RFID Mutual Authentication Protocols based on Gene Mutation and Transfer

Raghav V. Sampangi, and Srinivas Sampalli

Original scientific paper

Abstract: **Radio Frequency Identification (RFID) is a technology that is very popular due to the simplicity in its technology and high adaptability in a variety of areas. The simplicity in the technology, however, comes with a caveat – RFID tags have severe resource restrictions, which make them vulnerable to a range of security attacks. Such vulnerability often results in the loss of privacy of the tag owner and other attacks on tags. Previous research in RFID security has mainly focused on authenticating entities such as readers / servers, which communicate with the tag. Any security mechanism is only as strong as the encryption keys used. Since RFID communication is wireless, critical messages such as key exchange messages are vulnerable to attacks. Therefore, we present a mutual authentication protocol that relies on independent generation and dynamic updates of encryption keys thereby removing the need for key exchange, which is based on the concept of gene mutation and transfer. We also present an enhanced version of this protocol, which improves the security offered by the first protocol. The novelty of the proposed protocols is in the independent generation, dynamic and continuous updates of encryption keys and the use of the concept of gene mutation / transfer to offer mutual authentication of the communicating entities. The proposed protocols are validated by simulation studies and security analysis.**

Index terms: **RFID security, RFID authentication, mutual authentication, genetic mutation, encryption key generation and management**

I. INTRODUCTION

Radio Frequency Identification (RFID) technology is an emerging wireless technology that finds application in nearly all domains. Be it uniquely identifying objects in the retail industry, or tracking an object or an entity through a manufacturing line, or managing patients in healthcare, or the recent concept of “Internet of Things”, RFID has opened the doors for a wide range of applications.

An RFID system typically comprises of electronic circuits known as RFID tags, devices to read data on these tags known as RFID readers, and enterprise servers that store data about the object the tag represents. RFID tags store an identifier (a number) to uniquely identify the objects to which they are attached. To read the data on these tags, RFID readers transmit radio-frequency electromagnetic signals, which energize the tags and allow them to respond with the identifier. The readers then forward this information to the enterprise server requesting for more information about the object the tag represents. The server retrieves and forwards relevant information to the reader after validating the reader, either through wired or wireless networks [1].

RFID Tags can be broadly categorized as passive tags, semi-passive tags, and active tags [1]. This categorization is based on the availability of an on-chip power supply or battery, which either facilitate or not facilitate the tags to initiate communication with readers. Passive tags do not have an on-chip battery, which necessitates the reader to initiate communication. Active tags, on the other hand, have an on-chip battery, which allows them to initiate communication with readers as well as respond to requests from them. Semi-passive tags have an on-chip battery, however, requiring energy from readers to broadcast their message.

Passive RFID tags are typically required to perform one basic function — respond to queries by any readers, and when required, perform data update tasks as instructed by the reader (inherently by the enterprise server). However, they do not have adequate resources for performing sophisticated authentication of the entity giving them the instructions. This makes passive RFID tags vulnerable to a range of attacks such as replay attack, tag killing, tag over-writing, etc. Furthermore, readers with high signal strength can also be used as “rogue” readers, can read information from any tag, even if separated by a large distance. This further increases the vulnerability of the tags, increasing doubts in their widespread acceptance.

Research on strengthening data privacy and security of RFID tags has therefore, assumed focus in recent years. Existing work has focused extensively on using pre-shared secret keys and performing simple bitwise operations such as XOR (exclusive-OR) to perform symmetric encryption and authentication. It has to be noted that the reason for RFID tags to use encryption is to authenticate entities, as significant information is stored securely in the server. However, one thing to note is that the tag, even with the capability to perform minimal computations, does not authenticate either the reader or the server, or does so in very a trivial manner. The reader queries themselves do not pose much threat to tags.

Manuscript received December 15, 2012; revised February 22, 2013.

This work has been funded by the Boeing Company.

Authors are with the Faculty of Computer Science, Dalhousie University, Canada. E-mails: raghav@cs.dal.ca, srini@cs.dal.ca.

However, since readers are a medium for the server to communicate important updates, such as security key updates, from the enterprise server, it becomes a necessity for the tag to validate (or, authenticate) the server. This is based on the premise that even though there can be rogue RFID readers, they cannot extract any valuable information from the tags themselves, since all important information is stored in the server. We present a mutual authentication protocol that focuses on authentication between the tag and the server, using mechanisms to dynamically update the encryption key independently at both the tag and the server.

The mechanism used to update the encryption key introduces an inherent authentication feature. This protocol is based on the concept of genetic mutation and gene transfer. Gene transfer is the process of propagation of characteristics or genes from one generation of an organism to the next. Mutations are changes introduced in this set of characteristics or genes. Our protocol is based employing this concept for generation of encryption keys, and to provide authentication as an implicit feature. We then present an enhancement to this protocol, which further increases the security offered. Prior research has examined the need for key exchange messages over a wireless channel to be authentic [2][3], and many of the RFID protocols use key management / encryption for authentication. Our protocols remove the need for key exchange messages by introducing independent key generation, and provide an inherent mutual authentication feature. The presented protocols are validated using key similarity analysis, complexity evaluation and security evaluation.

We hypothesize that:

H1: The encryption keys are updated with every instance (or, every communicated frame); and,

H2: The encryption keys generated for each frame will be least similar to each other (with similarity quantified by a number in the range 0–1).

The rest of this paper is organized as follows: we present the background and related work in the section II, followed by a description of the proposed protocols in section III. Following the description of the protocols, we discuss the methodology used for evaluating them and present the experimental results / analyses in sections IV and V, respectively. We then present a discussion of our work presented here, and discuss some benefits and challenges in section VI, following which we conclude the paper in section VII.

II. Background and Related Work

A. RFID Security

Security in passive RFID tag based systems is always a critical factor, since passive tags impose several resource and computational restrictions. This makes complex algorithms, which require a high degree of computation for achieving security. On the other end of the scale, simple algorithms may prove easier for an adversary to crack. Much of the current work focus on employing key updates being sent by the server, to synchronize with and update the keys in the tags. However, one scheme focuses on pseudonyms and focuses on updating the identity of the key with each query. We discuss these schemes in this section.

A protocol with the tags storing encrypted versions of their IDs, with their original IDs stored in the database on the server was proposed by Osaka et al. [4]. In this protocol, readers transmit a random number along with their queries, and the tags respond with a number comprising of the mathematically hashed value of the random number received XOR-ed with the encrypted ID. The reader forwards this combination along with the random number to the server, which authenticates the tag and releases information about the object the tag represents after validating the reader. The protocol can be configured to work either with or without a change in the symmetric key used for encryption by the tag. The scheme further supports ownership transfer, thereby supporting privacy protection. It is our understanding that the tag reply consisting of a constant entity (the encrypted ID) coupled with the server transmitting key updates to the tag would reduce the security offered by the scheme, as it opens up avenues to use cryptanalytic techniques to break the key sequence. If one key is retrieved, the succeeding conversations and thus, the system are vulnerable to attacks.

Osaka et al.'s work was slightly modified by Gui et al. [5], to support forward security and prevention of denial of service (DoS) attacks. Their protocol facilitates mutual authentication between the tag and the reader, with the reader / server considered as the same entity to illustrate secure communication channel between the reader and the server. They have introduced an additional XOR computation and hashing at the tag, to verify the reader, and have hence, updated the work proposed by Osaka et al. Their protocol generates number a, that is computed and sent by the tag (using the random number sent by the reader, similar to the Osaka scheme [4]), and numbers e and m, computed using the updated encrypted ID and a random number b. The reader verifies the tag using a, while the tag uses m to authenticate the reader, and e and m to update its encrypted ID. It has to be noted that with e, m and b being transmitted in the open, and the new key being a combination of the numbers so transmitted and the previously agreed key, the system could still be vulnerable to attacks if one of the keys are recovered by standard cryptanalytic techniques.

