Clone Resistant Mutual Authentication for Low-Cost RFID and Contactless Credit Cards

Stéphane Lemieux
Acadia University,
Department of Mathematics
and Statistics,
Wolfville, NS, Canada, B4P 2R6
stephane.lemieux@acadiau.ca

Adrian Tang
University of Calgary,
Centre for Information
Security and Cryptography,
2500 University Drive NW,
Calgary, AB, Canada

Abstract

With Radio Frequency Identification (RFID) tags being used to secure contactless credit cards, great benefits but also serious security and information privacy issues have arisen. Recently many attempts have been made to resolve these issues. In particular, attempts have been made to provide a means for authentication between tag and reader. However, none have yet have been able to provide resistance to cloning attacks. Furthermore, authentication on lowest range of low-cost RFID tags, also remains a challenge. We propose a clone resistant, mutual authentication scheme that requires only 32 bits of read write memory, 90 bits of read only memory and can be deployed using as few as 300 logic gates. We also propose a stream cipher with the same memory constraints and magnitude of logic gates. These systems may also be scaled to provide a high level of security, using relatively little computational resources, on larger hardware platforms.

Keywords: EPC, RFID, Cloning, Mutual Authentication, Stream Cipher, Electronic Payments

AMS Classification: 94A62, 20L05, 94A60

1 Motivation

Radio Frequency Identification (RFID) is a wireless technology that stores and retrieves data remotely from devices. Systems employing this technology usually take the form of tiny RFID tags which communicate with hand-held or stationary RFID readers. This technology is currently used for inventory and access control, to monitor commercial supply chains, and for electronic payment systems such as E-Zpass and contactless credit cards. Low-cost RFID tags are even expected to replace the barcodes that currently exist on most commercial products.

Despite this widespread and growing deployment, serious security and privacy concerns remain to be addressed. One such concern is the need for an adequately secure mutual authentication protocol, particularly one which could be deployed on the lower cost EPC-type RFID tags. Another is the susceptibility of RFID tags to cloning, even when authentication algorithms are in place.

> "tag memory is insecure and susceptible to physical attacks [29, 31] revealing their entire contents. This includes a myriad of attacks such as shaped charges, laser etching, ion-probes, TEMPEST attacks, clock glitching and many others." [**?**]

Clone resistant mutual authentication is particularly important for contactless credit card systems where

> "[Theoretically] A criminal could simply walk up behind you, position his reader near your wallet, and steal your credit card data without you every knowing it." [**?**]

> "Customer safety depends on protection from impostors who may attempt to impersonate either the customer to the financial service, or the financial service to the customer." [**?**]

Current well trusted private key cryptosystems and mutual authentication algorithms are too large or require too much memory or too much computing power to be effectively deployed on RFID technology, particularly low-cost RFID tags. Furthermore, such cryptosystems were not designed to prevent the cloning attacks that can be applied to RFID technology.

We present a novel mutual authentication protocol and a novel private key cryptosystem that are clone resistant and efficient enough to provide adequate security on the lowest range of EPC-type RFID tags and readers. The efficiency of implementation and clone resistance can both be attributed to two unique features of these algorithms. First, the algorithms use unique algebraic structures to perform necessary calculations. Second, the implementation of the algorithms allows part of the secret key to be stored in two different ways- on the RFID tag as hardware in the form of logic gates, and on the RFID reader in its memory.

In section 2, we introduce the mutual authentication protocol and private key cryptosystem in an easily understandable but insecure form: namely without the use of the unique infinite, non-associative, and usually non-abelian, algebraic structuresalgebraic structures that the authors have termed Abstractions of Integer Arithmetic and Parital Abstractions of Integer Arithmetic. These terms will usually be abreviated as AIAs and PAIAs respectively. In the section 3, we define AIAs and PAIAs and illustrate how to represent each as a short binary string. In Section 4 we define the types of AIAs and PAIAs that we would want to use for cryptographic algorithms. Section 5, describes a mutual authentication algorithm suitable for low-cost RFID technology, or contacless credit cards. Section 6 lists the Tag and Reader specifications necessary to perform the algorithm explained in section 5. Section 7 gives a security analysis only of the mutual authentication. Section 8 introduces a new stream cipher based on a single or multiple AIAs or PAIAs. In Section 9 we discuss the logic gate consumption necessary to implement the algorithms. Finally in Section 10 we give a security analysis of the Stream Cipher.

2 Implementations using regular integer arithmetic

The following algorithms for authentication and encryption fail to achieve any security when calculations are performed via regular integer arithmetic, i.e. the process by which two integers are multiplied together, because the underlying "hard problems" become trivial. Without yet introducing the actual novel hard problems that the secure protocols will be based on, we comment that they become trivial over regular arithmetic because it then has a fast and efficient division algorithm. It is therefore necessary to assume for the moment that given two numbers M and E, one can not divide E by M even if it is known that M is a factor of E. The authentication algorithm then, amounts to multiplying the two integers as pictured below.

$$
\begin{array}{rccccc}
 & & k_n & \cdots & k_2 & k_1 \\
\times_s & & m_p & \cdots & m_2 & m_1 \\
\hline
 & x_{1,p+1} & x_{1,p} & \cdots & x_{1,2} & x_{1,1} \\
x_{2,p+1} & x_{2,p} & \cdots & x_{2,2} & x_{2,1} & \\
\hline
\end{array}
$$

$$
\begin{array}{ccccc}
x_{3,p+1} & x_{3,p} & \cdots & x_{3,2} & x_{3,1} \\
\hline
 & & & \cdot & \\
 & & \cdot & & \\
 & \cdot & & & \\
\end{array}
$$

$$
\begin{array}{ccccccccc}
x_{p,p+1} & x_{p,p} & \cdots & x_{p,2} & x_{p,1} & & & & \\
\hline
e_{p+n} & & \cdots & e_{p+1} & e_p & \cdots & e_2 & e_1 \\
\end{array}
$$

We assume that Alice and Bob have a common secret number $K = k_n \cdots k_2 k_1$ and a common secret digit d. Alice and Bob take turns, in each round, concatenating a randomly chosen digit onto the left hand side of the (changing) number M. M begins with only the digit $m_1 = d$ but grows as each new digit is concatenated. Furthermore all of the digits of M are public knowledge as they are produced, except of course m_1. In each round, Alice generates a new digit m_i, resets $M = m_i m_{i-1} \cdots m_1$ and calculates the product $E = K \times M$. She then transmits to Bob the ordered pair (m_i, e_i) where e_i is the i^{th} digit (from the right) of E. Bob calculates the same product to verify that e_i is in fact the correct digit that corresponds to m_i. If Bob is satisfied he randomly generates m_{i+1} and repeats the process. After r rounds of consecutive successes, both parties are convinced that they share the same secret number K and secret digit d. It is now apparent why regular integer arithmetic could not be used because dividing E by M and guessing d if necessary, would produce K.

