Pandaka: A Lightweight Cipher for RFID Systems

Min Chen Shigang Chen Qingjun Xiao
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32611, USA
Email:{min, sgchen, qxiao}@cise.ufl.edu

***Abstract*—The ubiquitous use of RFID tags raises concern about potential security risks in RFID systems. Because low-cost tags are extremely resource-constrained devices, common security mechanisms adopted in resource-rich equipment such as computers are no longer applicable to them. Hence, one challenging research topic is to design a lightweight cipher that is suitable for low-cost RFID tags. Traditional cryptography generally assumes that the two communicating parties are equipotent entities. In contrast, there is a large capability gap between readers and tags in RFID systems. We observe that the readers, which are much more powerful, should take more responsibility in RFID cryptographic protocols. In this paper, we make a radical shift from traditional cryptography, and design a novel cipher called Pandaka[1], in which most workload is pushed to the readers. As a result, Pandaka is particularly hardware-efficient for tags. We perform extensive simulations to evaluate the effectiveness of Pandaka. In addition, we present security analysis of Pandaka facing different attacks.**

I. Introduction

Radio frequency identification (RFID) technology has been pervasively used in numerous applications, such as inventory management, supply chain, product tracking, transportation, logistics and toll collection [1]–[6]. Typically, an RFID system consists of a large number of RFID tags, one or multiple RFID readers, and a backend server. As specified in EPC Class-1 Gen-2 (C1G2) protocol [7], each tag has a unique ID identifying the object it is attached to. The object may be a vehicle, a product in a warehouse, an e-passport that carries personal information, or a medical device that records a patient's health data. The integrated transceiver of each tag enables it to transmit and receive radio signals. Therefore, it is feasible for a reader to communicate with a tag over a distance as long as the tag is located in its interrogation area.

With the ubiquitous use of RFID technology in not only industries but also our daily life, security problems become a big concern. The security threats on an RFID system include information leakage, denial of service (DoS), tag forgery, unauthorized access to tag memory content, snooping, etc [8]. Security issues in different applications may be different. In this paper, we focus on establishing secure channels between readers and tags to avoid information leakage. Because the wireless channel between a reader and a tag is exposed to surrounding environment, it is easy for a malicious adversary to eavesdrop on their communications.

The biggest challenge for implementing cryptographic protocols in RFID systems is that tags are extremely resource-constrained devices. It is the low price that triggers the explosive growth in use of RFID tags. Generally speaking, a UHF passive tag now costs about 10-50 cents [9]. To keep the price low, a tag must not have many resources, rendering its capabilities in computation, storage and power supply very low. Hence, it is infeasible to directly implement existing sophisticated cryptographic primitives like DES, AES, or RSA on low-cost tags. New cryptographic algorithms specially designed for those low-cost tags are on great demand, which leads to a new branch of cryptography called *lightweight cryptography*.

Recently, some work focused on studying lightweight ciphers, and a number of cryptographic algorithms have been proposed. Since asymmetric (public-key) cryptography usually demands more computation resources than symmetric (secret-key) cryptography, most lightweight ciphers are developed in the scope of the latter. In [10], the authors classify existing lightweight ciphers into three categories: Some researchers tried to optimize and compact standardized block ciphers, like AES [11]–[13] and IDEA [14], thereby reducing their hardware requirements and making them suitable for resource-constrained devices. Those algorithms fall into the first category. The algorithms in the second category are devised by slightly revising classical block ciphers, so they can be applied to lightweight applications. For example, DESL and DESXL [15] are lightweight variants of DES. The third category includes a set of new algorithms that are particularly designed for low-cost devices such as RFID tags and wireless sensors. Among them are lightweight block ciphers, such as PRESENT [16], HIGH [17] and mCrypton [18], and compact stream ciphers, such as Grain, MICKEY and Trivium [19], which are the achievements from the eSTREAM project, while Hummingbird [10] is a hybrid of block cipher and stream cipher.

However, none of the above ciphers goes beyond the traditional paradigm for cryptography design: The two communicating parties are thought to be equipotent entities with comparable capabilities, and they should execute the protocols independently. This is true when communications happen between full-fledged computers. When it comes to RFID systems, however, a reader is much more powerful than a tag. If we stuck with the traditional paradigm when designing a new cipher, the tag would have to operate the same cryptographic functions as those implemented on the reader, which poses a heavy workload on the tag. To avoid this drawback, we should shift away from traditional paradigm. This paper is to provide a new lightweight secret-key cryptography design for systems where significant asymmetry exists between the communicating parties. Particularly, given their capability gap, readers and tags should play different

[1]Pandaka is one of the smallest fish in the world by mass.

978-14799-3360-0/14/$31.00 ©2014 IEEE

roles in a cryptographic protocol. We propose to push complicated tasks to the powerful readers while leaving the tags as simple as possible. Based on this idea, we design a novel lightweight cipher called Pandaka. It does not need a general-purpose processing unit. The tags only need to perform three simple operations: bitwise XOR, one-bit left circular shift, and bit flip, while all other work is done by the readers. We present extensive analysis and simulation results to evaluate Pandaka.

The rest of this paper is organized as follows. Section II gives the security model. Section III describes our new cipher in detail. Section IV evaluates Pandaka by simulations. Section V presents the security analysis. Section VI draws the conclusion.

II. Security Model

A. System Model

An RFID system consists of a large number of tags, one or multiple readers, a backend server, and the communication channels between them. There are three types of RFID tags: *active tag*, *passive tag* and *semi-active tag*. In this paper, we focus on the low-cost passive tags, which are powered by radio energy emitted from the readers. The computation and storage capabilities of each tag are very limited. The backend server, which takes charge of data storage, lookup and high-performance computations, is connected with the readers via high speed wired or wireless links. A reader must be authorized by the backend server before accessing confidential information, such as tag IDs and secret keys. Since the backend server and the readers have abundant resources to implement effective cryptographic primitives, they have no difficulty in establishing secure connections. Therefore, an authorized reader and the backend server can be considered as an integrated entity, still called an *authorized reader*. An *unauthorized reader* has no right to access the backend server, but it can eavesdrop on the wireless channels between the readers and tags. Note that in the following discussion, a reader without further annotation is an authorized one by default.