Yu et al. [6] proposed a protocol based on XTEA encryption, which addressed the issue associated with the unsecure radio frequency (RF) channel used for communication between RFID tags and readers. The nature of the RF channel makes communication vulnerable to attacks such as interception, data capture, data analysis, etc. Their protocol presents a way of encryption and exchange of messages using a non-volatile ID for the tag, a dynamically updated key set (128 bits). Their work is based on the assumption that the least significant 30 bits of a tag's ID can represent a tag uniquely. Server replies to each tag response with an acknowledgement, and instructs the tag to update its key set. For the purposes of authentication, least significant 30 bits of the tag ID are sent along with the message sent by the

server. It is to be noted that if the protocol reduces the number of bits that are significant for an authentication process (by employing the least significant 30 bits), it is also reducing the security and the uniqueness aspect, since it reduces the number of possible combinations of tag IDs.

The work proposed by Molnar et al. [7] employs the concept of pseudonyms and time limited delegation. The tag generates a different pseudonym, or a pseudo-random number, that enables the server to authenticate the tag. The server is referred to as a trusted center in the protocol and it delegates the responsibility of authenticating the tag to a reader by giving it a set of pseudonyms that it can use to verify the tag for a specified amount of time.

Their work has a central concept of "tree of secrets", where nodes of a binary secret tree has secret keys in each node. Each tag has a counter, which points to a leaf of the tree, which in turn represents a pseudonym. Therefore, a particular pseudonym can be used to represent one tag, depending on its present state and the pseudonym. This protocol necessitates the use of a trusted center, and assumes a protected channel of communication between the reader and the server, which might not always be the case.

A protocol for mutual authentication and privacy protection that conforms to EPC Class 1 Generation 2 stanard was proposed by Chen et al. [8]. This protocol necessitates a registration phase, where tags and readers have to register with the server, and a communication phase. The registration phase requires tags and readers to independently register with the server, which generates unique identifiers to represent their IDs. Further, each reader is assigned a set of tags, and it can only communicate with the assigned tags. This makes this protocol not employable in RFID systems with mobile readers, since such a system allows any reader to communicate with any tag.

During the communication phase of this protocol, tags use random numbers, XOR operations and CRC operations to authenticate the reader and vice versa. The protocol focuses on ensuring security and privacy by using a list of valid readers and tags, and by restricting the ability of readers to communicate with any tag in the system.

Vajda et al. [9] present multiple authentication protocols for RFID systems. One of their proposed protocols is based on a simple XOR operation that uses different encryption keys for securing the communication between tag and reader. The session keys are updated with each frame transmission. A block stream generator with a secret key is used to generate the session keys. The key used by the tag, however, remains a constant, which is also the seed used by the reader. The seed is either permuted or expanded by the block stream generator, and it uses recursive permutation of halves of the seed in case of the former. The resultant number is used as the updated key. The tag uses the seed it stores to verify the reader. In their analysis of lightweight authentication protocols, Defend et al. [10] critically analyze their work.

To summarize, we can say that most approaches require the server to perform key updates; all use random numbers, and one using time-limited delegation of responsibilities. One thing to note is that any approach that requires key updates to be performed by the server places makes the network vulnerable as there is a possibility, no matter how least likely, of an adversary cracking the updated keys and hijacking the sessions.

B. Biomimetics

Biomimetics is the use of concepts existing in biological sciences, such as the working mechanism of a neuron, the ascent of sap in plants and so on, in other fields such as engineering, robotics, electronics and so on to create new systems [11]. Traditionally, biomimetics has been used by several known people and organizations, such as Leonardo da Vinci (for his design of flying machines based on birds), Velcro (design derived from the hooked seeds of the burdock plant), anti-reflective surfaces (created using polythene sheets, based on insect eyes, wings and leaves of plants), and many more.

Although no framework exists to specifically use the concepts of biological sciences in other disciplines, one can choose to carefully understand the concept and design the system accordingly. The adoptability of any particular concept is however, subject to the prevailing conditions and requirements in the discipline where it is adopted.

III. PROPOSED PROTOCOLS

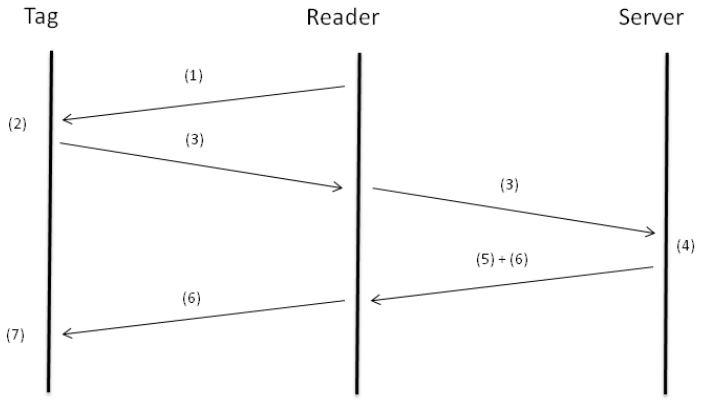
In this section, we describe the security protocol based on the gene mutation and transfer proposed in [12] [1] (Protocol A), and an enhanced version of the same (Protocol B).

A. Overview of Gene Mutation and Transfer

Deoxyribonucleic Acid (DNA) is the basic molecular structure in all living organisms, which contains genetic instructions that help in the organisms carrying out the various functions such as development and other activities. DNA is also responsible for propagating the genetic instructions from the parent generation to the progeny, thereby playing a significant role in the continuity of each species and in preserving characteristic elements (typically hereditary characteristics) of each species. The process by which genetic information is passed from one generation to the next is referred to as gene transmission [13][14].

There may be instances when genes alter, perhaps due to factors in the environment external or internal to an organism. In such cases, there is a very high probability that this alteration is passed on to the next generation, and for subsequent generations of the organism. Such alterations in genetic pattern are referred to as mutations. Let us consider the example of a pea plant. If there are seven peas in one pea pod, and one of them has a small genetic abnormality that has resulted in a dark brown patch on its surface, then, when this pea germinates and grows into a plant, the peas that grow from this plant will have a very high probability of having the brown patch. This depends on the characteristic gene being

[1] A preliminary version of this work has been published in the proceedings of the 2012 IEEE Symposium of Computational Intelligence for Security and Defence Applications (CISDA) [12].



(1) : Reader Query
(2) : $em = E_{ki}$ (ENC_ID)
$k_{i+1} = k_X$ or k_Y
$asv_{i+1} = h$ (numX ⊗ow $pattern_{asv}$)
(3) : *parent* || *generation* || *em* || asv_{i+1}
(4) : Regenerate *em'* using k_i and *ENC_ID*
if *em'* = *em*, tag is not invalid, then:
(4.1) Regenerate asv'_{i+1}
if $asv_{i+1} = asv'_{i+1}$, tag is authentic, then:
(4.2) Retrieve information about tag from database
(4.3) Update key, k_{i+1}
(4.4) Encrypt acknowledgement: E_{ki+1} *(ACK)*
(5) : Retrieved information about the tag
(6) : *Encrypted acknowledgement,* E_{ki+1} *(ACK)*
(7) : Tag: Regenerate encrypted acknowledgement, E'_{ki+1} *(ACK)*
if *E (ACK)* = *E' (ACK)* then:
(7.1) Server is valid

Figure 1. Overview of the proposed protocols.

either dominant (resulting in the patch being visible) or recessive (resulting in the patch not being visible).