The encryption and decryption operate in a similar manner but put an extra restriction on the type of arithmetic chosen. We must now assume that multiplication is not commutative and that we can not divide integers on the left but that given the digit e_i as above, and K and d, Bob can uniquely identify m_i. The algorithm is then similar to authentication except that Alice, instead of generating a random digit m_i uses the next digit in the message she wishes to send to Bob, encrypts it to get e_i and sends it to Bob. Bob then decrypts e_i to recover m_i and Alice begins again by encrypting m_{i+1}. Here again, if division were possible this stream cipher would fall quite easily to a known text attack.

In the next section we introduce classes of infinite non-abelian groupoids and quasigroups that can be used to efficiently implement these algorithms. Much work has been done in [4, 5, 7] on the classification of groupoids and quasigroups. The use of groupoids and quasigroups in cryptography is not even new, see [1, 2] for example. This may however be the first time that infinite groupoids and quasigroups (excluding semigroups and groups of course) have been proposed for cryptographic use.

3 Abstractions of Integer Arithmetic

Almost every school aged child can perform basic arithmetic. However, the specific set of steps by which two integers are multiplied together, can actually be viewed as a complex binary operation on strings of digits involving multiple iterations of two interlocking binary operations $(\oplus, \otimes)$ which act on pairs of digits. For example, we re-consider the product of an n digit integer $k_n \cdots k_2 k_1$ and a p digit integer $m_p \cdots m_2 m_1$ in some unspecified base b. The product is labeled $e_{p+n} \cdots e_2 e_1$.

$$
\begin{array}{cccccccc}
 & & & & k_n & \cdots & k_2 & k_1 \\
 & \times_s & & & m_p & \cdots & m_2 & m_1 \\
\hline
 & & & x_{1,p+1} & x_{1,p} & \cdots & x_{1,2} & x_{1,1} \\
 & & x_{2,p+1} & x_{2,p} & & \cdots & x_{2,2} & x_{2,1} \\
\hline
 & & x_{3,p+1} & x_{3,p} & \cdots & & x_{3,2} & x_{3,1} \\
\hline
 & & & & & & \cdot & \\
 & & & & & \cdot & & \\
 & & & & \cdot & & & \\
 & & x_{p,p+1} & x_{p,p} & \cdots & x_{p,2} & x_{p,1} & \\
\hline
 & & e_{p+n} & & \cdots & e_{p+1} & e_p \quad \cdots & e_2 \quad e_1
\end{array}
$$

In the above product, each number $x_{i,p+1}x_{i,p}\cdots x_{i,2}x_{i,1}$ is the intermediate product of the number $k_n\cdots k_2k_1$ and the digit m_i. If we consider the product of two single digit integers, say 3 and 7 in a fixed base, say 10, then the product can be viewed as a binary operation $\otimes$ that maps the set of ordered pairs of digits to the set of ordered pairs of digits between 0 and 9. In this case $\otimes : (3,7) \to (2,1)$ or $3 \otimes 7 = 21$. An addition operation $\oplus$, can be similarly defined on ordered pairs of digits so that, for example $\oplus : (3,7) \to (1,0)$ or $3 \oplus 7 = 10$. If we label the coordinates of the output as the carry and remainder of the operation, then we can write $\otimes : (3,7) \to ((3 \otimes 7)_c, (3 \otimes 7)_r)$. We can then use the regular steps commonly accepted for multiplying two integers 'by hand' to write each digit in the product of number $k_n\cdots k_2k_1$ and the digit m_i as a composition of these two operations. For example,

$$x_{i,1} = (k_1 \otimes m_i)_r,$$

$$x_{i,2} = ((k_2 \otimes m_i)_r \oplus (k_1 \otimes m_i)_c)_r,$$

$$x_{i,3} = ((m_3 \otimes k_1)_r \oplus ((m_2 \otimes k_1)_c \oplus ((m_2 \otimes k_1)_r \oplus (m_1 \otimes k_1)_c)_c)_r)_r.$$

We can then sum vertical columns of digits to derive a formula for each e_i but the nesting of remainders and carries will soon be unwielding.

Fortunately most of us already know the algorithm (arithmetic) by heart and so there is no reason to display the results in such long hand. Doing so however, does elucidate a number of interesting properties of integer multiplication:

1. Both digit-wise addition, $\oplus$, and digit-wise multiplication, $\otimes$, are binary operations that map each pair of digits (with respect to a given base b) to another pair of digits, namely the remainder and carry.

2. The algorithm for multiplication of integers (strings of digits) works independent of the choices of output for the operations $\oplus$ and $\otimes$. That is, for each of $\oplus$ and $\otimes$, if we change the output (carries and remainders) associated with one or more ordered pairs of digits, then the integer multiplication algorithm will still work but will produce different output strings.

3. Changing the outputs of $\oplus$ and $\otimes$ can, and in general will, alter the algebraic properties of the resulting string-wise multiplication.

Given that the algorithm for basic arithmetic is common knowledge, in order to define a new string-wise multiplication it is sufficient to simply list, in table format or as an ordered string, the remainders and carries associated with each ordered pair of digits for the $\oplus$ and $\otimes$ operations. Figure 3 and the subsequent derived string, 000102010210021011000000000102 000211, give the remainders and carries for actual addition and multiplication in base 3.

Going in the opposite direction, if we randomly generate a string, s, of $4b^2$ digits (base b) we can define tables for $\oplus_s$ and $\otimes_s$ and thus generate a new string-wise multiplication.

Addition Base 3, $\oplus$			
a	b	carry	remainder
0	0	0	0
0	1	0	1
0	2	0	2
1	0	0	1
1	1	0	2
1	2	1	0
2	0	0	2
2	1	1	0
2	2	1	1

Multiplication Base 3, $\otimes$			
a	b	carry	remainder
0	0	0	0
0	1	0	0
0	2	0	0
1	0	0	0
1	1	0	1
1	2	0	2
2	0	0	0
2	1	0	2
2	2	1	1

Derived String 000102010210021011000000000102000211

Figure 1: Base 3 Arithmetic

Definition 1 (Abstraction of Integer Arithmetic). *Let B be the set of all base-b strings of finite length. Then any base-b string, s, of length $4b^2$ defines a binary operation, $\times_s$ on B using the algorithm for regular integer multiplication but with the remainders and carries of digit-wise multiplication and addition taken from s as detailed above. We call the pair $(b, \times_s)$ an abstraction of integer arithmetic, or AIA for short. By AIAs we shall mean abstractions of integer arithmetic.*

Lemma 1. *For any positive integer base-b, there exists a base-b string, s_b, of length $4b^2$ such that $(B, \times_s)$ is regular integer multiplication restricted to non-negative integers.*

Proof. The string s for base 3 is given in Figure 3. Exhibiting the generic formula for arbitrary positive integer base b is left to the reader. □

We shall label regular integer multiplication restricted to the non-negative integers AIA_0. We now define a partial abstraction of integer arithmetic or PAIA.

Definition 2 (Partial Abstraction of Integer Arithmetic). *Let B be the set of all base-b strings of finite length. Then any base-b string, s, of length $2b^2$ defines a binary operation, $\oplus_s$ on B using the algorithm for regular integer addition but with the remainders and carries of digit-wise addition taken from s as detailed above. We call the pair $(b, +_s)$ a partial abstraction of integer arithmetic, or PAIA for short. By PAIAs we shall mean partial abstractions of integer arithmetic.*

4 Choosing AIAs

For our algorithms, we will only use AIAs and PAIAs that are regular according to the definitions below.

