  function GETA($i$)
    if $i = I$ then
      $A_i \leftarrow GETLIST(1, (W \times g_i), 1)$
      $Q_i \leftarrow (W \times g_i) + 1$
      return $Q_i$
    else
      $Q_i \leftarrow GETA(i+1)$
      $A_i \leftarrow GETLIST(1, (W \times g_i \times Q_i), Q_i)$
      return $(W \times g_i \times Q_i) + Q_i$
    end if
  end function

  function GETLIST($a, b, c$)
      return all integers $i$, where ($a \le i \le b \wedge i$ is multiple of $c$)
  end function
```

Figure 4. Algorithm to generate the $A_1, \ldots, A_I$ sets used in key generation.

$$x_1 = \left\lfloor \frac{305}{13} \right\rfloor * 13 = 299$$
$$x_2 = \left\lfloor \frac{305 \bmod 13}{1} \right\rfloor * 1 = 6$$

Decrypting 299 and 6 using the same key K_H gives 25 and 94, which are not found in any set of raw sub-keys. Therefore, encrypting the raw sub-keys satisfies the third condition: irreversibility.

- **Step 4: Keys assignment to the tags:** For each key k_n, the server selects randomly, without duplication, numbers from $A'_1, \ldots, A'_I$ sets and place them in the related groups of k_n until all the sub-keys are selected.

2.3. Sub-keys combination selection

Sub-keys combination selection technique should satisfy the following conditions:

(1) Each combination is selected only once.
(2) It has to be easy for the parties that have the selection key to find the order of the combinations.
(3) It has to be infeasible, if not impossible, for the parties that do not have the selection key to find or predict the order of the combinations.

Note that for any two natural numbers α and β where $1 < \alpha < \beta$ and $gcd(\alpha, \beta) = 1$, the function

$$\theta(i) = (i + \alpha) \mod \beta \qquad \text{for } 0 \le i \le \beta$$

is one-to-one function and $\theta(i) \neq i$. The selection algorithm uses the function $\theta(i)$ to select the combinations in different order based on a selection key (ks_n) and maximum (max) values, which are α and β, respectively, in the function $\theta(i)$. Given W, I, and $G = \{g_1, \ldots, g_I\}$, to find the max value, the minimum number of binary bits that represent all the elements in each group is added in ω. Then,

$$max = (2^{\omega}) - 1 \tag{3}$$

To select a selection key for a tag t_n, the server selects randomly ks_n where $0 < ks_n < max$ and $gcd(ks_n, max) = 1$. Each tag also stores a number that indicates the last selected combination, which is denoted as φ. In each selection process, φ is updated as

$$\varphi = (\varphi + ks_n) \mod max \tag{4}$$

Then, it is used to select the next combination. The selection of the sub-keys of the next combination is carried out by initially splitting the binary bits of φ into I binary groups where the length of ith group is equal to the minimum number of bits that represent g_i. The selection operation is completed by using the divided bits as indexes for the sub-keys in the groups $g_1, \ldots, g_I$. Note that φ has to be set to zero in the deployment stage.

The sizes of the groups are selected to be an exponential of 2 in the HEADA, such as 4 where 3 is the optimum value as shown in Appendix A. In this case, the mod operation can be carried out by simple integer addition with ignoring any overflow that may result from this addition process, which simplifies the implementation of the algorithm in the tags. Therefore, based on Table I, the first combination that is selected in tag t_1 is $\{4, 118\}$. It is selected as follows:

$$\begin{aligned} \textit{Current state} &: \varphi = 0 \\ \textit{New state} &: \varphi = (0 + 13)\text{mod}15 = 13 \\ \textit{Splitting the binary bits} &: 13 \rightarrow 1101 \rightarrow \{11, 01\} \\ \textit{Indexing} &: x_{1,1,4} = 4,\ x_{1,2,2} = 118 \end{aligned}$$

The server checks the successiveness of the received combinations by creating an m-length linked list that contains all the received combinations. The *next* pointer of each node (except the last node) points to the node that has the combination expected next to the one in the current node based on the same selection algorithm and key ks_n of the identified tag t_n. If the linked list is defined without disconnections or loops for m different combinations, they are considered successive; otherwise, they are considered disconnected.