B. Adversary Model

An adversary will exploit the weaknesses of the RFID system to achieve malicious objectives. In [20], the authors classify adversaries based on their objectives, level of interference, presence, and available resources. In our model, we assume the major purpose of the potential adversary is to intercept confidential information exchanged between the readers and the tags. The adversary is capable of manipulating a few unauthorized readers to eavesdrop on both the forward channel (reader to tag commands) and the backward channel (tag to reader responses). Moreover, the adversary possesses sufficient storage resources to record all messages it overhears for further analysis. Finally, for the sake of not being detected, the adversary will never carry out disruptive attacks that may expose its presence.

III. Cipher design

A. Motivation

A stream cipher generates a pseudorandom bit stream, called *keystream*, and uses it to encrypt the plaintext with a simple XOR operation. Compared with block ciphers, stream ciphers generally have a higher execution speed and a lower hardware complexity, which makes them better candidates for low-cost RFID tags. Since typical stream ciphers belong to symmetric ciphers, it is imperative that every RFID tag must independently produce the keystream if we want to apply such ciphers to RFID systems. However, implementation of common stream ciphers, such as RC4, incurs significant hardware cost. For example, RC4 built by [21] takes 13K GE (logic gate equivalence). Alternatively, we may leverage cryptographic hash functions to generate pseudorandom numbers and transform them into a keystream. A common hash function, such as MD4, MD5, and SHA-1, usually requires more than 7K logic gates [22]. In contrast, a low-cost RFID tag is only integrated with 7K-15K logic gates, of which 2K-5K are used for security purposes [20]. This bottleneck makes it impractical to implement complicated cryptographic functions on low-cost tags. Our goal is to design a new stream cipher that is more hardware-efficient for tags.

One interesting idea is as follows: If we handed the burdensome task of generating a keystream over to a reader and let the reader secretly inform the tags the generated keystream on the fly, the functions implemented on the tags could be much simplified. For example, suppose each RFID tag had an unlimited memory, and was pre-configured with infinite number of different $\langle index, key \rangle$ pairs, which were shared by the reader. When the reader communicates with a tag, it generates random indexes one by one to decide which keys to use. Meanwhile, those indexes are sent to the tag in sequence notifying it of the currently selected keys. Once a key is used, it will be deleted by the reader and the tag to avoid duplicate use. The reader and the tag can generate the same keystream by concatenating the chosen keys in order. More importantly, it is an easy task for the tag to construct the keystream according to the received indexes.

Unfortunately, far from infinity, the memory size of a passive tag is actually very limited, and even a small number of keys can occupy its whole memory. According to the EPC C1G2 standard [7], the memory of a tag logically comprises four distinct banks: *reserved memory*, *EPC memory*, *TID memory*, and *user memory*. User memory can be used for user-specific data storage, such as security keys. A passive UHF tag usually owns a 512-bit user memory. Even some high-end tags only have a memory capacity of up to 32KB [23]–[25], and the prices of such tags are much higher. For example, the x Sky-ID tag [23] with 8KB user memory currently costs $25 each. Due to the cost reason, we assume that the available user memory for storing keys, denoted as M bits, is no larger than 1Kb when we design our cipher.

We call the small number of keys accommodated by each tag *base keys*. Suppose the length of each base key is L bits, and the number of base keys in each tag is N. Clearly, the condition $N \times L \leq M$ must hold. In this case, those base keys

must be reused and updated to produce a massive number of so-called *derived keys*, which are used to form a keystream for encryption. Our basic idea is that the reader makes use of its powerful computation capability to generate derived keys with good randomness, and meanwhile it encodes the information about how each derived key is generated into messages sent to the tag. Instructed by the received messages, the tag can retrieve the same derived keys with little effort. To assist the tag in generating derived keys one by one on the fly, the reader divides its messages into L-bit blocks and encodes the information for producing derived keys in the message blocks.

The length of every derived key is fixed to L bits, the same as that of the base keys. We use $\mathcal{K}$ to represent the key space of all possible derived keys generated from the N base keys, and our objective is two-fold: (1) The cardinality of $\mathcal{K}$ should be 2^L, namely, $\mathcal{K}$ contains all possible values of L bits long; (2) the probability for the appearance of any derived key in $\mathcal{K}$ is approximately equivalent to $\frac{1}{2^L}$.

B. Design Details

We now describe our lightweight cipher Pandaka in detail, where tags only need to perform three simple operations: bitwise XOR, one-bit left circular shift, and bit flip. To simplify the discussion, we just consider the communications between one reader and one tag.

1) Initialization: First of all, some base keys must be shared by the reader and the tag. To generate a number N of L-bit base keys, denoted by k_0, k_1, ..., k_{N-1}, respectively, the reader employs a random number generator that produces random numbers uniformly in the range $\{0, 1, ..., 2^L - 1\}$. Those N base keys are configured into the tag's user memory before deployment. Also, those base keys as well as the corresponding tag ID are stored in the backend database, so the reader can retrieve the base keys with the tag's ID.

2) Derived Keys Generated by the Reader: We design an algorithm for the reader to produce derived keys based on the base keys. To generate a derived key k, each base key k_i ($0 \leq i \leq N-1$) is independently selected with a probability $\frac{1}{2}$. The selection process is achieved by a random bit generator that can uniformly produce 0 and 1. Before determining whether k_i is chosen or not, the reader calls the generator to produce a random bit. The base key k_i will be chosen only if the bit is 1. We denote the set of the chosen base keys as $\mathcal{K}_\mathcal{C}$. Obviously, there are $2^N - 1$ possible results for $\mathcal{K}_\mathcal{C}$, excluding the case that $\mathcal{K}_\mathcal{C}$ is empty when no base key is selected. Afterwards, the reader calculates k by applying XOR operation on all elements in $\mathcal{K}_\mathcal{C}$. For instance, if $\mathcal{K}_\mathcal{C} = \{k_0, k_2, k_{N-1}\}$, then $k = k_0 \oplus k_2 \oplus k_{N-1}$. The complete process for generating a derived key is illustrated in Fig. 1.