Definition 3 (Regular AIA). *An AIA will be called regular if the following hold.*

1. *The multiplicative carries, $(i \otimes j)_c$, are any non-zero digits $\{1, 2, ..., b\}$.*
2. *Each of the additive carries, $(i \oplus j)_c$ are either 0 or 1.*
3. *The additive carries of the form $(1 \oplus i)_c$ are all 0.*
4. *For each fixed $i \in \{1, 2, ..., b\}$, the multiplicative remainders $(i \otimes 1)_r$, $(i \otimes 2)_r$, ...,$(i \otimes b)_r$ are all distinct.*

5. *For each fixed* $i \in \{1, 2, ..., b\}$*, the additive remainders* $(i \oplus 1)_r$*,* $(i \oplus 2)_r$*, ...,*$(i \oplus b)_r$ *are all distinct.*

Remarks:

- In general, neither of $\otimes$ and $\oplus$ will be commutative.
- The digit 0 is a restricted, place-holding, digit that can not be accessed by either party using the authentication algorithm. In fact, when we say base-b from now on, we refer to the digits $\{1, 2, ..., b\}$.
- Properties 1, 2, and 4 will ensure that $\forall x, y, z, w \in \{1, 2, ..., b\}$, $((x \oplus y)_c + (z \otimes w)_c)_c = 0$ and $((x \oplus y)_c + (z \oplus w)_c)_c = 0$. In other words, the sum of a multiplicative carry and an additive carry or the sum of two additive carries will not produce another non-zero carry.
- Property 5 ensures that $\otimes$ is left-cancellative. Thus, if we know the two digits i and $(i \otimes j)_r$ then we can deduce j. Define σ_i to be the permutation of $\{1, 2, ..., b\}$ such that $(i \otimes j)_r = \sigma_i(j)$ and hence $\sigma_i^{-1}((i \otimes j)_r) = j$. Similarly, property 6 ensures that $\oplus$ is left-cancellative. We define γ_i to be the permutation of $\{1, 2, ..., b\}$ such that $\gamma_i^{-1}((i \oplus j)_r) = j$.
- Properties 5 and 6 together, ensure that the AIA is left-cancellative, and has a left-division algorithm, thus allowing for unique decryption.

For ease we can express A_1 as the concatenation of four strings $C_\times C_+ R_\times R_+$ where:

a) C_+, the carries under addition, is the concatenation of b zeros and a randomly chosen binary string of length $b^2 - b$.

b) R_+, the remainders under addition, is the concatenation of b randomly chosen permutations of $\{1, 2, ..., b\}$.

c) $C_\times$, the carries under multiplication, is a randomly chosen base-b string of length b^2.

d) $R_\times$, the remainders under multiplication, is the concatenation of b randomly chosen permutations of $\{1, 2, ..., b\}$.

Example 1. *A sample regular AIA for* $b = 4$*, might be*

$$A = 10110111010032412413314241232314443121132124234114234312 4123.$$

Of course, A can be more easily read as in Figure 4.

Definition 4 (Regular PAIA). *A PAIA will be called regular if the following hold.*

1. *Each of the additive carries,* $(i \oplus j)_c$ *are either* 0 *or* 1*.*
2. *For each fixed* $i \in \{1, 2, ..., b\}$*, the additive remainders* $(i \oplus 1)_r$*,* $(i \oplus 2)_r$*, ...,*$(i \oplus b)_r$ *are all distinct.*
3. *The additive carries of the form* $(1 \oplus i)_c$ *are all* 0*.*

Remarks:

- In general, $\oplus$ will not be commutative.

Add Base 4 $\oplus$			
K	M	C	R
1	1	0	3
1	2	0	2
1	3	0	4
1	4	0	1
2	1	1	2
2	2	0	4
2	3	1	1
2	4	1	3
3	1	0	3
3	2	1	1
3	3	1	4
3	4	1	2
4	1	0	4
4	2	1	1
4	3	0	2
4	4	0	3

Mult. Base 4 $\otimes$			
K	M	C	R
1	1	2	2
1	2	3	3
1	3	1	4
1	4	4	1
2	1	4	1
2	2	4	4
2	3	3	2
2	4	1	3
3	1	2	4
3	2	1	3
3	3	1	1
3	4	3	2
4	1	2	4
4	2	1	1
4	3	2	2
4	4	4	3

Figure 2: Example of a Regular AIA

- As with regular AIAs, the digit 0 is a restricted, place-holding, digit that can not be accessed by either party using the authentication algorithm.
- As before properties 2 and 3 ensure that carries do not propagate
- As before, property 2 ensures that $\oplus$ is left-cancellative. We define γ_i to be the permutation of $\{1, 2, ..., b\}$ such that $\gamma_i^{-1}((i \oplus j)_r) = j$. This ensures that the PAIA is has a left-subtraction algorithm, thus allowing for unique decryption.

5 Mutual Authentication

There are several possible ways to implement a mutual authentication scheme with a shared secret key comprised of one or more base b strings and one or more AIAs and or PAIAs. We will begin with what we believe to be the most efficient implementation. It runs over a fixed number of PAIAs but could be implemented over a single PAIA. Furthermore, after viewing the algorithm it should be obvious how it could be implemented using a single or a fixed number of AIAs. The following assumptions are made:

1. The authentication will occur between an unconstrained platform, for example an RFID reader, and a constrained platform, for example an RFID tag or label.
2. The reader will store many secret keys, each corresponding to a different RFID tag. Further the reader will store the secret keys as strings in memory and use them as imput to initialize a software implementation of the mutual authentication algorithm in much the same way that most computer algorithms are traditionally implemented.
3. The tag however, will have a single secret key and will only store K and d (defined below) in memory. The rest of the secret key, PA, will be implemented as hardware, in the form of logic gates on the tag.

4. As given in the sample specifications above (Figure 1) the tag has a random number generator and can perform simple calculations provided the maximum allowable gate count to perform these calculations is not exceeded.

We begin by choosing a base b. Next, we randomly generate a secret key (K, PA, d) where:

1. K is the concatenation of b strings $K_1, K_2, ..., K_b$ each $n+1$ digits long, i.e. $(n+1)log_2(b)$ bits, such that$(K_i)_1 \neq (K_j)_1$ for $i \neq j$ where $(K_i)_1$ is the rightmost digit of K_i. In general the t^{th} rightmost digit of K_i will be labeled $(K_i)_t$.
2. PA is the concatenation of n strings $PA_1, PA_2, ..., PA_n$ each $2b^2$ digits long. Further more each PA_i will correspond to a regular partial abstraction of integer arithmetic.
3. d will be a single randomly chosen base b digit.