2.4. Authentication protocol

Figure 5 shows the authentication processes with an example values based on the values in Table I with $n = 1$ and $m = 2$. Following are the details of the processes to authenticate a tag t_n in the server:

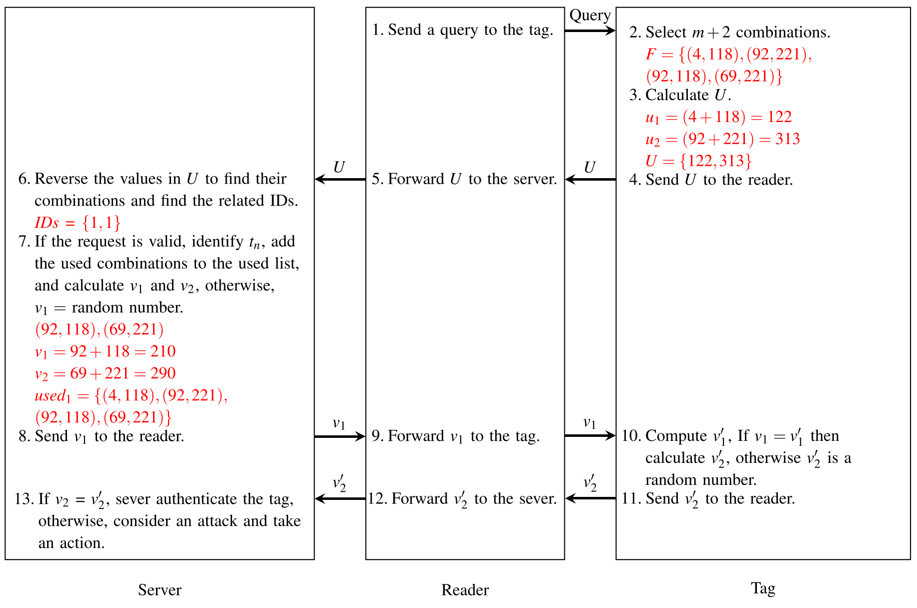

Server
Reader
Tag
Query
1. Send a query to the tag.
2. Select m+2 combinations.
F = {(4,118),(92,221),(92,118),(69,221)}
3. Calculate U.
u1 = (4+118) = 122
u2 = (92+221) = 313
U = {122,313}
4. Send U to the reader.
U
5. Forward U to the server.
U
6. Reverse the values in U to find their combinations and find the related IDs.
IDs = {1,1}
7. If the request is valid, identify tn, add the used combinations to the used list, and calculate v1 and v2, otherwise, v1 = random number.
(92,118),(69,221)
v1 = 92+118 = 210
v2 = 69+221 = 290
used1 = {(4,118),(92,221),(92,118),(69,221)}
8. Send v1 to the reader.
v1
9. Forward v1 to the tag.
v1
10. Compute v'1, If v1 = v'1 then calculate v'2, otherwise v'2 is a random number.
11. Send v'2 to the reader.
v'2
12. Forward v'2 to the sever.
v'2
13. If v2 = v'2, sever authenticate the tag, otherwise, consider an attack and take an action.

Figure 5. Sequence of operations and communications between the tag, the reader, and the server.

(1) The reader starts by a query to the tag t_n.
(2) The tag t_n selects a set of $m+2$ sequential combinations of sub-keys (F) from d_n using its selection key ks_n, so $F = \{f_1, \ldots, f_{m+2}\}$ where f_i indicates a single sequential combination.