3) Base Key Update by the Reader: Once a derived key is generated, the selected base keys should be updated to refresh the key materials. This time, the reader reuses the random bit generator in III-B2 to generate two random bits. Accordingly, four different update patterns are devised. For every chosen base key, it first performs a one-bit left circular shift as shown in Fig. 2, where a one-bit left circular shift is applied on the sequence $(0110\,1101)_2$. Next, a corresponding update pattern is adopted based on the two random bits as follows:

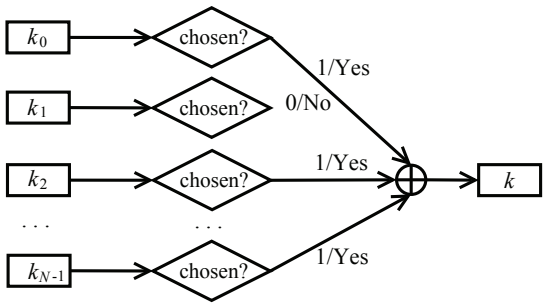

Fig. 1. Generation of derived keys.

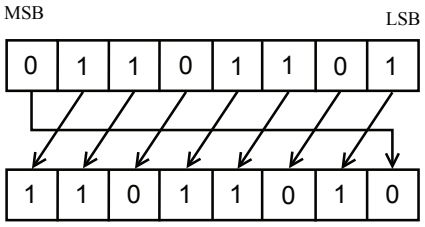

Fig. 2. One-bit left circular shift.

(1) Pattern 0: The two bits are 00. Nothing is to be done.
(2) Pattern 1: The two bits are 01. The reader flips the bits in the base key whose position indexes mod 3 is equal to 0.
(3) Pattern 2: The two bits are 10. The reader flips the bits in the base key whose position indexes mod 3 is equal to 1.
(4) Pattern 3: The two bits are 11. The reader flips the bits in the base key whose position indexes mod 3 is equal to 2.

After that, the update process at the reader side is finished.

4) Design Rationale: The rationale behind our update scheme is to reduce the mutual dependence between bits in each base key. We let $L = 2^\lambda$, where λ is a constant integer, and denote the L bit positions in the base key k_i as $b[i][L-1]$, ..., $b[i][1]$, $b[i][0]$. According to our update scheme for base keys, those L positions can be divided into three *flipping groups*, such that all bits in the same group will be flipped together if the corresponding update pattern is chosen, while bits belonging to different groups will never be flipped together. For example, if $L = 8$, the eight positions in k_i are divided into three flipping groups: $\{b[i][0], b[i][3], b[i][6]\}$, $\{b[i][1], b[i][4], b[i][7]\}$ and $\{b[i][2], b[i][5]\}$.

Consider two arbitrary bits in k_i, denoted by X and Y, which are treated as discrete random variables $\in \{0, 1\}$. Suppose X and Y initially locate at $b[i][p]$ and $b[i][q]$ ($0 \leq p < q \leq L-1$), respectively. Based on their relation when being updated, they are classified into two categories: (1) If $3 \nmid (q-p)$ and $3 \nmid (L+p-q)$, X and Y will never move to two positions that belong to the same flipping group. In other words, there is no chance for them to be flipped together. (2) If $3 \mid (q-p)$ or $3 \mid (L+p-q)$, X and Y may appear at two positions that are in the same flipping group, thereby possibly being flipped together (Note that by no means will both $3 \mid (q-p)$ and $3 \mid (L+p-q)$ hold because $3 \nmid L$). More specifically, within a period of L updates such that X and Y return to their original positions, if $3 \mid (q-p)$, X and Y are in the same flipping group for $(L-(q-p))$ times, and

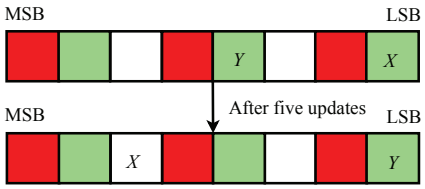

Fig. 3. Illustration of two bits that initially belong to the same flipping group moving to different flipping groups.

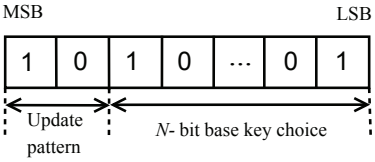

Fig. 4. A $(N+2)$-bit indicator.

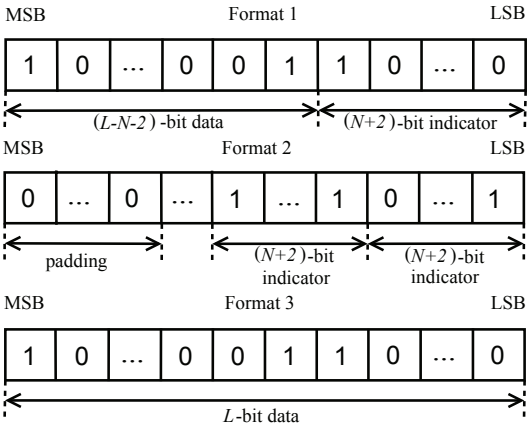

Fig. 5. Different formats of message blocks.

if $3 \mid (L+p-q)$, X and Y may be flipped together for $(q-p)$ times. For example, we let $L = 8$. If X is at $b[i][0]$ and Y is at $b[i][4]$ at the beginning, they belong to the first category. In contrast, if initially X is at $b[i][0]$ and Y is at $b[i][3]$, they belong to the second category. Their initial positions are in the same flipping group, but after five updates, they respectively move to $b[i][5]$ and $b[i][0]$, which are no longer in the same flipping group, just as illustrated in Fig. 3, where positions of the same flipping group are marked with the same color.

In conclusion, there is always a chance for any two bits X and Y in a base key to appear at positions in different flipping groups regardless of their initial positions, and then they can be flipped asynchronously, which reduces their mutual dependence.

5) Indicator for the Tag: Now that it is clear how the reader generates derived keys and updates the base keys, we proceed to show how those two processes can be repeated by the tag in a simpler way. To establish secure communication channels, the reader and the tag should always share the same base keys and generate same derived keys. Hence, each time the reader has to somehow inform the tag its choices of base keys and update pattern, thus instructing the tag to generate the same derived key and update the base keys following the same pattern. To do so, the reader encodes all necessary information into a bit vector, which is called an *indicator*, and sends that indicator to the tag. Let us take a look at the structure of an indicator as illustrated in Fig. 4: It is a $(N+2)$-bit vector, where the low-order N bits represent the reader's choices of base keys. If the i^{th} bit in the indicator is '1', it means the i^{th} base key is selected; otherwise, the i^{th} base key is not chosen by the reader. The remaining two high-order bits in an indicator manifest the update pattern for those chosen base keys.

With the help of an indicator, the tag knows exactly which base keys should be used to calculate the derived key and how to update the selected base keys. Note that there is no need for the tag to implement the same random number generators as the reader does, which simplifies the tag's function.