In the authentication algorithm, two parties Alice and Bob, both having the same secret key, (K, PA, d) described above, will simultaneously participate in an algorithm that is similar to multiplying two base b integers. Referring to the example multiplication,

$$
\begin{array}{cccccc}
 & & k_n & \cdots & k_2 & k_1 \\
\times_s & & m_p & \cdots & m_2 & m_1 \\
\hline
 & x_{1,n+1} & x_{1,n} & \cdots & x_{1,2} & x_{1,1} \\
x_{2,n+1} & x_{2,n} & \cdots & x_{2,2} & x_{2,1} & \\
\hline
\end{array}
$$

$$
\begin{array}{ccccccccc}
x_{3,n+1} & x_{3,n} & \cdots & x_{3,2} & x_{3,1} & & & & \\
\hline
 & & & & \cdot & & & & \\
 & & & \cdot & & & & & \\
 & & \cdot & & & & & & \\
x_{p,n+1} & x_{p,p} & \cdots & x_{p,2} & x_{p,1} & & & & \\
\hline
e_{p+n+1} & & \cdots & e_{n+1} & e_n & \cdots & e_2 & e_1 &
\end{array}
$$

as mentioned earlier, each number $x_{i,n+1}x_{i,n}\cdots x_{i,2}x_{i,1}$ is really the product $k_n \cdots k_2k_1 \times_s m_i$. We will define $K_j = k_n \cdots k_2k_1 \times_s m_i$ if $m_i = j$. In so doing, we do not need to use a number $k_n \cdots k_2k_1$ in order to perform the multiplication. Instead, the number $m_p \cdots m_2m_1$ will dictate the order in which we should add the rows $x_{i,n+1}x_{i,n}\cdots x_{i,2}x_{i,1} = k_n \cdots k_2k_1 \times_s m_i = K_j$.

Both Alice and Bob begin by making $m_1 = d$ so that $K_d = x_{1,n+1}x_{1,n}x_{1,2}x_{1,1}$. Notice in the multiplication above, the digit $x_{1,1}$ is not added to any other digit and we don't want to transmit this digit so we can discard it and only store the leftmost n digits of K_d which we refer to as $(K_d)'$. We will label the register (volatile memory) as X.

Notation:

- For any base-b string X, we denote to the i^{th} right-most digit by $(X)_i$.
- If X has length n, by X', we denote the left-most $n-1$ digits. Thus $X = X'(X)_1$.

Thus for both parties we have:

Step 1: $X \leftarrow (K_d)'$

Now one of the parties, Alice say, generates a random base b digit m_2. In so doing she effectively chooses $x_{2,n+1}x_{2,n}\cdots x_{2,2}x_{2,1}$ to be K_{m_2}. She must now add $X = (K_d)'$ to K_{m_2}

using PA to indicate how she will perform the addition. Recall that PA is the concatenation of n partial abstractions of integer arithmetic $PA_1, PA_2, ..., PA_n$. Digits $x_{1,2}$ and $x_{2,1}$ will be added by looking up the ordered pair of digits $(x_{1,2}, x_{2,1})$ in the table defined by PA_1 to produce the remainder and carry $((x_{1,2} \oplus_{PA_1} x_{2,1})_c, (x_{1,2} \oplus_{PA_1} x_{2,1})_r)$. Notice that in the multiplication above, $e_2 = (x_{1,2} \oplus_{PA_1} x_{2,1})_r$. Alice can then transmit the pair of digits (m_2, e_2) to Bob and continue to add the two strings $(K_d)'$ and K_{m_2}.

The next step in this addition, depends on whether $(x_{1,2} \oplus_{PA_1} x_{2,1})_c$ is a 0 or 1. If $(x_{1,2} \oplus_{PA_1} x_{2,1})_c = 0$ there is no intermediate step and Alice can add the digits $x_{1,3}$ and $x_{2,2}$ via PA_2. If however, $(x_{1,2} \oplus_{PA_1} x_{2,1})_c = 1$, the intermediate step $x'_{1,3} = (1 \oplus_{PA_2} x_{1,3})_r$ needs to be calculated (essentially by looking up the ordered pair in the table generated by PA_2. Notice that $(1 \oplus_{PA_2} x_{1,3})_c$ will be 0. In this instance, Alice can now add $x'_{1,3}$ and $x_{2,2}$ via PA_2, and continue to add the rest of the digits in the strings $(K_d)'$ and K_{m_2}. At each stage, the $(i-1)^{th}$ additive carry is added to $(i+1)^{th}$ digit of $(K_d)'$ and then this sum is added to the i^{th} digit of K_{m_2} with PA_i indicating how to perform the addition. As a final note for this step, the digit e_1 does not need to be stored for the next round so only the n rightmost digits of the sum are stored in X to update it for the next round. In summary this step is:

Step 2a:	Generate $m_2 \in \{1, 2, ..., b\}$
Step 2b:	Calculate $X +_{PA} K_{m_2}$
Step 2c.	Transmit (m_2, e_2).
Step 2d.	Update $X \leftarrow (X +_{PA} K_{m_2})'$

Bob can now receive (m_2, e_2), perform the same addition as Alice and check if $(X +_{PA} K_{m_2})_1 = e_2$. If it does not, Alice fails to authenticate. If it does Bob initiates the next round by randomly choosing m_3, calculating $(X +_{PA} K_{m_3})_1 = e_3$, transmitting (m_3, e_3) to Alice and updating his register to $X \leftarrow (X +_{PA} K_{m_3})'$. The process can be repeated for a preset number of rounds at which time both parties will be convinced that they share the same secret key. Each time the i^{th} round looks like:

Step 3a:	Recieve (m_{i-1}, e_{i-1})	
Step 3b:	IF $\{(X +_{PA} K_{m_{i-1}})_1 \neq e_{i-1}\}$ then QUIT.	ELSE:
Step 3c:	Update $X \leftarrow (X +_{PA} K_{m_{i-1}})'$	
Step 3d:	Generate $m_i \in \{1, 2, ..., b\}$	
Step 3e:	Calculate $X +_{PA} K_{m_i}$	
Step 3f:	Transmit (m_i, e_i)	
Step 3g:	Update $X \leftarrow (X +_{PA} K_{m_i})'$	

5.1 Authentication over a single AIA

We comment that the above implementation can be run, with less security but requiring less memory from the reader, over a single PAIA, simply by setting all PA_i's equal. The algorithm can also be run over a single regular AIA, say A_1, provided the strings K_i are precomputed as $K_i = K \times_{A_1} i$ where K is the single randomly generated string. If $\otimes$ and $\oplus$ are the digitwise addition and multiplication associated with A_1 then this precomputation is as follows.

Input:	$K = k_n k_{n-1} \cdots k_2 k_1, i \in \{1, 2, ..., b\}$
Output:	$K_i = K \times_{A_1} i$
Step 1:	$t_1 \leftarrow (k_1 \otimes i)_r, carry \leftarrow (k_1 \otimes i)_c$
Step 2:	For $j = 2$ to n
Step 2a:	$t_j \leftarrow ((k_j \otimes i)_r \oplus carry)_r$
Step 2b:	$carry \leftarrow ((k_j \otimes i)_c \oplus ((k_j \otimes i)_r \oplus carry)_c)_r$
	End For
Step 3:	$t_{n+1} \leftarrow carry$
Step 4:	Output $K_i = t_{n+1} t_n \cdots t_2 t_1$, stop.