(3) t_n calculates the summations of the sub-keys for the first m selected combinations and store them in U; therefore, $U = \{u_1, \ldots, u_m\}$ where u_i is the sum of the related combination, that is, f_i.
(4) t_n sends U to the reader.
(5) The reader forwards U to the server.
(6) The server (a) decrypts the numbers in U using K_H, (b) reverses the decrypted values to retrieve the selected combinations, (c) searches the keys list (K) for the retrieved combinations and returns the related IDs.
(7) The server checks whether (a) there is no ID' that appears m times or (b) the combinations, which are used to generate U are not sequential or (c) at least one of the combinations, which are used to generate U is used before.
If any of the aforementioned checks is correct, the server generates a random number for (v_1). Otherwise, the server considers ID' as an ID of a candidate tag subject to verification. The server (a) selects two combinations next to the ones used in U and calculate their summations (v_1 then v_2), and (b) adds the combinations selected for U as well as the ones selected for v_1 and v_2 to the related used list, that is, $used_n$.
(8) The server sends v_1 to the reader.
(9) The reader forwards v_1 to t_n.
(10) t_n calculates v'_1 in the same way as in the server using the m + 1th selected combination and compares it with v_1 value received from the reader. If $v'_1 = v_1$, then the tag authenticates the server and generates v'_2 using the $m+2$th selected combination. Otherwise, a random number is stored in v'_2.
(11) t_n sends v'_2 to the reader.
(12) The reader forwards v'_2 to the server.
(13) The server compares v_2 with v'_2. If $v_2 = v'_2$, then the tag is authenticated in the server. Otherwise, the server considers the process as an attack and takes an action.

Therefore, the authentication is carried out, and both the server and the tag verified each other.

3. EFFICIENCY AGAINST SOME KNOWN ATTACKS

The HEADA is verified to meet the 7 + 2 requirements as follows:
Tag forgery resistance: Without knowing the security and selection keys, it is infeasible for an attacker to find $m+2$ sequential summations ($u_1, \ldots, u_m$, v'_1, and v'_2) related to the same tag. The probability of being m + 2 randomly selected numbers sequential summations related to the same tag is P_{forge}, where

$$P_{forge} = \frac{(m+2)!}{(W*s)^{m+1}} \qquad (5)$$

For example, suppose that $W = 10^8$, $M = 160$, $I = 40$, $g_i = 4$ for $1 \leq i \leq I$, and $m = 5$, then $P_{forge} < 1.62 \times 10^{-189}$, which is considered negligible. On the other hand, without the selection key, an attacker is unable to predict the selected combinations because the selection

Table II. Average correlation value for randomly selected s_n values, where $W = 5$, $G = \{4, \ldots, 4\}$, and $M = 12, 16, 20$ and 40.

Key	$M = 12$	$M = 16$	$M = 20$	$M = 40$
k_1	0.174	0.081	0.029	0.006
k_2	0.117	0.068	0.031	0.005
k_3	0.241	0.084	0.011	0.001
k_4	0.096	0.032	0.008	0.008
k_5	0.204	0.065	0.022	0.003

is nonlinear. To show that the correlation between linearly selected combinations and the combinations selected using the selection algorithm and the selection keys are compared. For a key k_n, the average correlation value is calculated by partitioning the selected combinations into four equal parts. The correlation between each two parts is calculated separately, and the average of absolute correlation values is taken as the average correlation value of k_n. Note that s_n is partitioned into four parts according to the size of the sub-key groups, which cause the patterns to be repeated recursively number of times equal to this size if they are selected in linear order. The correlation value for the linear ordered s_n values is always 1. Table II shows the average correlation values of 5 tags for $M = 12, 16, 20,$ and 40, where $g_i = 4$ for $1 \leq i \leq I$. It shows that the selection algorithm removes any patterns in s_n where the average correlation values are getting close to zero as M increases, which prevents any patterns detection attacks.