6) Formats of Message Blocks: The length of a message block is fixed to L bits, and it is bitwise XORed with a derived key for encryption. To avoid the leakage of information in an indicator, it is encrypted as part of a message block. This brings about a problem that no existing derived key can be used to encrypt the first indicator. To address the problem, the reader generates another L-bit random number when initializing the tag, and that random number is stored in both the reader and the tag, serving as the first (pre-stored) derived key. One or more indicators for subsequent derived keys can be piggybacked by current message block, which is encrypted before transmission.

To accomplish the mutual communication between the reader and the tag, we design three different formats of message blocks, which are depicted in Fig. 5. The format 1, denoted by F_1, shows how the reader organizes its message to be sent. A L-bit F_1 message block is composed by two parts: the high-order $(L-N-2)$ bits are data to be transmitted, and the low-order $(N+2)$ bits represent an indicator. The format 2, denoted by F_2, is utilized by the reader to assist the tag in generating derived keys when the tag intends to transmit some data to the reader. Each F_2 message consists of $\lfloor \frac{L}{N+2} \rfloor$ indicators, while the remaining $(L - \lfloor \frac{L}{N+2} \rfloor \times (N+2))$ high-order bits are padded with 0s. The format 3, denoted by F_3, is employed by the tag to send its data to the reader, where all bits are used for encoding the data to be transmitted.

No matter which of the three formats is used, a CRC code is calculated for that block before encryption. A C1G2 RFID tag supports two CRC types: 16-bit CRC and 5-bit CRC [7]. In this paper, we adopt the 16-bit CRC code for the purpose of integrity verification of message blocks. For each CRC code, it is transmitted along with its corresponding ciphertext block. Fig. 6 shows the overall format of a transmitted message block. After receiving a block, the reader or the tag first strips the L-bit encrypted message to decrypt it, and then calculates the CRC code of the decrypted block, which is compared with the attached CRC code. If the verification of the CRC code fails, the reader or the tag sends a request for retransmission of that block.

C. Two-Phase Communications

Communications between the reader and tag are always initialized by the reader. Each session consists of two phases:

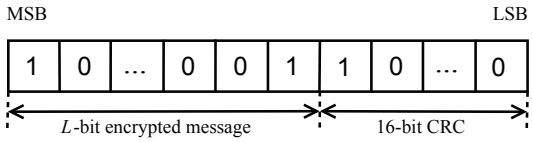

Fig. 6. Overall format of transmitted message blocks.

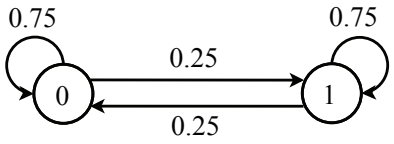

Fig. 7. State transitions of one bit.

In phase one, the reader transmits an encrypted message to the tag, and during phase two, the tag sends its encrypted message with the aid of the reader. More generally, phase one or phase two may be skipped if there is only a one-way message to be transmitted. Before exchanging the messages, the reader first sends a query to the tag, and the tag responds with its ID (Note that privacy of tag ID is not our concern in this paper). Using this received ID, the reader can retrieve the set of base keys and the pre-stored derived key (or the derived key left from last session) from the backend server.

1) Phase One: In phase one, the reader encrypts and sends message blocks in form of F_1 to the tag continuously. The reader segments its message into $(L-N-2)$-bit blocks. For each $(L-N-2)$-bit data block, the reader combines it with a randomly generated $(N+2)$-bit indicator. The derived key for encrypting the first block is the pre-stored random number or left from last session. The message blocks are XORed with the corresponding derived keys for encryption, and then consecutively sent by the reader to the tag in order. Once receiving an encrypted block, the tag is capable of decrypting it using the corresponding derived key. The tag extracts and stores the high-order $(L-N-2)$-bit data from the decrypted block. Meanwhile, the tag calculates the next derived key and updates the chosen base keys guided by the indicator. Note that the indicator can be discarded once it is used. When the transmission is done, the reader informs the tag that phase one ends, so the tag can switch to phase two.

2) Phase Two: Phase two may consist of multiple rounds, where message blocks are exchanged bidirectionally. In each round, the reader first randomly creates $\lfloor \frac{L}{N+2} \rfloor$ indicators, which are concatenated into a F_2 message block. Among those $\lfloor \frac{L}{N+2} \rfloor$ indicators, the first $(\lfloor \frac{L}{N+2} \rfloor - 1)$ ones instruct the tag to generate derived keys that will be used for encrypting its own message, while the last one specifies the derived key that the reader will use to encrypt the next F_2 block. Afterwards, the reader encrypts the F_2 block with a derived key. The first derived key used by the reader in phase two is the last key left from phase one. The tag divides its message to be transmitted into blocks of format F_3. When receiving a message block from the reader, the tag decrypts it, thereby obtaining the $\lfloor \frac{L}{N+2} \rfloor$ indicators. The tag then takes one indicator at a time (from low-order to high-order) to compute a derived key, which is XORed with a F_3 block for encryption. The encrypted message block is sent to the reader. Totally, $(\lfloor \frac{L}{N+2} \rfloor - 1)$ blocks of format F_3 can be transmitted by the tag in one round. The reader knows exactly which derived key is used by the tag to encrypt each block, and therefore can decrypt it correctly to obtain the original message. This process continues round by round until the tag finishes transmitting its message. One may notice that at the end of phase two, at least one indicator is not used, namely, the last indicator in the final F_2 block sent by the reader, and it is shared by the reader and the tag. This guarantees that the reader can initiate another session anytime using the derived key generated with that indicator.

D. Randomness Analysis

1) Randomness: Randomness is a probabilistic property that is the most important metric for assessing a random number generator or a stream cipher. For our protocol, it is of great importance that the generated derived keys are random and unpredictable. Before we lucubrate the randomness of the derived keys, we first study the randomness of a base key k_i during its consecutive updates.