Figure 3: Precomputation

(Note: If $n = \text{length}(K_1)$, then $K_1 \times_{A_1} i$ can have length $n + 1$.)

The authentication algorithm can now be run using only the additive table associated with A_1 in place of PA.

6 Necessary Specifications for Implementation

Sample EPC Specifications	
Storage:	$128 - 512$ bits of read-only storage.
Memory:	$32 - 128$ bits of volatile read-write memory.
Gate Count:	$1000 - 10000$ gates.
Security Gate Count Budget:	$200 - 2000$ gates.
Operating Frequency:	$868 - 956$ MHz (UHF).
Scanning Range:	3 meters.
Performance:	100 read operations per second.
Power Consumption:	10 microwatts.
Features:	Anti-collision Protocol Support. Random Number Generator.

Figure 4: Sample EPC Specifications from [3]

The most severe of the restrictions, that the specifications in Figure 1 pose for potential cryptographic solutions, are the small number of logic gates $200-2000$ which can be devoted to security algorithms, and the volatile memory available to store intermediate calculations. The thriftiest implementation of the standard private key cryptosystem, AES (Advanced Encryption Standard), currently requires approximately 4000 logic gates [6]. The one-way private key authentication scheme proposed in [3], appears to meet the above requirements for EPC type RFID tags but may fail to authenticate legitimate tags and readers with some

non-trivial probability, and provides only 24 bits of security for the lowest range of allowable re-writable memory. The authentication scheme in [6] provides mutual authentication, and claims to be within the necessary range of logic gate usage (approximately 300), but requires 480 bits of volatile memory, considerably more than the above specification allows.

The following specifications are necessary, in terms of base b, and $n+1$ = number of digits in each K_i. For the tag, we require:

Necessary RFID Specifications	
Storage:	$b(n+1)\log_2(b)+2$ bits for K and d.
Memory:	$(n+2)\log_2(b)$ bits of volatile read-write memory.
Security Gate Count:	$30n$ to $40n$ gates, when $b=4$.
Features:	Anti-collision Protocol Support. Random Number Generator.

Figure 5: Necessary Specs for Authentication.

The reader is required to store PA as a binary string, each PA_i consisting of b^2-b additive carry bits (no carries for $1 \oplus i$), and b of the $b!$ possible permutations of $\{1,2,\ldots,b\}$. This requires $n((b^2-b)+b\log_2(b!))$ bits plus $b(n+1)\log_2(b)+2$ bits for K and d.

For example if we choose $b=4$, $n=10$, the Tag will require 90 bits of Read only memory, or periodically re-writable memory (see Section **??**), 24 bits of Volatile memory, and 300 to 400 logic gates, as explained in Section 7. The reader will require 410 bits of memory to store (K, PA, d).

7 Security Analysis of Authentication Based on PAIAs

We give the security against brute force attacks under what be believe to be the relevant security models. We will assume $b=4$ and $n=10$.

First note that at each round the probability of one party correctly guessing the correct digit e_i to transmit with m_i is $1/4$. Therefore, guessing correctly for 40 rounds would occur with probability $(1/4)^{40}$, i.e. requiring approximately 2^{80} guesses. We hope to provide at least 80 bits of security, against the various attack models so we will assume that the Tag and Reader each perform 40 rounds of the authentication algorithm. Of course this number is easily increased to the maximum that the following models allow.

7.1 Model 1

Model 1 will be the strongest implementation possible where each Tag is designed with a unique set of logic gates to perform the authentication. In this instance the attacker does not know ahead of time any portion of (K, PA, d). Brute force would then require uncovering all four of K_1, K_2, K_3, K_4, as well as d and the table values for each of PA_1, PA_2,..., PA_{10}. Given this amounts to 4^{44} guesses for K_1, K_2, K_3, K_4, 4 guesses for d and $(2^{12}\times(4!)^4)^{10}$ guesses for PA_1, PA_2,..., PA_{10} for a total of $2^{255.85}$.

It bears mentioning that building a unique set of logic gates for each RFID Tag may be cost prohibitive in some circumstances, but may be justified for applications such as financial smart cards. In so doing the following security properties appear to be satisfied.

1. **Mutual Authentication.** As both the reader and tag are asking and answering distinct challenges, the identity of both is simultaneously being tested.

2. **Man in the middle attack prevention.** The fact that mutual authentication is demanded for each round of the protocol prevents a man in the middle attack.

3. **Resistance to Cloning Attacks.** Even if the secret string K is lifted from the tag, an attacker wishing to clone the tag would need to read the logic gate configuration on the tag, and produce new tags with this same logic gate configuration in order to imitate the original tag.

4. **Forward Security (optional.)** As the secret string K is stored in memory, periodically, once authentication is successful the tag's secret string could be updated.

5. **Replay attack prevention.** Storing all messages from the tag or reader and and replaying them to the appropriate device (in order to fool that device) will not work because because both parties participate in generating the string M.

7.2 Model 2

For Model 2 we assume that building a unique set of logic gates for each RFID Tag is cost prohibitive. Instead, we propose mass producing one set of logic gates per product line or per company, and maintaining this set of logic gates and the binary string PA associated with it, as a trade secret. We make the following assumption from [**?**],

> Tag memory is insecure and susceptible to physical attacks revealing their entire contents. This includes a myriad of attacks such as shaped charges, laser etching, ion-probes, TEMPEST attacks, clock glitching and many others. Fortunately, these attacks require physical tag access and are not easily carried out in public or on a wide scale without detection. Privacy concerns are rather moot if someone can remove a tag or steal the item it is attached to without detection. The key point is that tags cannot be trusted to store long-term secrets, such as shared keys, when left in isolation.

In this attack model, we therefore assume that the memory of Tag A has been compromised and that the attacker knows K and d but not PA. In this instance, the attacker would first observe numerous successful and unsuccessful authentication attempts and then uncover K and d and use the combined information to determine PA. If the attacker is successful in recovering PA, he or she will then be able to use this information to attack the remaining tags that use the same PA as Tag A. The probability of success in this second stage attack will be examined in Model 3. The total number of possible PA's are then be $2^{165.85}$ however, there is a definite priority with which the attacker would like to uncover the string (or logic gate configuration associated with the string) PA. The right-most digit adder is used most often, and can be uncovered by only knowing the four possible pairs $(1, e_{1_1})$, $(2, e_{1_2})$,$(3, e_{1_3})$, and $(4, e_{1_4})$. In fact, if K, and d are known, only the 2^{22} pairs of the form (j, e_{i_j}) for $i \in \{1, ..., 11\}$, $j \in \{1, 2, 3, 4\}$ are needed to recover PA. To compensate for this we could either increase the size of PA by concatenating 40 PAIA's, and thus requiring 1200 to 1600 logic gates or by simply requiring that K and d be reset after each successful authentication, or after a preset number of authentications. We therefore have the same security characteristics as the previous model, except that forward security is essential instead of being optional. This would off course increase the necessary rewritable memory constraints to cover the 24 bits needed for calculations and 90 bits need to store K and d.