Untraceability: The tag uses different authentications key in each query. It was shown in the discussion of the *tag forgery resistance* requirement that there is no detectable relation between any two or more authentication keys or authentication keys and specific tags, that can be used to trace or identify a tag.

Resistance to DoS attack: The tag does not use any transmitted data to update its key in each session. Also, it does not need to be synchronized with the server. Therefore, the attacker cannot make the tag unreachable by changing any transmitted data or by desynchronizing the tag and the server. Moreover, it is infeasible to use all the authentication keys that can be generated by a tag to make it exhausted. For example, suppose that $M = 160$, $m = 5$, $I = 40$, and $g_i = 4$ for $1 \leq i \leq I$, then the number of authentication keys that can be generated by a tag is $4^{40}/(5+2) \simeq 1.7 \times 10^{23}$. Knowing that a century consists of 3.1536×10^9 s, an attacker needs to make more than $(1.7 \times 10^{23})/(3.1536 \times 10^9) \simeq 5.3 \times 10^{13}$ queries per second for one century to use all the possible authentication keys, which is infeasible.

Server forgery resistance: v_1 is used to authenticate the server. It is shared only between the server and the related tag by sharing the authentication and selection keys. However, v_1 should be sequential to the related numbers in U. It was shown in the discussion of the *tag forgery resistance* requirement that it is infeasible for an attacker to find v_1 for U without the security and the selection keys.

Table III. Comparison between different tag authentication techniques.

Requirements	Chien [10]	P.Lopez [18]	Karthikeyan [13]	Chen [19]	Qingling [20]	Choi [21]	Sun [2]	Yi [4]	Weis [6]	Yeo [7]	Dixit [22]	Bringer [8]	Juels [9]	Akgun [11]	HEADA
1- Untraceability	·	√	√	√	√	·	√	√	∇	√	√	·	·	∇	√
2- Resistance to replay attack	√	√	·	·	√	√	·	√	·	·	√	∇	∇	√	√
3- Resistance to DOS attack	∇	·	·	√	√	√	√	∇	√	·	·	·	·	∇	√
4- Mutual authentication	√	√	√	√	√	√	√	√	·	·	√	·	·	√	√
5- Tag forgery resistance	·	√	√	√	·	√	√	√	·	·	√	·	·	√	√
6- Server forgery resistance	·	√	√	√	√	·	·	√	·	∇	∇	√	√	√	√
7- Data recovery	√	·	·	√	√	√	√	√	√	·	·	√	√	·	√
8- Operations in the server	·	√	√	·	·	√	·	·	·	·	·	√	√	·	√
9- Operations in the tag	√	√	√	√	√	√	√	√	·	·	·	√	√	·	√

√= provided ∇= partially provided ·= not provided

Resistance to replay attack: Any used session key cannot be used again because the server marks the used temporary keys. In case the attacker has got a session key using a fake query on the tag, he will not be able to authenticate himself in the server because he does not have the tag-to-server verification code (v_2). Moreover, any attempt to reuse a valid authentication will be detected by the server.
Data recovery: If an authentication process did not complete, the server and tag continue in the next authentication process without any problem because there is no need for synchronization between them.
Mutual authentication: v_1 is used to authenticate the server. U and v_2 are used to authenticate the tag. Therefore, both the server and the tag are authenticated.
Low cost operations in the tag: In each authentication process, the tag executes a constant number of integer addition operations for combinations selection and temporary key calculation, which has $O(1)$ complexity. Moreover, comparing the execution time of one SHA-250 hash function to 5×40 integer additions (which is the case in the HEADA example) shows that the execution time of the HEADA in the tag is approximately 8% of the SHA-250 hash function execution time in the same processor and under the same conditions. Moreover, the tag in the HEADA needs small memory to store the key. For example, if $M = 160$, $m = 5$, $W = 10^8$, $I = 40$, and $g_i = 4$ for $1 \leq i \leq I$, then the memory size required to store the key in the tag in the HEADA is 64 Bits $\times$ 160 Sub-keys = 1.25 KB, which is more than feasible compared with the memory size required to store the same number of the generated authentication keys, which is more than 104 Bits $\times$ $\left(4^{40}/(5+2)\right)$ Authentication keys $\simeq 2 \times 10^{12}$ TB.