Thus, in general, characteristics of a generation of any organism are transferred from one generation to another by means of gene transmission and the transferred characteristics may include abnormalities (or, mutations) as well.

B. Mutual Authentication Protocol based on Gene Transfer and Genetic Mutation (Protocol A)

This protocol mimics the concept of generations and genetic mutation that was described in the previous section. We consider its application on light-weight to heavyweight RFID tags, which have the capacity to perform minimally complex computations, such as the mathematical one-way hash function, and have sufficient storage capability.

Listed below are the assumptions of this protocol:

- RFID tags are initially loaded with the data by the owner. The owner of the tag is the organization where the tag will be deployed.
- The data contained on the tag is an encrypted identifier (ID) uniquely identifying the object it is associated with. For example, if 1234 is the ID associated with an object, the data stored on the tag will be ENCRYPTED (1234), encrypted using any standard encryption scheme by the enterprise server. We denote this as ENC_ID.
- We denote the ENC_ID subjected to one round of simple encryption as the encrypted message, *em*.
- The tag and the server share a pre-loaded 128-bit initial encryption key (IK).
- The tag and the server store states of the three (3) previously used keys in their memory to retrieve the previously saved synchronization state in case of dropped messages. These are referred to as the key states.
- Key states include the seeds of the random number generator, integer numbers (*parent*, *generation*) and the keys. These are required by the communicating entities to restore state in case of dropped messages, as will be explained in the sections below.
- The tag and the server store a pre-loaded 128-bit authentication-synchronization vector (ASV), which is used to authenticate the tag and synchronize with the server for the very first message. The ASV is unique for all tags deployed in the said environment, and is updated with every key update. The ASV helps in identifying loss of synchronization or any attempt of data modification.
- The authentication of readers with servers is beyond the purview of this work. The readers are assumed to use any standard authentication mechanism for this purpose.

Protocol A works as follows. The reader queries the tag, to which the tag responds with an encrypted version of ENC_ID. The ENC_ID is encrypted using the initial key (IK) to generate *em*. Following the encryption, the tag updates its encryption key to the new key (NK), and generates the ASV during the key update. The tag transmits a message that consists of two integers, *parent* and *generation*, which indicate the current state of the tag, the *em*, and the mathematically hashed ASV.

On receiving this message, the reader forwards it to the server. The server then authenticates the tag by first verifying the state of the tag using the parent and generation, then the encrypted message and finally the hashed ASV, and validates the tag. After the tag is validated (and implicitly, the reader is validated), the server retrieves and releases information about the object the tag represents to the reader, along with an encrypted acknowledgement (E_{ki+1}*(ACK)*) message. The key used for encrypting the acknowledgement is the updated key, which is generated at the server during the tag authentication and key update process. On reception of E_{ki+1}*(ACK)*, the tag authenticates the server, and performs key updates (if needed for synchronization) and is ready for the next query. This operation is summarized in Figure 1.

Updates to the encryption key and the ASV is achieved in our protocol using bitwise operations, which are used to mimic the action of genetic mutation and gene transmission across

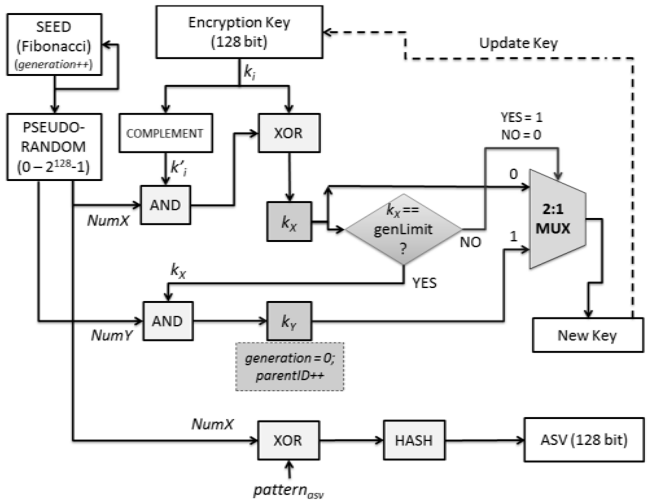


Figure 2 (a). Illustration of the working of the protocol (Protocol A)

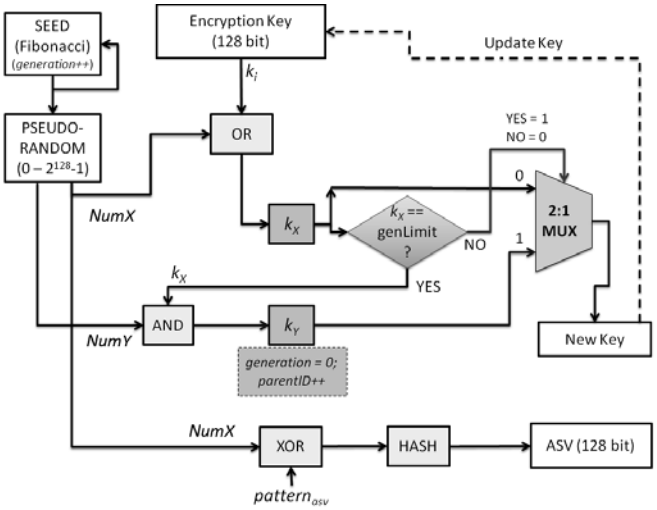


Figure 2 (b). Optimized version of Protocol A

generations of keys. Figure 2(a) illustrates the details of Protocol A. The operation to generate K_x in Figure 2(a) can be optimized and can be achieved with the help of one OR operation instead of a combination of XOR operations. This is illustrated in Figure 2(b). We have included Figure 2(a) to illustrate the concept of preserving the 1 bits of the parent key and applying mutations on the other bits. Note that the 2:1 MUX (the multiplexer) is an illustration for the logical operation to choose an appropriate key based on the condition being satisfied. The following paragraphs describe the scheme in detail.

B.1 Key Generation and Management

The novelty of our protocol is in using the concept of gene transfer and genetic mutation to generate the keys. In RFID systems, the simplicity of tags allows for adversaries to clone tags or to track tags using the responses. Some of the protocols described earlier [4][5][6] try to overcome this flaw by employing key refreshes and updates, i.e. the server transmits keys in either acknowledgement messages or explicit key refresh messages. On the other hand, the pseudonym protocol generates a new identifier (ID) using a generator that is synchronized with the server [7]. One aspect common to all these protocols is the transmission of keys, either from the server to the tag or from server to the reader.

In our protocol, we eliminate key exchange messages and key update messages. Using simple logical operations and pseudorandom numbers, we ensure that the tags and servers are able to:

- Generate keys independently
- Synchronize the keys
- Ensure mutual (or, bilateral) authentication

The tag uses the encryption keys so generated to further encrypt the already encrypted ID. It has to be noted that in RFID systems, encryption is used as a way of authenticating the entities, since RFID tags by themselves do not store any critical information. The tag and server are pre-loaded with an initial key (IK). When the tag is queried for the first time by a reader, the tag is energized and it encrypts ENC_ID to generate *em*. Following this, it initiates the process of generating the new key (NK).

New key generation process is as follows. The system first generates a mask of the IK, to preserve the pattern of its 1 bits by computing its one's complement. This is akin to the preserving the genetic characteristics of an organism. The tag has a synchronized pseudorandom number generator (PRNG) with the server, having a Fibonacci number as seed. A 128 bit pseudorandom sequence is generated by this PRNG, referred to as *NumX*. The protocol performs a bitwise AND operation on *NumX* with the mask to give a mutation pattern, which is a pattern of 1s and 0s (*NumX* ') to fill the 0 bits of the IK, without modifying the 1 bits. Finally, *NumX* ' is XOR-ed with the IK to generate K_X. This is analogous to applying mutation to an existing pattern. Please note that in an optimized version of the protocol illustrated in Figure 2(b), this set of operations to generate K_X can be achieved using one logical OR operation between the IK and *NumX*. The Fibonacci number is updated to obtain a new seed for the PRNG, in order to generate a new pattern for the subsequent communication.