7.3 Model 3

Model 3 can be viewed as a continuation of Model 2, where the logic gate configuration for Tag A has been determined and the secret key for Tag B, with the same logic gate configuration, is now desired. Thus we can assume PA is known, and try to determine K, and d. By brute force there are 2^{90} such keys. This could also be a valid attack if the logic gate configuration was mass produced for every product thus rendering PA essentially

public knowledge. However, in this instance there are many ways to further optimize the implementation. For example, using the same PA_i for each digit would shorten the key stored on the reader. Periodically reseting K and d would be essential in this case.

8 Stream Cipher

We now demonstrate a stream cipher on a single regular AIA, labeled A_1. Likewise, several variations can be implemented on AIAs or PAIAs. It is currently common practice in securing RFID technology, to store only already encrypted data on the RFID tag, thus minimizing the computational work that the tag must do. However, we shall demonstrate the computational cost of encrypting data using the stream cipher, we describe below, is so small that the tag can efficiently perform this encryption on already encrypted data, effectively allowing the tag to sign each of the messages it sends.

We will therefore need to call upon the algorithm in Figure 3, which precomputes K_i, for $i \in \{1, ..., b\}$. Encryption works as follows.

8.1 Encryption

Notation:

- For any base-b string X, we denote to the i^{th} right-most digit by $(X)_i$.
- If X has length n, by X', we denote the left-most $n-1$ digits. Thus $X = X'(X)_1$.

The encryption algorithm works as follows:

Input:	K, d, M. (M can be read in digit by digit)
Output:	Ciphertext $C = K \times_{A_1} M$ (digit by digit)
Step 1:	Pre-compute $K_i = K \times_{A_1} i$, $\forall i \in \{1, 2, ..., b\}$.
Step 2:	$Y \leftarrow K_d$
Step 3:	$X \leftarrow Y'$
Step 4:	$j \leftarrow 1$
Step 5:	while (stream continues)
Step 5a:	$Y \leftarrow X +_{A_1} K_{m_j}$
Step 5b:	$X \leftarrow Y'$, $s_j \leftarrow (Y)_1$
Step 5c:	Send s_j
Step 5d:	$j \leftarrow j + 1$
	end while

Figure 6: Encryption

8.2 Decryption

By construction, the AIA, A_1, is regular, so $\oplus$ and $\otimes$ are left-cancellative. I.e. $\forall i, j \in \{1, 2, 3, 4\}$, let $\sigma_i, \gamma_i \in Sym_b$ such that $(i \times j)_r = \sigma_i(j)$ and $(i + j)_r = \gamma_i(j)$. We now prove the following.

Theorem 1. *Every regular AIA $(b, \times_s)$ is left-cancellative. I.e. if K, M_1, and M_2 are base-b strings of length n, p and p respectively, then $K \times_s M_1 = K \times_s M_2$, then $M_1 = M_2$. Furthermore $(b, \times_s)$ has a left division algorithm.*

Proof. We prove both ascertains simultaneously by uniquely recovering M from the product $D = K \times_s M$. We assume knowledge of K. Let $K = k_n k_{n-1} \cdots k_1$, $M = m_p m_{p-1} \cdots m_1$, and let $D = d_{p+n} d_{p+n-1} \cdots d_1 = K \times_s M$. The formula for each digit of D could be give recursively using remainders and carries of digits from K and M. However, to further demonstrate the similarity to integer arithmetic, we express the multiplication in long-hand as follows.

$$
\begin{array}{cccccccc}
 & & & k_n & \cdots & k_2 & k_1 \\
\times_s & & & m_p & \cdots & m_2 & m_1 \\
\hline
 & & x_{1,p+1} & x_{1,p} & \cdots & x_{1,2} & x_{1,1} \\
 & x_{2,p+1} & x_{2,p} & \cdots & x_{2,2} & x_{2,1} & \\
\hline
 & x_{3,p+1} & x_{3,p} & \cdots & x_{3,2} & x_{3,1} & \\
 & & & & \cdot & & \\
 & & & & \cdot & & \\
 & & & & \cdot & & \\
\hline
x_{p,p+1} & x_{p,p} & \cdots & x_{p,2} & x_{p,1} & & \\
\hline
d_{p+n} & & \cdots & d_{p+1} & d_p & \cdots & d_2 & d_1
\end{array}
$$

The variable $x_{i,j}$, is used to denote the as of yet unknown digit $(K \times_s m_i)_j$. We begin recovering M, by noticing that $(k_1 \otimes_s m_1)_r = x_{1,1} = d_1$. Therefore, $\sigma_{k_1}^{-1}(d_1) = m_1$ and we can compute $K \times_s m_1 = x_{1,p+1} x_{1,p} \cdots x_{1,2} x_{1,1}$.

The rightmost digit, $x_{1,1}$ plays no further role in computation. It is for this reason that we can transmit it during encryption and save only $(x_{1,p+1} x_{1,p} \cdots x_{1,2} x_{1,1})'$. Continuing on, we have $(x_{1,2} \oplus_s x_{2,1})_r = d_2$. Therefore $\gamma_{x_{1,2}}^{-1}(d_2) = x_{2,1}$. But we also have $(k_1 \otimes_s m_2)_r = x_{2,1}$, so $\sigma_{k_2}^{-1}(d_2) = m_2$ and we can recover $K \times_s m_2 = x_{2,p+1} x_{2,p} \cdots x_{2,2} x_{2,1}$.

Continuing on in this way, we can recover all p of the rows and the number M. □

This stream cipher could be easily adapted to use multiple Regular PAIA's as described in the authentication protocol. Successful decryption would still be possible because of the stipulation that the strings K_1, K_2, K_3, K_4 that make up K, all have a different right-most digit.

Figure 7 gives a more succinct description of the decryption algorithm.

Note that only the right-most p digits of the ciphertext are necessary to uniquely recover M. This makes it necessary, when implementing the stream cipher, for the person encrypting to append a variable length garbage string to prevent the poor avalanche property discussed in section 8.4. However, the garbage string will be short compared to the length of the message and so will not greatly reduce the efficiency of the stream cipher.

A tool for analyzing regular AIAs stems from the fact that only right-most p digits are necessary for unique decryption. If we define a truncating operation, T_p, that projects any string of p digits or more, onto the substring of its right-most p digits then the composition of T_p and $\times_{A_1}$ associates to each fixed K_1, an action on the base-b strings of length p. The following theorem shows that with further restrictions on regular AIAs, we can produce a permutation quasigroup.