First, we consider one arbitrary bit in k_i, whose value is designated as random variable $X \in \{0, 1\}$. Suppose X is currently located at position $b[i][j]$ $(0 \le j \le L-1)$. When k_i is updated, X is shifted and then flipped with a probability of 0.25 regardless of its new position. Fig. 7 shows the probabilities of state transitions of X. Correspondingly, the transition matrix for X during each update is given by $P_1 = \begin{bmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{bmatrix}$. Using singular value decomposition (SVD), $P_1 = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \times \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}$. Since the choice of an update pattern is independent with each other, after α updates, the transition matrix for X is ${P_1}^{\alpha} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{bmatrix}^{\alpha} \times \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}+\frac{1}{2}\alpha+1 & \frac{1}{2}-\frac{1}{2}\alpha+1 \\ \frac{1}{2}-\frac{1}{2}\alpha+1 & \frac{1}{2}+\frac{1}{2}\alpha+1 \end{bmatrix}$, which converges to $\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$ quickly as α increases. We take $\alpha = 8$ as an example, ${P_1}^{8} = \begin{bmatrix} 0.502 & 0.498 \\ 0.498 & 0.502 \end{bmatrix}$. Therefore, whatever the initial value of X is, it turns to be 0 or 1 with about equal probabilities after several updates. Based on the above analysis, we conclude that the bit at any position in k_i is equally likely to be 0 or 1 in long term.

Now let us investigate two arbitrary bits in an arbitrary base key k_i, denoted by X and Y. Suppose X and Y are initially located at $b[i][p]$ and $b[i][q]$, respectively. Recall from Section III-B4 that the relation between X and Y can be classified into two categories based on their original positions in k_i. For the first category, X and Y are updated independently. Therefore, the values at $b[i][p]$ and $b[i][q]$ are independent. If X and Y belong to the second category, the mutual influence makes the situation more complicated. The left half of Fig. 8 shows state

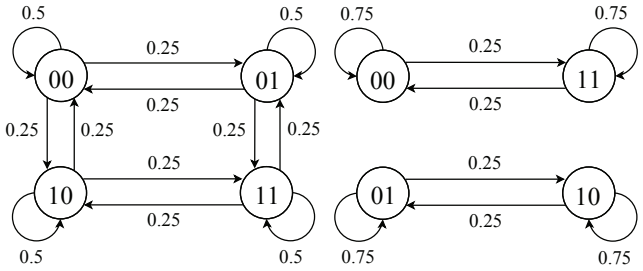

Fig. 8. State transitions of two bits.

transitions of XY when they are flipped asynchronously, while the right half shows state transitions when they are flipped together. The corresponding transition matrixes for left half and right half of Fig. 8 are

$$P_2 = \begin{bmatrix} 0.5 & 0.25 & 0.25 & 0 \\ 0.25 & 0.5 & 0 & 0.25 \\ 0.25 & 0 & 0.5 & 0.25 \\ 0 & 0.25 & 0.25 & 0.5 \end{bmatrix},$$

$$P_3 = \begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0.5 \end{bmatrix}.$$

Assume that within L updates, X and Y move to positions belonging to different flipping groups for β times, and positions in the same flipping group for γ times, subjecting to $1 \leq \beta < L$, $1 \leq \gamma < L$, and $\beta + \gamma = L$. We observe that

$${P_2}^{\beta} \times {P_3}^{\gamma} = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

for arbitrary combination of β and γ. Hence, for any two bits in k_i, regardless of their initial values, they will have the same probability of 0.25 to become 00, 01, 10, and 11 when they are shifted back to their original positions after L updates. In other others, the bits in k_i become pairwise independent.

With the above insight, we go further to study the randomness of the derived keys. Let us consider the j^{th} bit in every base key. Suppose the number of 0s in those N bits is $n_0[j]$, and the number of 1s is $n_1[j]$, where $0 \leq j \leq L-1$, and $n_0[j] + n_1[j] = N$. We have proved that any bit in a base key is 0 or 1 with the same probability. Meanwhile, the values of the j^{th} bits in different base keys are independent. Thus, we have $n_0[j] \sim B(N, 0.5)$, and $P(n_0[j] = t) = \binom{N}{t} \times \left(\frac{1}{2}\right)^N$. We designate the j^{th} bit in a derived key k as $k[j]$. For the value of $k[j]$, we should consider two possible cases: (1) $n_0[j] = N$, namely, there is no 1 in those N bits. In this case, $k[j]$ is definitely 0; (2) If $0 \leq n_0[j] < N$, then $k[j]$ can be 0 or 1. In the second case, if $k[j] = 0$, it implies an even number of base keys whose j^{th} bit is 1 are chosen, and that probability is:

$$\begin{aligned} & P(k[j] = 0 \,|\, 0 \leq n_0[j] < N) \\ =& \frac{2^{n_0[j]} \times \sum_{t=0}^{\lceil \frac{n_1[j]}{2} \rceil} \binom{n_1[j]}{2t} - 1}{2^N - 1} \\ =& \frac{2^{n_0[j]} \times 2^{n_1[j]-1} - 1}{2^N - 1} \\ =& \frac{2^{N-1} - 1}{2^N - 1}, \end{aligned} \tag{1}$$

which is a constant if N is fixed. Thus, we can obtain the probability for $k[j] = 0$ is:

$$\begin{aligned} P(k[j] = 0) =& P(n_0[j] = N) \times P(k[j] = 0 \,|\, n_0[j] = N) \\ & + P(0 \leq n_0[j] < N) \times P(k[j] = 0 \,|\, 0 \leq n_0[j] < N) \\ =& \frac{1}{2^N} \times 1 + (1 - \frac{1}{2^N}) \times \frac{2^{N-1} - 1}{2^N - 1} \\ =& \frac{1}{2}. \end{aligned} \tag{2}$$

Therefore, $P(k[j] = 1) = P(k[j] = 0) = \frac{1}{2}$, which means any bit in the derived key is uniformly distributed over $\{0, 1\}$. Moreover, $k[j]$ is determined only by the j^{th} bits of the base keys, and we have proved bits in every base key are pairwise independent, so bits in k are also pairwise independent.