By using a mask, we preserve the characteristics of the IK, which is similar to preserving the characteristics of each generation during gene transfer. By introducing a new pattern to occupy the 0-bit positions of the IK, we are introducing "mutations", and the K_X represents the "mutated" pattern. The variable *generation* keeps track of the modifications on the particular key.

The *genLimit* pattern is used to decide whether K_X will be used as the updated key. If K_X is different from *genLimit*, K_X will be used as the key for the next encryption cycle. In such a case, each key acts as the parent to the next key, and its bit pattern is preserved in the new key. When K_X becomes the same as the *genLimit*, the protocol decides in favor of abandoning K_X and generating a new - unrelated - key. In this case, the next usable 128 bit sequence from the PRNG is generated, and is AND-ed with K_X to generate K_Y. This is when a new parent key is generated, which is recorded in the key state being updated by incrementing the *parent* and

resetting the *generation* to 0. All subsequent new key generations will follow the same procedure, making the protocol generate a set of "generations" of keys for each parent key. The variable *parent* can be imagined as keeping track of the number of evolutions in the key generation cycle. An evolution in this context is when the key changes from its predecessor to an extent that the similarity is very low between them. This is analogous to evolution of species which means that the characteristics of one species changes to an extent where the changes are more than similarities between generations. Therefore, our protocol ensures that keys are generated in a manner that they are linked to the previously generated keys, demonstrating the concept of gene transfer and genetic mutation. This is a key step in ensuring mutual authentication between the server and the tag.

The tag (and the server) performs new key generation for every communication, i.e. every time a tag responds to a query (and every time a server authenticates the tag). The tag and the server synchronize their keys when the server sends the acknowledgement (described in detail in the synchronization section).

B.2 Data Encryption

As described earlier, the tag ID that is stored in the tag memory is already encrypted by the server, at the time of deployment. This is the first layer of security to protect the identity of the tag. However, we have to note that if this is not further encrypted, and if the encryption keys used for this encryption are not continuously updated, the tag is vulnerable to attacks such as tag tracking. In tag tracking, an adversary can track the presence of a particular tag among a set of other tags and / or objects. This would be a violation of privacy.

To protect the tag from such attacks, our protocol introduces an additional layer of simple encryption, with continuous key updates. This encryption process generates the encrypted message, *em*. Our protocol ensures that encryption keys are updated for every communicated frame, when the server and tag are synchronized. This ensures that the tag's susceptibility to tracking attacks is reduced significantly. Furthermore, it has to be noted that if the key update is continuous and stable, the encryption process can be simple, such as XOR, which would mean that resource utilization in RFID tags is at a minimum.

B.3 Synchronization

In our protocol, synchronization is an important activity, given that there are no key exchanges and that the server and the tag need to be in the same state to successfully authenticate each other. If the server and the tag states are not synchronized, it is highly likely that the protocol might fail due to de-synchronization. To avoid this, the protocol provides an inherent synchronization feature. This works as follows.

On receiving the reader query, the tag encrypts ENC_ID with the current key. Furthermore, the tag stores two immediate previous keys (prevKey1 and prevKey2), and their associated states (Fibonacci seed, parent and generation) in the tag memory. When the tag generates a new key, it replaces the oldest key in memory with the current key in the following manner - prevKey1 is assigned to prevKey2, currentKey is assigned to prevKey1 and the new key becomes the currentKey. On receiving the tag's response, the server authenticates the tag, updates its key, and responds to the reader with the tag information and an encrypted acknowledgement frame. The server also maintains three keys as in the tag.

Figure 3. Format of the tag's response to a reader's query

Synchronization is necessary because in a wireless environment, either of the following can occur:

- The tag's response to a reader query can be lost enroute to the server
- The server's acknowledgement can be lost enroute to the tag

Ideally, we would assume that there are no such losses. However, practical deployment environments for such systems may mean that some frames may be lost in transmission. However, even in such scenarios, we assume that even if frames are lost, they may not be lost more than three times consecutively, since the system would sense that there is either an attempt to desynchronize or an error in the channel. Hence, our protocol mandates that three keys (and their states) are stored by the tag (and the server).

Along with its response, the tag sends its ASV as the "*tagSignature*" as shown in Figure 3. On reception of this frame, the server first shortlists the set of tags with the identical *parent* and generation values, following which it attempts to generate sequences similar to the received *em* from these tag-states. If a match is found, it means that the key is within the three recently generated keys of one of these tags. The server then uses the ASV to find the exact tag in question. On reception of the encrypted acknowledgement, and synchronizes its key states.

With the server not having resource restrictions, our protocol places the onus on synchronization on it. Furthermore, the unlikely event that a frame is lost, may be frequent with the acknowledgement frame since a mobile reader may change its zones thereby making it difficult for it to follow-up on the successful reception of a forwarded acknowledgement. This may not be with the tag information request, since the reader needs this information and will query the server until it receives the response. Therefore, we can say that the states will be synchronized with almost every frame at the server, and if there is any de-synchronization, it might be at the tag. Therefore, our protocol requires the tag and the server to store three keys in memory for synchronization in such situations. The reason behind the number "3" is to give enough room for the tag-server pair to accommodate any lost frames, while not so as to facilitate repeat or replay attacks. When the server and the tag are synchronized, they retain only the synchronized key (and its associated state), and discards the rest.

B.4 Mutual Authentication

Mutual authentication is the central aspect of our protocol. In an RFID system, the tag needs to ascertain that its deployment environment is authentic before it can either respond to queries or perform key updates / refreshes as instructed by the reader, and the server needs to verify that all tags in a specific zone are authentic. For this purpose, our protocol provides a simple inherent mutual authentication process with the key generation mechanism.

The *tagSignature* (Figure 3) is used by the tag to authenticate the server. The ASV (which becomes the *tagSignature*) is generated as follows - the system first generates the XOR of *NumX* and *patternASV*, and performs a mathematical hash of this value. *patternASV* is a specific (constant) pattern that is chosen for the application by the server. This is used by the server to uniquely authenticate each tag. This ASV is sent as the *tagSignature* along with the encrypted message, *em*.

On receiving the data frame, the server uses the *parent* and *generation* to retrieve the value of the encrypted ID and the encryption keys of the tags, and regenerate *em*. The server may contain identical values for *parent* and *generation* for several tags. Therefore, it first retrieves the states of and generates *em* for them, and this serves as the first level of filter for tag authentication. If the generated *em* matches that sent by the tag, the server updates the encryption key. Then, it generates the ASV and compares it with the tag signature. If they match, it means that the tag is authentic. On authenticating the tag, the server synchronizes the values of the current *parent*, *generation* and the encryption key for the tag in question and discards the previous values. This is because once synchronized, the previous keys are not required by either the tag or the server, and any future query by a reader with one of the previous keys would imply that it is an attempt at a replay attack [15]. If however, there is any scenario when the received states match, but the ASV does not match, it implies that either there has been an attempt of changing the data, or that there has been an error in transmission.

On reception of the acknowledgement frame, the tag is momentarily energized. The tag regenerates the received frame using the acknowledgement pattern and keys stored in its memory. The tag stores the updated key and two previous keys and their respective states to re-synchronize in case of dropped frames. If the generated encrypted acknowledgement so generated matches the received acknowledgement, the server is authentic. If the key used to generate the pattern is not the same as the current key, it updates the key generator states with the corresponding values of *parent*, *generation* and Fibonacci seed. If this process fails, the tag will assume that the server is not authentic, and will not respond to any further queries.