Low cost operations in the server: The server decrypts the values received from the reader (U) and uses reversing algorithm to obtain the sub-keys. Both of these operations use constant cost algorithms. The server uses simple search for the sub-keys which in $O(log\,(W{*}M))$, which is more efficient than brute–force operations.

Table III compares the HEADA with some similar tag anonymous authentication techniques according to the 7+2 requirements.

4. CONCLUSION

In this paper, 7 + 2 requirements are defined. The 7 refers to the seven security requirements required to satisfy a highly secured RFID tag authentication process. The 2 refers to the two efficiency requirements related to the complexity of the operations used during the authentication process. Some of the techniques in the literature used simple operations in the tag or the server or both, but they did not achieve all the security requirements. On the other hand, some techniques achieved many security requirements using high complexity operations in the tag and the server. None of the discussed techniques achieved all the 7 + 2 security and efficiency requirements, which gives the proposed technique (called HEADA) the advantage over them. It was shown that the HEADA not only achieves the seven security requirements but also improves the way of working in both the server and the tag by using simple search, instead of brute–force search, in the server as well as using integer addition operation in both the server and the tag in the authentication process. These properties of the HEADA make it suitable to be applied efficiently in the current generations of RFID systems. We will use the technique for anonymous authentication in the mobile and cloud computing systems. It is also an essential part of our future work as it will be part of the complete privacy preserving data retrieval system that we build.

REFERENCES

1. Shih DH, Chin-Yi L, Lin B. Rfid tags: Privacy and security aspects. *International Journal of Mobile Communications* 2005; **3**(3): 214–230.
2. Sun HM, Ting WC. A gen2-based rfid authentication protocol for security and privacy. *IEEE Transactions on Mobile Computing*; **8**(8): 1052–1062.
3. Lee H, Kim J. Privacy threats and issues in mobile rfid. *The First International Conference on Availability, Reliability and Security (ARES 2006)*, Vienna, Austria, 2006; 510–514. doi: 10.1109/ARES.2006.96.
4. Yi X, Wang L, Mao D, Zhan Y. An Gen2 Based Security Authentication Protocol for RFID System. *Physics Procedia,* 2012: 1385–1391, doi:10.1016/j.phpro.2012.02.206.
5. Chien HY. Sasi: a new ultralightweight rfid authentication protocol providing strong authentication and strong integrity. *IEEE Transactions on Dependable and Secure Computing* 2007; **4**(4): 337–340.
6. Weis SA, Sarma SE, Rivest RL, Engels DW. Security and privacy aspects of low-cost radio frequency identification systems. In *Security in Pervasive Computing, Lecture Notes in Computer Science*, Vol. 2802, Hutter D, Mller G, Stephan W, Ullmann M (eds). Springer: Berlin Heidelberg, 2004; 201–212.
7. Yeo SS, Kim S. Scalable and flexible privacy protection scheme for rfid systems. In *Security and Privacy in Ad-hoc and Sensor Networks*, vol. 3813, Molva R, Tsudik G, Westhoff D (eds), Lecture Notes in Computer Science. Springer: Berlin Heidelberg, 2005; 153–163.
8. Bringer J, Chabanne H, Dottax E. Hb++: a lightweight authentication protocol secure against some attacks. *Second International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (SecPerU 2006)*, Lyon, France, 2006; 28–33.
9. Juels A, Weis SA. Authenticating pervasive devices with human protocols. In *Advances in Cryptology - CRYPTO 2005*, vol. 3621, Shoup V (ed), Lecture

Notes in Computer Science. Springer: Berlin Heidelberg, 2005; 293–308.