2) Gap Length: Another metric for evaluating the performance of a random number generator is the gaps occurring between the same digits in the series [26]. This metric can also be extended to assess our algorithm for producing derived keys. *Gap length*, denoted by L_g, is defined as the number of derived keys before the first recurrence of a given derived key. For example, for a derived key whose value is 0, L_g is 3 in the derived-key sequence 0, 1, 2, 3, 0, and L_g is 4 in the derived-key sequence 0, 1, 2, 3, 3, 0. For a L-bit derived key, if it is produced uniformly, the expectation of L_g is

$$\begin{aligned} E(L_g) &= \sum_{j=0}^{\infty} j \times Prob(L_g = j) \\ &= \sum_{j=0}^{\infty} j \times \frac{\left(2^L - 1\right)^j}{(2^L)^{j+1}} \\ &= \frac{1}{2^L} \times \sum_{j=1}^{\infty} j \times \left(1 - \frac{1}{2^L}\right)^j \\ &= \frac{1}{2^L} \times \frac{1 - \frac{1}{2^L}}{\left(1 - \left(1 - \frac{1}{2^L}\right)\right)^2} \\ &= 2^L - 1. \end{aligned} \tag{3}$$

It means that before a derived key reappears it is expected that all (or most) other keys will appear. In Section IV, we will conduct simulations to evaluate the randomness of the derived keys.

E. Hardware Cost for Tag Implementation

Given the constraint that only 2K-5K GEs in low-cost tags can be used for security functions, we estimate the hardware requirement of Pandaka to evaluate its hardware efficiency. The number of GEs required by each cryptographic component

Functional Block	Cost (GEs)
2 input NAND gate	1
2 input XOR gate	2.5
2 input AND gate	2.5
FF (Flip Flop)	12
n-byte RAM	$n \times 12$

TABLE I
COST ESTIMATIONS FOR TYPICAL CRYPTOGRAPHIC HARDWARE.

is not a constant and varies with actual implementation. We use the same parameters for cost estimations as in [20], which are outlined in Table I. A shift register is a group of flip-flops connected in chain, and a circular shift register can be created by connecting the serial input and last output of a shift register [27]. Hence, the circular shift register can be built with $L \times 12$ GEs, where L is the length of a base key. To generate a derived key, we need another two L-bit registers. The first register, which is initialized to all ones, keeps the intermediate result. Each time, the tag reads one chosen base key from memory and stores it in the second register, which is bitwise XORed with the first register. The intermediate result is written back to the first register. Proceeding in this way, the derived key is finally stored in the first register after all chosen base keys are XORed with the first register. The two registers and the XOR operation need $2 \times 12 \times L + L \times 2.5 = 26.5 \times L$ GEs. In addition, we need $N \times L$ bits RAM to store the base keys, which requires $N \times L \times 1.5$ GEs. As a result, the total number of GEs for a tag to implement Pandaka is $L \times 12 + L \times 26.5 + N \times L \times 1.5 = (38.5 + 1.5N) \times L$, which is determined by L and N. For example, if we let $L = 16$ and $N = 6$, a tag needs about $(38.5 + 1.5 \times 6) \times 16 = 760$ GEs to implement Pandaka; if we let $L = 32$ and $N = 6$, a tag needs about $(38.5 + 1.5 \times 6) \times 32 = 1520$ GEs to implement Pandaka. As a comparison, we list the hardware costs of other lightweight ciphers in Table II, where Pandaka(L, N) means Pandaka with N L-bit base keys. It is not surprising that Pandaka requires much fewer GEs than others since it only needs to perform three simple operations.

Cipher	Key bits	Block bits	Cost (GEs)
PRESENT-80 [16]	80	64	1570
PRESENT-128 [16]	128	64	1886
AES [11]	128	128	3400
HIGHT [17]	128	64	3408
mCrypton [18]	96	64	2681
DES [15]	56	64	2309
DESL [15]	56	64	1848
DESXL [15]	184	64	2168
Hummingbird [28]	128	16	2159 [2]
Trivium [29]	80	1	2580
Trivium×8 [29]	80	8	2952
Trivium×16 [29]	80	16	3166
Grain [30]	80	1	1450
Grain×8 [30]	80	8	2756
Grain×16 [30]	80	16	4248
MIKEY [29]	128	1	5039
Pandaka(16, 6)	96	16	760
Pandaka(32, 6)	192	32	1520

TABLE II
COMPARISON OF LIGHTWEIGHT CIPHERS IN TERMS OF HARDWARE COMPLEXITY.

IV. SIMULATIONS

In this section, we use simulations to examine the randomness of derived keys in Pandaka.

A. Frequency Test

First, we examine the randomness of derived keys using frequency test. Specified in the EPC C1G2 Standard [7], one requirement for the generation of 16-bit pseudorandom numbers is that the probability of any 16-bit number $RN16$ with value v being drawn from the generator shall be bounded by $\frac{0.8}{2^{16}} < P(RN16 = v) < \frac{1.25}{2^{16}}$. We extend that requirement to the generation of derived keys with arbitrary length L-bit: The probability of any L-bit derived key k having value v shall be bounded by $\frac{0.8}{2^L} < P(k = v) < \frac{1.25}{2^L}$. To check whether the randomness of the derived keys meets the requirement, we generate N random base keys, and execute the derived-key generator for $2^L \times r$ times to produce a large number of derived keys, where r is a simulation parameter. As a result, approximately $2^L \times r \times (1 - \frac{1}{2^N})$ derived keys are generated. The frequency of each derived key is defined as the number of its appearances divided by the total number of keys derived. From simulation results, we compute the standard deviation σ for the frequencies of derived-key values. Note that the average frequency, denoted by F_{avg}, should be $\frac{1}{2^L}$. Meanwhile, we count the number of derived keys whose frequencies locate in $(\frac{0.8}{2^L}, \frac{1.25}{2^L})$, denoted by N_s, thereby calculating the satisfactory rate, which is defined as $R_s = \frac{N_s}{2^L}$. Constrained by the computation capability of our computer, we are not able to run simulations with $L = 32$ or larger. For this reason, we set the parameters as follows:

(a) $r = 500$, $L = 8$, and N varies from 1 to 4.
(b) $r = 500$, $L = 16$, and N varies from 1 to 4.

As a comparison, we implement Grain (Grain×8 and Grain×16 as shown in Table II), one of the most lightweight existing stream ciphers.

The simulation results are shown in Fig. 9, Table III and Table IV. To make the figures legible, we sample a fraction of 1/50 among all derived keys for displaying. Our results show that the frequencies of derived keys in Pandaka perfectly meet the randomness requirement, even if the number of base keys is very small. Also, we find that the randomness of derived key becomes better when N increases. Moreover, the randomness of keystream in Pandaka is almost as good as that of Grain when $N \geq 4$, while recalling from Table II that the hardware cost of Pandaka(16, 6) is only about one sixth of the cost of Grain×16.