C. Enhanced Mutual Authentication Protocol based on Gene Transfer and Genetic Mutation (Protocol B)

Although the protocol presented in the previous section is secure, is able to generate unique keys for almost all the communicated frames and facilitates mutual authentication between the communicating entities, the manner in which new keys are linked to their parent keys could be security vulnerability. This is because keys converge at the bit pattern of all 1s (the generation limit) before the parent key changes. This may not pose a severe security issue when this protocol is used as a standalone authentication protocol, however, since RFID systems (or any other system) might employ authentication to be a part of the key generation / management process, this is significant. Considering this issue, we present an improved version of the previous protocol in this section.

In this protocol, the manner of linking keys to subsequent keys has been modified and the generation limit has been changed to a count rather than a pattern to offer better security. Figure 4(a) shows working of the enhanced protocol. The operation to generate the New Key in Figure 4(a) can be optimized and can be achieved with the help of one OR operation instead of a combination of XOR operations. This is illustrated in Figure 4(b). We have included Figure 4(a) to illustrate the concept of preserving the 1 bits of the parent key and applying mutations on the other bits. The working of this protocol is as follows. In this protocol, we save an extra number in the memory, called parent key (*parentKey*). For the very first communicated frame, the *initial key* becomes the parent key, and the parent key is updated whenever a new parent is generated, i.e. the *parent* is updated. Furthermore, *generationLimit* is now an integer number between 0 and 4, which means that there can be a maximum of five generations per parent key. The next change defines the enhancement – we do not have linked "generations" of keys, but one parent key having either zero or multiple "children" keys. There can, however, be a maximum of *generationLimit* number of children for each parent.

After performing the simple encryption process to generate the encrypted message, *em*, this key generation module first generates a random number between 0 and 1.

- If the random number is 0, the system then proceeds to check if the number of children (indicated by the *generation* variable) for the current *parentKey* is five.
  - If the number of children is not 5, a new *child key* is generated as follows. The *parentKey* pattern is first preserved by inverting the bits and a mutation pattern is generated by AND-ing this pattern with a new random number (*n* bits for an *n*-bit key). Note that the generation of the mutation pattern is the same as the previous protocol. The mutation pattern is then XOR-ed with the *parentKey* to generate the new key. The *generation* variable is now incremented to update the number of children for this *parentKey*. The new key update and storage in memory then proceeds as in the previous protocol.
  - If the number of children is 5 (i.e. *generation* = *generationLimit* = 5) then, we force a *parentKey* change as discussed next.
- If the random number is 1, it means that we are forcing a *parentKey* change. In this case, the system generates a new random number (*n* bits) and XORs it with the current *parentKey* to generate the new *parentKey*. All future children (or generations of) keys will now use the new *parentKey.*

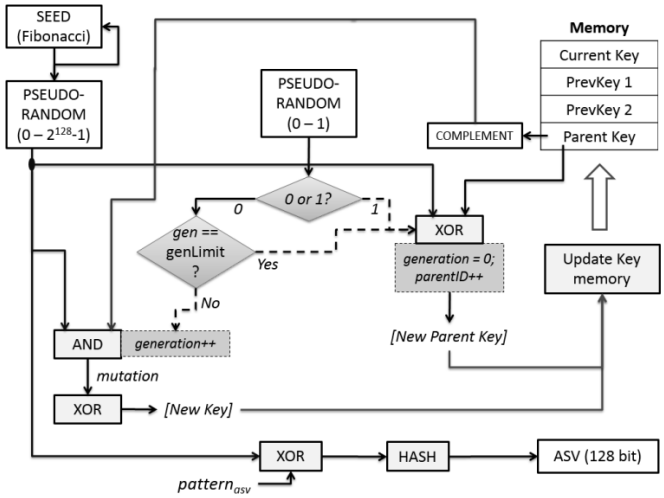


Figure 4 (a). Working of the Enhanced Mutual Authentication Protocol (Protocol B)

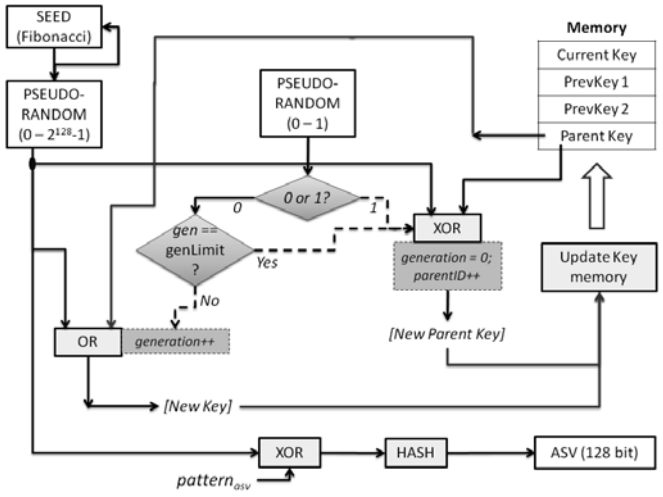


Figure 4 (b). Optimized version of Protocol B

By updating the protocol in this manner, we ensure that the keys do not converge to any specific pattern, that a unique encryption key will be generated for every frame (H1) and that consecutive keys will not be similar to each other (H2). Such an update also gives us the flexibility to use any algorithm for the seed generation, while the previous protocol required the use of only Fibonacci numbers (or an algorithm that combined numbers in a manner to Fibonacci number generation) as seed generators.

The enhanced version of the protocol significantly improves the security offered by Protocol A, improving its generality and enables its adoption in a variety of applications.

IV. Evaluation Methodology

We evaluated the protocols using a proof of concept implementation in Java. To validate our hypotheses, we performed similarity analysis between consecutive keys, examined the number of unique keys generated, and evaluated whether the keys would converge towards any specific pattern.

To analyze the similarity between pairs of keys, we considered each key and examined how similar it was to the previous. We compared the first key, however, with a pattern of all 1 bits (i.e. 128 or 64 bits of 1s), assuming that we start at a state of equal keys. The desirable behavior in this case is that each new key is highly dissimilar to the predecessor.

To quantify the similarity between keys, we computed the Sorensen's Similarity Index (SSI) [16], which is a measure of how similar the various pairs of keys are, and plotted the SSI for 500 frames. For each pair of keys, SSI is the ratio of twice the total similar characters in the two keys to the total size (in characters) of each key. Equation (1) presents the equation to compute the SSI.

$$SSI = \frac{2 \times n(A \cap B)}{n(A) + n(B)} \quad (1)$$

where, $n(A \cap B)$ represents the number of characters (or, numbers) in the key pair that are same, $n(A)$ and $n(B)$ represent the total number of characters (or, numbers) in each of the keys A and B of the key pair, respectively.

To analyze the stability of the scheme, and its ability to be generalized to any key size, we considered ten configurations, with different initial keys, different key sizes and different Fibonacci initial seed for the PRNG. This is described in Table I. For each configuration, we generated 500 keys, and computed the SSI for each pair of keys.

To analyze the security offered by the proposed scheme, we consider a security analysis. We consider five security goals associated with protecting data during communication, namely — confidentiality, integrity, authentication, non-repudiation and forward security [17]. We also present a qualitative analysis of the performance of the protocols with respect to known attacks categorized by Mitrokotsa et al. [15]. We also present a discussion on the behavior of the system in case of dropped frames and de-synchronization attempts.