10. Chien HY, Chen CH. Mutual authentication protocol for rfid conforming to epc class 1 generation 2 standards. *Computer Standards & Interfaces* 2007; **29**(2): 254–259.
11. Akgun M, Bayrak AO, Caglayan MU. Attacks and improvements to chaotic map-based rfid authentication protocol. *Security and Communication Networks* 2015; **8**(18): 4028–4040.
12. Cheng ZY, Liu Y, Chang CC, Chang SC. Authenticated rfid security mechanism based on chaotic maps. *Security and Communication Networks* 2013; **6**(2): 247–256.
13. Karthikeyan S, Nesterenko M. Rfid security without extensive cryptography. *Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '05)*, ACM, New York, NY, USA, 2005; 63–67.
14. Arbit A, Livne Y, Oren Y, Wool A. Implementing public-key cryptography on passive rfid tags is practical. *International Journal of Information Security* 2015; **14**(1): 85–99.
15. Canard S, Ferreira L, Robshaw M. Improved (and practical) public-key authentication for uhf rfid tags. In *Smart Card Research and Advanced Applications*, vol. 7771, Mangard S (ed), Lecture Notes in Computer Science. Springer: Berlin Heidelberg, 2013; 46–61.
16. Batina L, Guajardo J, Kerins T, Mentens N, Tuyls P, Verbauwhede I. Public-key cryptography for rfid-tags. *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops '07)*, New York, USA, 2007; 217–222.
17. Xiang G, Yu B, Zhu P. A algorithm of fully homomorphic encryption. *2012 9th International Conference on, Fuzzy Systems and Knowledge Discovery (FSKD)*, Sichuan, China, 2012; 2030–2033.
18. Peris-Lopez P, Castro JCH, Estevez-Tapiador JM, Ribagorda A. An ultra light authentication protocol resistant to passive attacks under the gen-2 specification. *Journal of Information Science and Engineering* 2009; **25**(1): 33–57.
19. Chen CL, Deng YY. Conformation of epc class 1 generation 2 standards rfid system with mutual authentication and privacy protection. *Engineering Applications of Artificial Intelligence* 2009; **22** (8): 1284 –1291.
20. Qingling C, Yiju Z, Yonghua W. A minimalist mutual authentication protocol for RFID system & BAN logic analysis. *2008 ISECS International Colloquium on Computing, Communication, Control, and Management (CCCM 08),* vol. 2, 2008; 449–453.
21. Choi EY, Lee DH, Lim JI. Anti-cloning protocol suitable to epcglobal class-1 generation-2 rfid systems. *Computer Standards & Interfaces* 2009; **31**(6): 1124 –1130.
22. Dixit V, Verma H K. *Singh AK Enhanced hash chain based scheme for security and privacy in rfid systems*, 2011.

APPENDIX A: OPTIMUM SIZES OF THE SUB-KEYS GROUPS

The set of G is the same for all the keys; therefore, the number of different combinations that can be generated from a key k_n ($size(s_n)$) is the same for all the tags. This number is referred as s in this appendix. For any value of M, the minimum value of I that maximize s should be selected to minimize the number of addition operations in each query in the tag side. Suppose that $g_i = g$ for $1 \leq i \leq I$, then

$$s = g^{\frac{M}{g}} \tag{6}$$

To maximize s, the first derivative of Equation (6) is taken and made equal to zero to find the optimum value of g

$$\frac{ds}{dg} = -\left(Mg^{\frac{M}{g}-2}\right)(\ln(g) - 1) = 0 \tag{7}$$

Solving Equation (7) for g gives $g = e$ where e is the Euler's number, which equals to 2.718281828. The optimum value of g would be the nearest integer value to e, which is 3. Therefore, the optimum value of I (I') with respect to M for $3 \leq M$ can be expressed as:

$$I' = \left\lfloor \frac{M+1}{3} \right\rfloor \tag{8}$$