B. Gap Test

Second, we check the randomness of derived keys by gap test. We run simulations to obtain the gap length of any first generated derived key. Under each setting of L and N, the test is conducted 500 times to obtain the average value of

[2]This is the number of GEs required by the most lightweight implementation of Hummingbird.

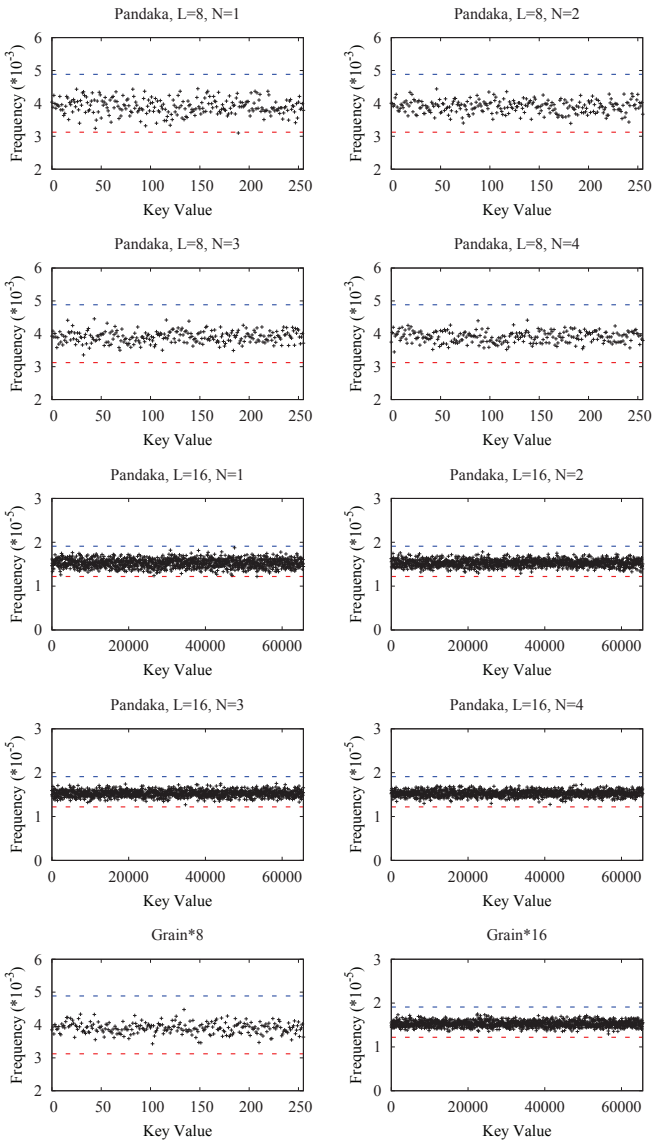

Fig. 9. Frequency test for randomness.

Pandaka					Grain×8
N	1	2	3	4	
F_{avg}	3.91×10^{-3}				
$\sigma(\times 10^{-4})$	2.4	1.9	1.8	1.7	1.7
R_s	99.6%	100%	100%	100%	100%

TABLE III
STATISTICS FOR PANDAKA WITH $L = 8$ AND GRAIN×8.

Pandaka					Grain×16
N	1	2	3	4	
F_{avg}	1.526×10^{-5}				
$\sigma(\times 10^{-7})$	9.7	7.9	7.3	7.1	6.8
R_s	99.95%	100%	100%	100%	100%

TABLE IV
STATISTICS FOR PANDAKA WITH $L = 16$ AND GRAIN×16.

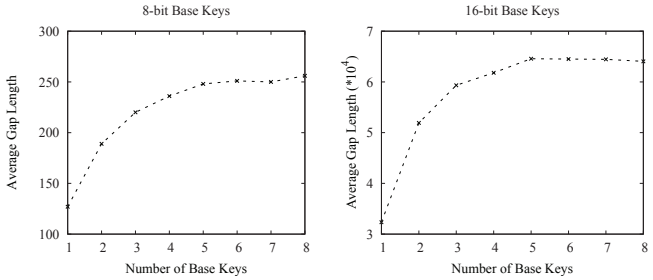

Fig. 10. Gap test for randomness.

gap length, denoted by $\bar{L_g}$. The results are depicted in Fig. 10. According to (3), we know $E(L_g) = 255$ when $L = 8$, and $E(L_g) = 65535$ when $L = 16$. We can see from Fig. 10 that for a given L, $\bar{L_g}$ rises with the increase of N at the beginning. However, the increasing rate of $\bar{L_g}$ gradually slows down, and finally $\bar{L_g}$ slightly oscillates around a stable value that is very close to $E(L_g)$ when $N \geq 5$. That is the reason why we set $N = 6$ for Pandaka in Section III-E. These results show that our derived key generation algorithm can achieve nice randomness with just a few base keys.

V. SECURITY ANALYSIS OF PANDAKA

In this section, we consider several general attacks on Pandaka and analyze to what extent Pandaka can resist against those attacks.

A. Ciphertext Only Attack

Suppose the adversary can only obtain some ciphertext. It has to search through all possible base keys to decrypt the message. For each set of base keys, the adversary tries to decrypt the ciphertext with every possible derived key, and hopes to obtain some recognizable plaintext. However, this is not feasible as the space for base keys is as large as 2^{NL} (e.g. if $L = 16$ and $N = 6$, the space is 2^{96}).

B. Known Plaintext Attack

The only function of an indicator is to help the tag generate a derived key, so neither the reader nor the tag needs to store this temporary data. Hence, we assume the only plaintext an adversary may obtain is the data in a F_1 block or a F_3 block. When an adversary somehow obtains a $\langle plaintext\ block,\ ciphertext\ block \rangle$ pair, it can calculate the corresponding derived key (or part of the derived key if the plaintext is the data part of a F_1 block) by XORing them. The obtained derived keys, appearing completely random (Section IV), will not provide sufficient information to break base keys, which are only used once before being updated; without knowing the indicators, the adversary has no idea on how the base keys are updated.

C. Time-Memory-Data Tradeoff Attack

The authors of [31] propose a new time-memory-data tradeoff attack on stream ciphers. The generic attack on a stream cipher requires about $T = N_s{}^{\frac{2}{3}}$ time, where N_s represents the size of the search space. In our case, $N_s = 2^{NL}$, so $T = 2^{\frac{2NL}{3}}$. If we let $N = 6$ and $L = 32$, $T = 2^{128}$, which makes

the time-memory-data tradeoff attack impracticable. Also, we should keep in mind that the base key materials are always being updated, which makes this attack more difficult to be performed.