TABLE I
TEST CONFIGURATION USED FOR ANALYSIS OF THE PROPOSED PROTOCOLS

Config. ID	Initial Key	Key Size	Initial Fibonacci Seed
C1	92eb8d6ecf7f808a705d1a4566991af0	128 bit	2178309
C2	12f03c157890a08a501d5d37bb10aae9	128 bit	2178309
C3	42f5876eaf9f8066b05ff140067b1a51	128 bit	2178309
C4	92eb8d6ecf7f808a705d1a4566991af0	128 bit	39088169
C5	12f03c157890a08a501d5d37bb10aae9	128 bit	39088169
C6	42f5876eaf9f8066b05ff140067b1a51	128 bit	39088169
C7	42f5876eaf9f8066	64 bit	2178309
C8	c157890a08a501d5	64 bit	2178309
C9	2f03c157890a08a5	64 bit	2178309
C10	42f5876eaf9f8066	64 bit	39088169
C11	c157890a08a501d5	64 bit	39088169
C12	2f03c157890a08a5	64 bit	39088169

Furthermore, we present a performance analysis for the protocols. Performance of an algorithm is mainly decided by the amount of system resources it utilizes. As part of analyzing the resources utilized by the proposed protocols, we consider an algorithm complexity analysis presenting the total number of arithmetic and logical operations that are involved. This will help in estimating the computational overhead, which will in turn impact the performance of the system, and hence, represent the time complexity of the system.

V. Analysis

A. Analysis of Similarity

To make data presentable and to help in analysis, we consider the first 100 frames of the results of configuration C1 for both protocols A and B.

Figure 5 shows the ability of protocols A and B to generate new keys for every frame. We observe that Protocol A generates 89 new keys, while Protocol B generates 100 new keys, in 100 frames. Over 500 frames, Protocol A generates 431 new keys, while Protocol B generates 500. It must be noted that the desired behavior for Protocol A was that the keys are similar to the previous key, because we want to mimic the gene mutation and transfer concept.

As an initial step, we chose to examine how the similarity increases until the keys converge to a pattern of all 1 bits (FFF…F – the *generationLimit*). However, in Protocol B, we introduced another random choice between to check if there will be a parent change and limited the number of child keys of each parent key to 5. The difference in the algorithms can be observed in the results.

Figure 6 (a) presents a plot of the SSI for the first 100 keys generated using Protocol A, while Figure 6 (b) presents the same plot for Protocol B. The differences in philosophies of the algorithms discussed in the previous paragraphs can be observed in these plots as well. We observe that the similarity increases until a point when the keys reach the *generationLimit* pattern in Protocol A, while the behavior is much more random in Protocol B.

TABLE II
Security Analysis of the Proposed Protocols

Security Goal	Brief Description	Status in proposed protocols
Confidentiality	Eavesdropping should not be fruitful	Supported, since encrypted ID is further encrypted
Integrity	Message received must be the same as message sent	Supported, since the encryption of the encrypted ID also acts as a message digest; only the tag with a relevant encrypted tag ID can send the appropriate message.
Authentication	Sender / receiver validation	Supported; both tag and server authenticate each other
Non-repudiation	Source cannot deny that the message was sent by it	Supported; with the keys being continuously updated, only a legitimate tag can send the valid encrypted message
Forward Security	Protection of previously transmitted data	Supported; keys are continuously updated and the synchronization of the initial keys are only between the server and the tag at the time of deployment.

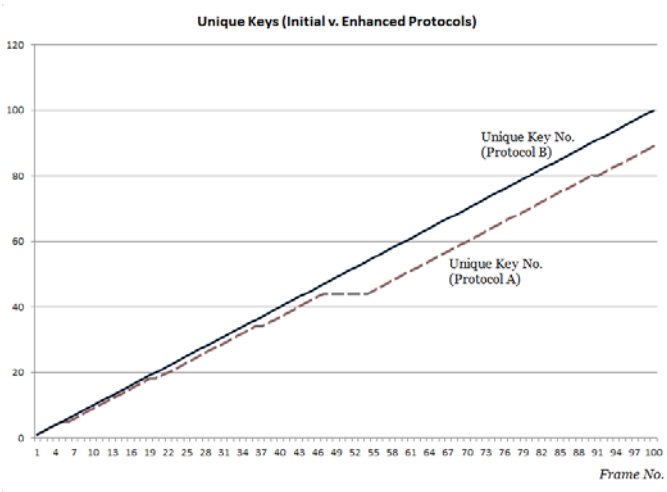


Figure 5. Unique key generation in the presented protocols

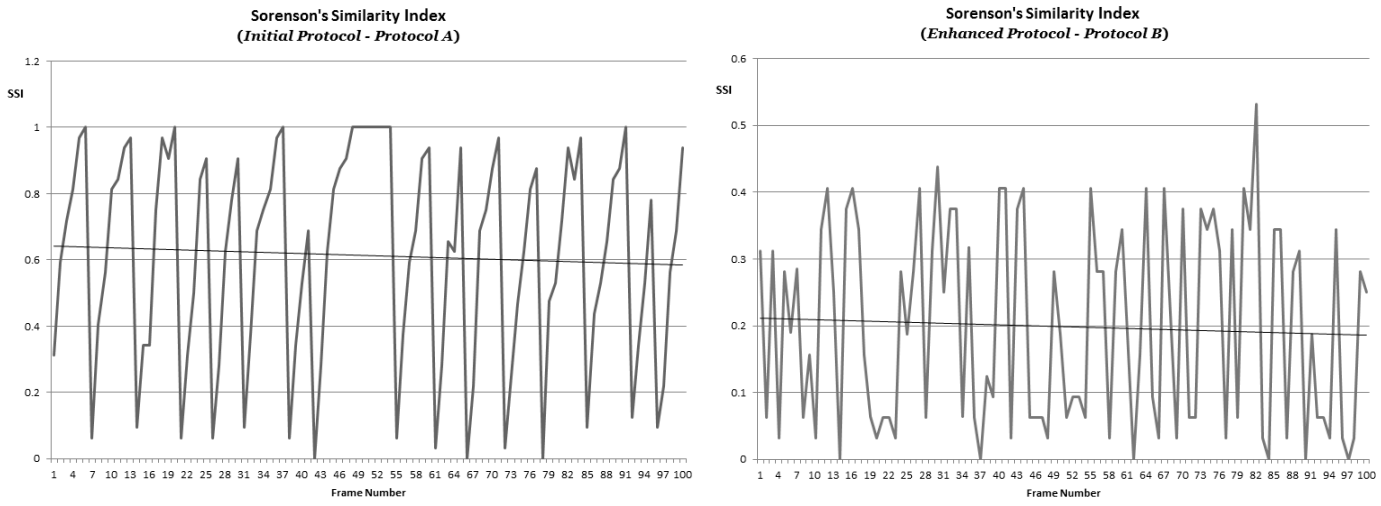
Our experiment revealed that this behavior is consistent across the other configurations, C2 – C12. From the results presented here, we can that hypothesis 1 was verified for Protocol B, as it was able to generate unique keys for every instance, while Protocol A generated 84% new keys on average for all configurations. Furthermore, hypothesis 2 is verified for both protocols as we are able to ensure that the generated keys are not similar to the previous keys. However, it has to be noted that because of the low values of similarity in Protocol B(average SSI value of nearly 0.199), it is more secure when compared to Protocol A (average SSI value of nearly 0.613).

B. Security Analysis

Table II summarizes the security analysis of the proposed scheme. Furthermore, the double encryption and continuously updated keys ensure that tags are secure from tracking by unauthorized readers. Note that the security analysis is the same for both protocols A and B; although due to an update to the mechanism of linking keys makes Protocol B fundamentally more secure than Protocol A.

Presented below is a discussion on the performance of the proposed protocols with respect to some of the known attacks. These are some of the attacks that are relevant in RFID systems, classified under various network attacks by Mitrokotsa et al. [15].