Definition 5 (Restricted AIA). *A regular AIA, $(b, \times_s)$, is called restricted if it also satisfies the following two properties.*

1. *The $b \times b$ table, whose (i, j) entry is $(i \otimes_s j)_r$, is a latin square.*

Description	Decoding messages
Input:	$s_j, \forall j \in \mathbb{N}$
Output:	M
Step 1:	$X \leftarrow (K_1 \times_{A_1} d)'$
Step 1a:	$m_1 \leftarrow \sigma_{k_1}^{-1}(\gamma_{(X)_1}^{-1}(s_1))$.
Step 2:	$X \leftarrow (X +_{A_1} L_{m_1})'$.
Step 3:	$j \leftarrow 2$
Step 4:	while (stream continues)
Step 4a:	$m_j \leftarrow \sigma_{k_1}^{-1}(\gamma_{(X)_1}^{-1}(s_j))$, output m_j
Step 4b:	$X \leftarrow (X +_{A_1} L_{m_j})'$.
Step 4c:	$j \leftarrow j + 1$
	end while

Figure 7: Decryption

2. The $b \times b$ table, whose (i, j) entry is $(i \oplus_s j)_r$, is a latin square.

Theorem 2. *If $(b, \times_s)$ is a restricted AIA, then $(b, \times_s)$ is both left-cancellative and right-cancellative. I.e., $(b, \times_s)$ is a quasi-group. Furthermore, on strings of length p, the composition of T_p and $\times_s$ yields a non-associative permutation representation of $(b, \times_s)$ of degree d^p.*

Proof. The fact that $(b, \times_s)$ is left-cancellative follows directly from Theorem 1. The proof that $(b, \times_s)$ is also right-cancellative is similar except we recover, at each step, the intermediate columns (diagonals) of the product instead of the intermediate rows. The details are left to the reader. The final ascertain is easily deduced from the fact that restriction of $T_p \circ \times_s$ to strings of a fixed length p yields a finite quasigroup whose right regular representation consists of non-associative permutations of base-b strings of length p. □

One implication of this result, is that rather than performing a bruteforce search to recover the secret key, an attacker may restrict him/her-self to restricted AIAs and PAIAs, thus possibly decreasing the keyspace. See Section **??** for details.

8.3 Speed

Performing the operations $(i \oplus j)$ and $(i \otimes j)$ can be viewed as looking up the (i, j) entry of the respective table to find the corresponding remainder and carry. We can then speak of the speed of encryption and decryption in terms of the number of table look-ups required to perform the respective operations. Furthermore, for small enough base b, the table lookups can be seen as comparable to bit operations. For example, if $b = 4$, then multiplication table will have 16 ordered pairs and addition will have 25 (recall addition by 0 is allowed but the value is pre-set and so does not have to be stored as part of A_1.) Bit operations (binary additions) can be view as table look-ups with a table of size 8, the set of all ordered binary triples. Of course table look-ups for the present stream cipher, are ordered pairs and so will have to be performed more often. The following theorem details the speed of encryption and decryption.

Theorem 3. *The number of table look-ups required for encrypting a base-b string of length p is $3(n-1)b+1+2n(p+c-1)$ where c is the variable (short) length of of a garbage string*

appended to the message prior to encryption. Decryption requires $3(n-1)b+1+2n(p+c-1)+2(p+c-1)$ *table-lookups.*

Proof. Note that n and b are fixed constants so the constant, $3(n-1)b+1$, reflects the pre-computation required to produce the b strings that represent $K_1 \times_{A_1} i$, for $i \in \{1,2,...,b\}$. To see that this number is correct, notice that producing the right-most digit of $K_1 \times_{A_1} i$, for fixed i, requires a single table look-up for digit-wise multiplication. For each subsequent digit, j, of $K_1 \times_{A_1} i$, we have an additive carry, x_j, from caculating the previous digit and can perform the following table look-up substitutions.

1. $(i,j) \rightarrow ((j \otimes_{A_1} i)_r, (j \otimes_{A_1} i)_c)$
2. $(x_j, (j \otimes_{A_1} i)_r) \rightarrow ((x_j \oplus_{A_1} (j \otimes_{A_1} i)_r)_r, (x_j \oplus_{A_1} (j \otimes_{A_1} i)_r)_c)$
3. $(j \otimes_{A_1} i)_c, (x_j \oplus_{A_1} (j \otimes_{A_1} i)_r)_c) \rightarrow (((j \otimes_{A_1} i)_c \oplus_{A_1} (x_j + \oplus_{A_1} (j \otimes_{A_1} i)_r)_c)_r, 0)$

The first output digit of step two,$(x_j \oplus_{A_1} (j \otimes_{A_1} i)_r)_r$, will be the digit of $K_1 \times_{A_1} i$ and the first ouput digit of step 3, $((j \otimes_{A_1} i)_c \oplus_{A_1} (x_j + \oplus_{A_1} (j \otimes_{A_1} i)_r)_c)_r$ will be the new carry.

The message, together with the appended garbage string is $p+c$ digits long so we will have to add $p+c-1$ pairs of $n+1$-digit strings together. for each string-wise addition, the right-most digit is left unaltered and the two table look-ups are required to sum each pair of subsequent digits. The first table look-up adds the carry to the corresponding digit of the top row (note that this will not produce a carry), and the second will sum the result of the first table look-up to the corresponding digit on the lower row. In total, summation takes $(n+1-1) \times (p+c-1)$ table look-ups.

Finally, decryption is performed in the same manner with two additional table look-ups required for each string-wise addition. The first table look-up retrieves the right-most digit of the current row, i, to be added. This table look-up corresponds to the appropriate inverse permutation γ^{-1} of $\oplus_{A_1}$. The second table look-up retrieves, via the appropriate inverse permutation σ^{-1} of $\otimes_{A_1}$, the digit m_i. □

It is noteworthy that n and b are constants and c can be bounded by n so that the encryption and decryption are both linear in p which is log_b of the length of the message. Therefore, increasing b will decrease the number of table look-ups required for encryption and decryption, but in general, this will probably not increase the over all speed of the these algorithms because the size of the tables being searched will grow as a function of the square of b.

9 Logic Gate Usage

Special thanks to Duane Currie for his contributions to the following analysis.

In this section we assume the multiple PAIA implementation, and show that the addition of a pair of 11 digit (base-4) numbers can be performed using between 300 and 420 logic gates, depending on the choice of K. We demonstrate an upper bound on the average number of logic gates needed to perform a single digit adder. Concatenating ten of these single digit adders produces the desired addition machine. Furthermore, adding the final carry (left-most digit) can be performed using only a few gates. The upperbound we produce is approximately 50 gates per digit adder, but simple logical reduction techniques can be used to bring this average down to 42 gates. Furthermore, a large percentage of the adders require 30 or fewer gates after this reduction. Given that there are $2^{165.85}$ possible ten digit adders, we can restrict ourselves to those with 300 or fewer gates, without compromising the 80 or 90 or even 128 bits of security we are after.

Note that in what follows $\oplus$ still refers to the digit-wise addition defined by K. As with conventional adding machines We can link ten single digit adders in sequence so it suffices to calculate the number of gates required to calculate the sum of one single-digit additive carry c_i and a pair of two-digit numbers $a = a_1a_2$ and $b = b_1b_2$. The steps will be as follows.

1. Calculate $a' = c_i \oplus a$
2. Calculate $a' \oplus 1$, $a' \oplus 2$, $a' \oplus 3$, and $a' \oplus 4$
3. Use a multiplexer on input b to select $a' \oplus b$.

For step 1, recall that $0 \oplus a = a$ and $1 \oplus a$ does not produce a carry and so can be seen as just a permutation of $\{00, 01, 10, 11\}$. Listing the possible bits of a' we need to fill in the last two columns of the following table where the rows are a permutation of $\{00, 01, 10, 11\}$.

a_1	a_2	a'_1	a'_2
0	0		
0	1		
1	0		
1	1		

It is clear that each of the columns must have two 1's and two 0's of which there are six possible columns. The possible columns are listed below. The number of logic gates and the logical operation needed to produce each from a' appears below each column.