D. Tradeoff among Throughput, Security, and Hardware Cost

It is clear that there exists a tradeoff among throughput, security and hardware cost in the design of Pandaka. The security of Pandaka is proportional to the number of key bits NL, and the greater the value of NL is, the more secure Pandaka becomes. However, when N is large, the indictors account for a large proportion of the transmitted messages, resulting in a waste of throughput. Moreover, large NL requires more hardware expenditure for implementing Pandaka. Hence, we should tune those two parameters according to the different constraints and requirements in different application scenarios.

VI. Conclusion

In this paper, we propose a novel lightweight cipher Pandaka tailored to RFID systems. Unlike classical symmetric ciphers in which two communicating parties are burdened with equal workload, Pandaka assigns a heavy workload to the reader as it is more powerful than the tag. The analytical and simulation results demonstrate the effectiveness and hardware efficiency of Pandaka for low-cost tags. Complementary to the traditional cryptographic design approaches, this paper provides a new perspective for developing symmetric cryptography in systems where significant asymmetry exists between the communicating parties.

References

[1] L. Ni, Y. Liu, and Y. C. Lau, "Landmarc: Indoor Location Sensing Using Active RFID," *Proc. of IEEE PerCom*, 2003.

[2] Y. Li and X. Ding, "Protecting RFID Communications in Supply Chains," *Proc. of IEEE ASIACCS*, 2007.

[3] C. H. Lee and C. W. Chung, "Efficient Storage Scheme and Query Processing for Supply Chain Management Using RFID," *Proc. ACM SIGMOD*, 2008.

[4] B. Sheng, C. Tan, Q. Li, and W. Mao, "Finding Popular Categories for RFID Tags," *Proc. of ACM Mobihoc*, 2008.

[5] "AEI Technology," *Softrail. http://www.aeitag.com/aeirfidtec.html*, October 2008.

[6] "Sun Pass," *https://www.sunpass.com/index*.

[7] "EPC Radio-Frequency Identity Protocols Class-1 Gen-2 UHF RFID Protocol for Communications at 860MHz-960MHz, EPCglobal," *http://www.epcglobalinc.org/uhfclg2*, April 2011.

[8] A. Juels, "RFID Security and Privacy: a Research Survey," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, vol. 24, no. 2, pp. 381–394, February 2006.

[9] "atlasRFIDstore," *http://www.atlasrfidstore.com/tags_RFID_chips_s/14.htm*.

[10] D. Engels, X. Fan, G. Gong, H. Hu, and E. M. Smith, "Hummingbird: Ultra-Lightweight Cryptography for Resource-Constrained Devices," *Proc. of International Conference on Financial Cryptography and Data Security*, pp. 3–18, 2010.

[11] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong Authentication for RFID Systems Using the AES Algorithm," *Proc. of CHES*, pp. 357–370, 2004.

[12] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES Implementation on a Grain of Sand," *Proc. of IEEE Information Security*, vol. 15, no. 1, pp. 13–20, October 2005.

[13] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen, "Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core," *proc. of EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools*, 2006.

[14] D Liu, L. Yang, J. Wang, and H. Min, "A Mutual Authentication Protocol for RFID Using IDEA," March 2009.

[15] G. Leander, C. Paar, A. Poschmann, and K. Schramm, *Fast Software Encryption, New Lightweight DES Variants*, Springer Berlin Heidelberg, March 2007.

[16] A. Bogdanov, L. R. Knudsen, G. Le, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," *Proc. of CHES*, pp. 450–466, 2007.

[17] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee, "HIGHT: A New Block Cipher Suitable for Low-Resource Device," *Proc. of CHES*, pp. 46–59, 2006.

[18] C. H. Lim and T. Korkishko, "mCrypton - a Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors," *Proc. of Information Security Applications*, pp. 243–258, 2005.

[19] "eSTREAM: the ECRYPT Stream Cipher Project," *http://http://www.ecrypt.eu.org/stream/*.

[20] D. C. Ranasinghe and P. H. Cole, *Networked RFID Systems and Lightweight Cryptography, Chapter 8 An Evaluation Framework*, Springer Berlin Heidelberg, November 2008.

[21] L. Batina, J. Lano, N. Mentens, S. B. Ors, B. Preneel, and I. Verbauwhede, "Energy, Performance, Area versus Security Trade-offs for Stream Ciphers," *Proc. of Encrypt workshop SASC*, pp. 302–310, October 2004.

[22] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, and Y. Seurin, "Hash Functions and RFID Tags: Mind the Gap," *Proc. of CHES*, pp. 283 – 299, 2008.

[23] "High Memory On Metal UHF RFID Tags," *http://www.oatsystems.com/OAT_Xerafy_RFID_Aerospace_2013/media/High-Memory-Tag-Guide.pdf*.

[24] "DSP VALLEY, newsletter," *http://www.dspvalley.com/userfiles/lr_2845_DSP_NB-april09-E.pdf*.

[25] S. Pais and J. Symonds, "Data Storage on a RFID Tag for a Distributed System," *International Journal of UbiComp*, vol. 2, no. 2, April 2011.

[26] M. G. Kendall and B. Babington Smith, "Randomness and Random Sampling Numbers," *Journal of the Royal Statistical Society*, vol. 101, no. 1, pp. 147–166, 1938.

[27] "Shift Register," *http://en.wikipedia.org/wiki/Shift_register*.

[28] D. Engels, M. O. Saarinen, P. Schweitzer, and E. M. Smith, "The Hummingbird-2 Lightweight Authenticated Encryption Algorithm," *Proc. of RFIDSec*, pp. 19–31, 2012.

[29] T. Good and M. Benaissa, "Hardware Performance of eSTREAM Phase-III Stream Cipher Candidates," *Proc. of CHES*, pp. 46–59, 2006.

[30] M. Hell, T. Johansson, and W. Meier, "Grain; a Stream Cipher for Constrained Environments," *International Journal of Wireless and Mobile Computing*, vol. 2, no. 1, pp. 86–93, May 2007.

[31] A. Biryukov and A. Shamir, "Cryptanalysis Time/Memory/Data Tradeoffs for Stream Ciphers," *Advances in Cryptology Asiacrypt*, pp. 1–13, 2000.