- *Eavesdropping*: An attempt to extract the tag ID will prove to be unsuccessful in the proposed protocols. This is because encryption keys are updated with each communicated frame. In Protocol A, even though we let the keys grow until they converge at a point where all bits are 1s, the protocol ensures that the parent keys and the linking between keys are regularly refreshed. This however is not an issue at all in Protocol B, where the linking between consecutive keys is very minimal and randomly determined. Regular changes in parent keys are



(a) (b)

Figure 6. Plots of the Sorensen's Similarity Index (SSI) for the first 100 keys generated using Protocol A (6.a) and Protocol B (6.b)

similar to the concept of renewing the security association between the communicating entities.

- *Replay attack*: In both protocols, if an adversary attempts to replay any previously transmitted frames, the server or the tag will immediately recognize, and will take appropriate action as specified by the implementation. However, if an acknowledgement is not received by the tag, and the adversary attempts to replay one of the three recent messages, there is a possibility of the tag not recognizing the replay attack. Nevertheless, the server will always recognize the attack.
- *Man-in-the-middle attack*: This attack will be recognized by the tag and the server in both protocols. Regular updates in the keys and the ASV ensure that any change in the message is immediately identified by the tag and the server. However, if the adversary just acts as a relay for the communication, without making any changes, the proposed scheme will not be able to identify. This relay activity will not be fruitful as discussed in the replay attack scenario.
- *RFID tag tracking*: The protocols protect the tag from tracking. The ID stored is encrypted, and the response by the tag is further encrypted with keys that are updated with each transmitted frame. Continuously changing keys and the absence of key exchange messages ensure that the tag's response is different for each communicated frame. Thus, there is little or no likelihood of tracking being possible if the RFID application employs the proposed scheme.
- *Denial of service (DoS)*: To a certain extent, we can say that DoS attack in the form of multiple queries being sent to the tag is prevented by our protocols. This is because, the tag will detect that the server is not synchronized if it does not receive an acknowledgement frame after three queries. The tag will not respond to further queries, and the server and the application will know that there is a problem. However, it has to be noted that if the tags are flooded with queries, they will be overwhelmed and will not be able to perform their tasks successfully. This will result in denial of service by flooding.
- *De-synchronization attack*: This attack is likely to affect the performance of the protocols presented in this paper. If consecutive acknowledgement frames by the server are not be synchronized in their key states, even though they may be in the same state. In the current implementation of the scheme, the tag will not respond when more than a predefined number (3) of acknowledgement frames are not received. We had configured the protocol to not respond for further queries in case of de-synchronization, in our test and evaluation scenario. However, this aspect of the protocol can be easily modified so that the entities communicate with the last acknowledged key states, until an acknowledgement is received. This requirement is dependent on the application, since some applications may necessitate the use of stringent action in case of dropped frames (i.e. the "assume-guilty-until-proven-innocent" model, where multiple dropped frames may be assumed to be an attack on the system) or otherwise. We assumed this worst case scenario in our test case.
- *Dropped frames*: In the description of the protocol, we discussed briefly of the behavior of the system in case of dropped frames. Frames may be dropped in two parts of the system – (a) in the channel between the tag and the reader, and (b) in the channel between the reader and the server. Furthermore, such dropped frames may occur either due to defects in the communication channel, or due to selective packet dropping attack being performed by an adversary. As described in the discussion on de-synchronization attack, we assume a worst case scenario and configure the tag not to respond in case of multiple dropped frames. However, as discussed, the protocol may be tweaked (as required by the deployment scenario) to work with the previously acknowledged keys.

TABLE III
COMPUTATIONAL OVERHEAD ANALYSIS IN THE TAG

Operation Performed by the Tag	Protocol A (Figure 2(a))	Protocol A (Figure 2(b))	Protocol B (Figure 4(a))	Protocol B (Figure 4(b))
Addition or Increment	2 + α	2 + α	2	2
Pseudorandom Number Generation	1 + α	1 + α	2	2
Logical Operations:				
- NOT - AND - XOR - OR	1 1 + α 1 + α 0	0 0 0 1 + α	1 (or 0) 1 (or 0) 2 0	0 0 2 (or 1) 1 (or 0)
Mathematical Hash (assumed to be one operation)	1	1	1	1
Encryption (assumed to be one operation)	1	1	1	1

TABLE IV
COMPUTATIONAL OVERHEAD SUMMARY

Operation Performed by the Tag	Protocol A (Figure 2(a))	Protocol A (Figure 2(b))	Protocol B (Figure 4(a))	Protocol B (Figure 4(b))
Best case (a = 0)	8	6	8	7
Worst case (a = 1)	12	9	10	9

C. Performance Analysis

Table III presents the computational overhead analysis in the proposed protocols. We restrict our analysis to the computational abilities of the tag, since the server in an RFID system has no known restrictions in performing computations. Please note that we have included both optimized (Figures 2(b) and 4(b)) and un-optimized versions (Figures 2(a) and 4(a)) of our protocols presented in this paper, to illustrate that the computations are at a minimum in either case. We use the variable α to indicate the presence of a parent key generation, which necessitates an alternate key generation in Protocol A. Table IV presents the best and worst case scenarios, denoted by α = 0 and α = 1, respectively. Table V presents the memory overhead for the protocols.

TABLE V
MEMORY OVERHEAD ANALYSIS IN THE TAG

Data on the Tag	Protocol A	Protocol B
Key set	*n* x 3	*n* x 3
Fibonacci number set	*n* x 4	*n* x 4
Parent ID set	8 x 3	8 x 3
Generation set	8 x 3	8 x 3
Encrypted ID	*n* x 1	*n* x 1
ASV	*n* x 1	*n* x 1
patternASV	*n* x 1	*n* x 1
Total	**10 *n* + 48**	**10 *n* + 48**

VI. DISCUSSION

This paper presented a new security protocol for RFID systems (Protocol A) and an enhancement to the said protocol (Protocol B), based on the concept of gene transfer and genetic mutation that enables independent generation of encryption keys at both the tag and the server, in turn ensures mutual authentication.

It may be argued that such protocols work in single tag environments alone. However, we need to note that every acknowledged query updates the keys in the tag and in the server, which in turn means that the *parent* / *generation* are updated.

These updates, coupled with unique initial keys (pre-loaded on each tag, synchronized with the server), mean that this scheme can be applied in multi-tag environments as well. There may be scenarios in such multi-tag environments where the *parent* / *generation* values of several tags will be the same; however, the encrypted data would then enable identifying the exact tag in question.

Presented next are some benefits of the protocols presented in this paper. The protocols offer improved security, since the encryption keys are independently generated without the need for key exchange or update messages. This makes reading attempts by rogue readers futile, since the readers are assumed to be paired with the server initially. Furthermore, the presented protocols update encryption keys continuously, thereby ensuring protection from tracking attacks. Finally, the presented protocols have high generality, which means that they offer flexibility in the size of the keys and in their application. The protocols are flexible because of their modular implementation, which allows us to change any module of the protocol without any changes required in the

other modules. To present an example of the benefit of modular implementation, let us consider the example of encryption. XOR is used in the current implementation, which can be changed with no effect or corresponding changes to any other blocks in the protocols.

Linking keys to the previous keys, as has been done in the presented protocols, gives us the important benefit of mutual authentication. The layered filtering mechanism for authentication, using the *parent* / *generation* numbers, *em* and ASV, also allows the protocols to be scalable to tag / reader intensive environments.