0	1	1	0	1	0
0	1	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	1
a'_1	$\neg a'_1$	a'_1 XNOR a'_2	a'_1 XOR a'_2	$\neg a'_2$	a'_2
0	1	4	4	1	0

Figure 8: Possible Permutation Columns

Once the first column of the permutation is chosen there remain 4 choices for the second column as re-choosing the selected column or choosing its negation would not yield a permutation. The average gate count is then 3.33.

For step 1, we will require four more permutations (one is duplicate but recopied) adding 13.33 more gates to the count and, for the carries, 4 extra columns chosen at random from the 2^4 possibilities. Of these columns, two require 4 gates, six require 2 gates, four require 1 gate and four don't require any gates. This gives an average of 1.5 gates for each carry column and 6 gates in all.

For step 3, the multiplexer will require six gates to produce the four bits which essentially select i for $i \in \{1, 2, 3, 4\}$. Each bit is then Anded with the appropriate $a' \oplus i$ (where $a' \oplus i$ is three bits) requiring an additional $3 \times 7 = 21$ gates for a total of 27 gates.

Thus the total is $27 + 3.33 + 13.33 + 6 = 49.66$ gates per digit for an overall average gate count of ≈ 500. However, as mentioned above, we can reduce this using logical reductions to 420 gates and even further reduce this to 300 gates without compromising the desired level of security.

10 Security Analysis of the Stream Cipher

Stream cipher ciphertext, resulting from the randomly generated plaintext, underwent the sixteen statistical tests developed by the National Institute of Standards and Technology to detect non-randomness in binary sequences. Each test was conducted on one hundred samples, and all sixteen tests were passed.

In the least secure case, the set of all possible keys (K, A_1, d), where A_1 is a single AIA, K is a string of n base-b digits, and d is a single digit, forms a keyspace of size $b^n \times b^{b^2} \times 2^{b^2-b} \times (b!)^{2b} \times 2^2$. However, an attacker may also choose to view the secret key as a set of b strings of length $n+1$, together with only the addition portion of A_1, which we label PA_1. Thus the we can also view the keyspace as the set of all $(\{K_1, ..., K_b\}, PA_1)$ of which there are $2^4 \times b^{4n} \times 2^{b^2-b} \times (b!)^b$. Therefore the security against exhaustive search is

$$min\{b^n b^{b^2} 2^{b^2-b} (b!)^{2b} 2^2, 2^4 b^{4n} 2^{b^2-b} (b!)^b\}$$

$$= 2^{b^2-b+2} b^n (b!)^b \times min\{2^2 b^{3n}, b^{b^2} (b!)^b\}$$

For $n = 10$ and $b = 4$, as in the example above, this gives approximately $2^{102.68}$ possible keys of the form (K, A_1, d) and approximately $2^{114.34}$ possible keys of the form $(\{K_1, ..., K_b\}, PA_1)$. However, there may exist another (or several) key(s) (K', A_1', d') such that $K \times_{A_1} M = K' \times_{A_1'} M'$ for all M where M' is the same as M but with d' instead of d as the right-most digit. Equivalently there may be other $(\{K_1', ..., K_b'\}, PA_1')$ which have an action identical to $(\{K_1, ..., K_b\}, PA_1)$, on all M.

For regular AIAs, (PAIAs), the operation $\times_{A_1}$, $(+_{PA_1})$ is left-cancellative but not necessarily right-cancellative unless we use only restricted AIAs, (PAIAs). There are 576 Latin squares using the digits $1, 2, 3$, and 4. If we restrict possible A_1 in this way we reduce the keyspace to $4^{10} \times 4^{4^2} \times 2^{4^2-4} \times (576)^2 \times 2^2 \approx 2^{84.34}$. On this set we are guaranteed that if $A_1 = A_1'$ then $d = d'$ and $K = K'$, so we can consider the actions of each (K_1, d) to be unique for fixed A_1. However, the number of collisions where $A_1 \neq A_1'$ still needs to be investigated. To date however, there is no known efficient algorithm (other than brute force) for identifying when such collisions occur. If we use only a restricted PAIAs to produce keys of the form $(\{K_1, ..., K_b\}, PA_1)$, then for $n = 10$, $b = 4$, we get approximately $2^{105.17}$ keys.

A final note is that concatenating n distinct AIAs or PAIAs to perform the encryption will not significantly alter the logic gate count, or the speed of the algorithm but will greatly increase the size of the keyspace. The only drawback to doing this is that the private key stored in the readers memory will be longer.

10.1 Weaknesses as a block-cipher

It bears mentioning that the stream cipher described above could be adapted easily into a block cipher. Simply break up the plaintext stream into blocks of length, say p, and encrypt each block in turn by multiplying by K_1 subject to the operation dictated by A_1. However, such a block cipher would have very poor plaintext avalanche properties since two plaintext blocks, M_1 and M_2, that differ only in the leftmost digit would encrypt to ciphertexts where the rightmost $p-1$ digits are identical.

It is also worth noting that the flaw in the block-cipher does not appear to be exploitable to attack the authentication scheme given above because even in an active attack, one of either the tag or the reader would be considered a legitimate user and hence would be producing every second digit of M at random.

References

[1] Atkinson, J., Contactless Credit Cards Consumer Report 2006, http://www.findcreditcards.org/, 2006.

[2] Czeslaw Ko S Cielny, A New Approach to the Elgamel Encryption Scheme, *Int. J. Appl. Math. Comput. Sci.*, 14, No. 2, (2004), 265-267.

[3] Gligoroski, D., Stream cipher based on quasigroup string transformations in Z_p^*, accepted in *Macedonian Academy of Science and Arts*, Proceedings in Mathematical and Technical Sciences (2004)

[4] Juels, Ari; Weis, Stephen A., Authenticating pervasive devices with human protocols *Advances in cryptology—CRYPTO 2005*, 293–308, Lecture Notes in Comput. Sci., 3621, Springer, Berlin, 2005.

[5] Kock, Anders, Classifying Surjective Equivalences, http://home.imf.au.dk/kock/classf.pdf, (2004)

[6] Markovski, S.; Gligoroski, J., Classification of Quasigroups by Random Walk on Torus, accepted for *IJCAR workshop on Computer Supported mathematical Theory Development*, Cork, Ireland (2004)

[7] Pedro Peris-Lopez, Julio Cesar Hernandez-Castro, Juan Estevez-Tapiador, Arturo Ribagorda., M2AP: A Minimalist Mutual-Authentication Protocol for Low-cost RFID Tags *International Conference on Ubiquitous Intelligence and Computing - UIC'06*, 2006.

[8] Weis, S.; Sarma, S.; Rivest, R.; Engles, D., Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems, *Lecture Notes in Comput. Sci.* vol. 2802 (2004) ,pp. 201-212.

[9] Wenstein, Alan, Groupoids: unifying internal and external symmetry, *Notices of the AMS*, 43, No. 7, 744-752