The challenges of this scheme include those imposed by certain assumptions. We assume that the readers are valid, in other words, paired with the server initially. This implies that the onus is on the organization to ensure that the readers are valid. However, with a slight modification, the current scheme can be modified to include readers as an entity. Another important challenge of these (or any protocol focusing on continuous updates with synchronization) is the threat of de-synchronization. This is present, as if multiple key synchronization messages are dropped, the communicating entities lose synchronization. However, with the two previous key states saved, we reduce the chances of, but do not completely avoid, de-synchronization, thereby giving some chance to the entities to re-synchronize.

It could also be argued that the use of Fibonacci numbers as seeds for the pseudorandom number generators limits the security of the proposed protocols, as these numbers become predictable after a certain time. However, our experiments also showed that the seeds need not necessarily be Fibonacci numbers, but, they need to update in the manner in which Fibonacci numbers grow (i.e. new number is the sum of the current and previous numbers). The need for using such numbers as seeds only arose due to the need for randomness and high unpredictability for the series of experiments to verify the concept of Protocol A, where we allowed the keys to update in a particular way until all the bits of the key converged to a sequence of 1 bits. This restriction has been removed in Protocol B. Preliminary tests have shown that Protocol B performs as expected given any seed generation algorithm. This is another benefit of having a modular design for our protocols.

The memory and computational overhead analysis described in tables III, IV and V illustrate that even though the protocol appears complex, it is minimalistic in terms of computations and memory utilization.

VII. Concluding Remarks

In this paper, we presented a novel approach for mutual authentication between the server and the tag, based on the concepts of gene transfer and genetic mutation, and discussed an enhancement to this protocol. The presented protocols use encryption keys initially synchronized between the tag and the server, validated readers, and independent key generation at the tag and server to ensure mutual authentication and security. The protocols ensure that the communicating entities update their keys continuously. The key update mechanism mimics the concept of genetic mutation and gene transfer, and uses acknowledgement based synchronization of key states. The protocols support saving previous key states for recovery and re-synchronization in case of dropped frames.

With the presented protocols, we are able to realize the following security goals – confidentiality, integrity, authentication, non-repudiation and forward security. In their current form, the protocols are applicable to any two entity communication system such as the RFID tag and server. However, the modular design of the protocols allows us to include a third entity, such as the RFID reader, in the mutual authentication protocols thereby making the protocols more secure. The benefits of the protocols outweigh the challenges and allow us to conclude that the presented protocols are secure, simple and flexible, and can be generalized to other application domains.

Acknowledgement

This work has been funded by the Boeing Company. The authors gratefully acknowledge the support and the feedback given by the company. The authors would also like to thank the anonymous reviewers for their valuable comments and suggestions. These have resulted in a substantial improvement in the quality of the paper.

References

[1] R. Want, "An Introduction to RFID Technology," *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 25-33, 2006.

[2] N. Dottling, D. Lazich, J. M-Quade and A. S. de Almeida, "Vulnerabilities of Wireless Key Exchange Based on Channel Reciprocity," *WISA 2012 LNCS 6513*, pp. 206-230, Springer-Verlag Berlin Heidelberg, 2011.

[3] A. Miyaji and M. S. Rahman, "KIMAP: Key-Insulated Mutual Authentication Protocol for RFID," *International Journal of Automated Identification Technology (IJAIT)*, vol. 3, no. 2, pp. 61-74, 2011.

[4] K. Osaka, T. Takagi, K. Yamazaki and O. Takahashi, "An Efficient and Secure RFID Security Method with Ownership Transfer," in *2006 International Conference on Computational Intelligence and Security*, Guangzhou, China, 2006.

[5] Y.-Q. Gui, J. Zhang and H. K. Choi, "An improved RFID security method with ownership transfer," in *2011 International Conference on ICT Convergence (ICTC)*, Seoul, South Korea, 2011.

[6] J. Yu, G. Khan and F. Yuan, "XTEA Encryption Based Novel RFID Security Protocol," in *24th Canadian Conference on Electrical and Computer Engineering (CCECE), 2011*, Niagara Falls, Canada, 2011.

[7] D. Molnar, A. Soppera and D. Wagner, *A Scalable, Delegatable Pseudonym Protocol Enabling Ownership Transfer of RFID Tags*, Cryptology ePrint Archive, 2005.

[8] C.-L. Chen and Y.-Y. Deng, "Conformation of EPC Class 1 Generation 2 standards RFID system with mutual authentication and privacy protection," *Engineering Applications of Artificial Intelligence*, vol. 22, pp. 1284-1291, 2009.

[9] I. Vajda and L. Buttyan, "Lightweight authentication protocols for low-cost RFID tags," in *Second Workshop on Security in Ubiquitous Computing - Ubicomp*, Seattle, USA, 2003.

[10] B. Defend, K. Fu and A. Juels, "Cryptanalysis of Two Lightweight RFID Authentication Schemes," in *Fourth IEEE International Workshop on Pervasive Computing and Communication Security*, New York, USA, 2007.

[11] J. F. V. Vincent, O. A. Bogatyreva, N. R. Bogatyrev, A. Bowyer and A.-K. Pahl, "Biomimetics: Its Practice and Theory," *Journal of the Royal Society Interface*, vol. 3, pp. 471-482, 2006.

[12] R. V. Sampangi and S. Sampalli, "Tag-Server Mutual Authentication Scheme based on Gene Transfer and Genetic Mutation," 2012 IEEE Symposium on Computational Intelligence for Security and Defence Applications (CISDA), pp. 1-8, July 2012.

[13] A. J. F. Griffiths, J. H. Miller, D. T. Suzuki, R. C. Lewontin and W. M. Gelbart, An Introduction to Genetic Analysis, New York: W. H. Freeman, 2000.

[14] Genetic Science Learning Center, "Tour of the Basics," Learn.Genetics, Utah, USA, 1969.

[15] A. Mitrokotsa, M. R. Rieback and A. S. Tanenbaum, "Classifying RFID Attacks and Defenses," *Information Systems Frontiers*, vol. 12, no. 5, pp. 491-505, 2010.

[16] T. Sörensen, "A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons," *Biologiske Skrifter Kongelige Danske Videnskabernes Selskab*, vol. 5, no. 4, pp. 1-34, 1957.

[17] W. Stallings, Cryptography and Network Security: Principles and Practices, Prentice Hall, 2006.

[18] A. Juels, "RFID Security and Privacy: A Research Survey," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 381-394, 2006.

Raghav Sampangi is pursuing his Ph.D. in the area of Security in Radio Frequency Identification (RFID) Systems and Mobile Devices, at the Faculty of Computer Science, Dalhousie University, Canada. His research interests include security and reliability in emerging wireless networks such as, RFID based networks, Wireless Body Area Networks (WBAN), and Vehicular Ad-Hoc Networks (VANET). He is currently involved in research in RFID based systems and Wireless Body Area Networks. Specifically, he is interested in identifying security loopholes, and contributing to the research on security by addressing such issues. Raghav is an active member of the student body at the Faculty of Computer Science, involved in organizing the computer science in-house conference series and representing the graduate student body at the student society. He is a member of the IEEE and the ACM.

Dr. Srinivas ("Srini") Sampalli is a professor and 3M National Teaching Fellow in the Faculty of Computer Science, Dalhousie University, Halifax. His research is in emerging wireless technologies, especially in the intersection of smartphones, near field communications (NFC) and mobile cloud computing. He has investigated protocol vulnerabilities, security best practices, risk mitigation and analysis, design of intrusion detection and prevention systems, and applications in healthcare and mobile commerce. He has recently co-founded a startup company in NFC along with his students. His projects have been sponsored by NSERC, Industry Canada and NRC. Dr. Sampalli has received many teaching awards at the Faculty, University, provincial and national levels, including a named teaching award and 3M National Teaching Fellowship, Canada's most prestigious teaching acknowledgement.