3.3. Using a Reconfigurable Computer to Compute Algebraic Immunity

M. Eric McCay Jon T. Butler

Pantelimon Stănică

3.3.1. Why do we Need Algebraic Immunity?

The benefits we enjoy from the internet depend critically on our ability to communicate securely. For example, to bank online we depend on the encoding of plaintext messages (our personal identifications numbers, for instance), which, if divulged, could result in significant loss of personal funds. Unfortunately, the theft of credit/debit card numbers is one of the most common crimes in the internet age. Such a theft is especially enticing since the individual committing the theft is likely to be far away from the victim, therefore feeling immune to prosecution.

Encryption is a process of converting a plaintext (cleartext) message into a scrambled message, called *ciphertext*. With the internet as is typically configured now, actual interception of a message is easy; anyone with access to a server and some programming skills can easily acquire the communication. However, in the case of an encrypted message, it is necessary to then decrypt the message, and in so doing extract the original plaintext message. The present benefits of the internet depend critically on the degree of difficulty associated with the discovery of the plaintext message.

For example, one could encrypt a message by exclusive ORing a code (the same code) on each ASCII character in the message. In this case, 'A' is *always* transformed into some other letter, like 'K', 'B' is *always* transformed to say 'T', etc. Breaking such a code is easy, by using statistical analysis, since one can guess that the most common letter represents an 'E', while a rare letter represents a 'Z'. Although the guess may be wrong, with enough computing power and enough of the encrypted message, a dedicated thief would eventually succeed.

Present day encryption algorithms are significantly more sophisticated than this simple example. However, in addition to much better encryption algorithms, there has also been a significant increase in computing power. Indeed, we live in an era of the *supercomputer on the desk*, in which thousands of processors exist within the space that recently housed only one processor. Therefore, there is an enhanced ability to *try out* many guesses and to analyze huge quantities of ciphertext.

An attempt to decipher a message is called a (cryptographic) *attack*, or *cryptanalysis*. An attack is often predicated on a perceived weakness in the encoding algorithm. For example, one of the earliest attacks, called the linear attack, is most successful when the encoding algorithm uses a not too complicated approximation of a linear (or affine) Boolean function. A linear (affine) function is the exclusive OR of linear terms only (or their complement). For example, $f_1 = x_1 \oplus x_3 \oplus x_4$ is a linear function (both f_1 and $f_1 \oplus 1$ are affine), while $f_2 = x_1 x_2 \oplus x_3 x_4$ is not linear (nor affine as it has degree two). In a linear attack, the attacker takes advantage of an affine approximation of the action of the cipher. If the function used in some step of the cipher is linear or even 'close to linear', such an attack is likely to succeed. To mitigate against such attacks, Boolean functions that are highly nonlinear are used in the encryption instead. This led to interest in *bent* functions, Boolean functions whose nonlinearity exceeds that of all others. For a tutorial description of bent functions, see [147].

Since Rothaus' seminal contribution on bent functions in 1976 [226], there has been much work on the cryptographic properties of Boolean functions [313]. Such properties include strict avalanche criterion [138, 244], propagation criteria [12], and correlation immunity [312]. However, within the past 10 years an effective attack that uses Gaussian elimination has emerged [222, 223].

Any stream or block cipher can be described by a system of equations expressing the ciphertext as a function of the plaintext and the key bits. An *algebraic attack* is simply an attempt to solve this system of equations for the plaintext. If the system happens to be overdefined, then the attacker can use linearization techniques to extract a solution. However, in general, this approach is difficult, and not effective, unless the equations happen to be of low degree. That is (somewhat) ensured

if, for instance, the nonlinear Boolean function combiner in an LFSR-based generator (a widely used encryption technique) has low degree or the combiner has a low algebraic immunity (defined below) [222, 223].

Let be $f(x_1, x_2, \ldots, x_n)$ a Boolean function that depends on the bits $\{x_1, x_2, \ldots, x_n\}$ of a linear feedback shift register (LFSR). Let be L another function that defines the LFSR. Specifically,

$$L : \{0,1\}^n \to \{0,1\}^n$$

defines how the values of the LFSR bits at some clock period depend on the bit values in the previous clock period. Suppose the keystream $z_0, z_1, z_2, \ldots$ is computed from some initial secret state (the *key*) given by n bits $a_0, a_1, \ldots, a_{n-1}$ in the following way. Let $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$ be the initial state and define the keystream bits by

$$\begin{aligned} z_0 &= f(\mathbf{a}) \\ z_1 &= f(L(\mathbf{a})) \\ z_2 &= f(L^2(\mathbf{a})) \\ \vdots & \quad \vdots \\ z_t &= f(L^t(\mathbf{a})) \ . \end{aligned}$$

The problem of extracting the plaintext message in this context is equivalent to the problem of finding the initial key $\mathbf{a}$, knowing L and f, and intercepting z_i. Assume that f is expressed in its algebraic normal form (ANF). Specifically,

$$f(x_1, x_2, \ldots, x_n) = c_0 \oplus c_1 x_1 \oplus c_2 x_2 \oplus \ldots c_{2^n-1} x_1 x_2 \ldots x_n \ ,$$

where $c_i \in \{0,1\}$ is uniquely determined by $\mathbf{c}$. Let $\deg(f) = d$ be the number of variables in the term of the ANF with the *largest* number of variables. Since $\deg(f) = d$, every term on the right hand side of any equation in the above set of equations has d or fewer variables. Therefore, there are $M = \sum_{i=0}^{d} \binom{n}{i}$ or fewer of these terms, and we define a variable y_j for each one of them. If a cryptanalyst has access to at least $N \geq M$ keystream bits z_t, then he/she can solve the linear system of N equations for the values of the variables y_j, and thus recover the values of $a_0, a_1, \ldots, a_{n-1}$. If d is not large, then the

cryptanalyst may well be able to acquire enough keystream bits so that the system of linear equations is highly overdefined (that is, N is much larger than M).

If we use Gaussian reduction to solve the linear system, then the amount of computation required is $O\left(\binom{n}{d}^{\omega}\right)$, where ω is the well–known *exponent of Gaussian reduction* ($\omega = 3$ (Gauss-Jordan [105]); $\omega = \log_2 7 = 2.807$ (Strassen [326]); $\omega = 2.376$ (Coppersmith-Winograd [71])). For $n \geq 128$ and $d \sim n$, we are near the upper limits for which this attack is practical for actual systems, since the complexity grows with d.

Courtois and Meier [222] showed that if one can find a function g with small degree d_g such that $fg = 0$ or $(1 \oplus f)g = 0$, then the number of unknowns for an algebraic attack can be reduced from $\binom{n}{d_f}$ to $\binom{n}{d_g}$. We say that g is an *annihilator* of f. That is easy to see, since $f(L^i(\mathbf{a})) = z_i$ becomes $g(L^i(\mathbf{a})) \cdot f(L^i(\mathbf{a})) = 0 = z_i\, g(L^i(\mathbf{a}))$, and so, we get the equations $g(L^i(\mathbf{a})) = 0$, whenever the intercepted $z_i \neq 0$. That gives us a reduction in complexity, from

$$O\left(\binom{n}{d_f}^{\omega}\right) \text{ to } O\left(\binom{n}{d_g}^{\omega}\right) .$$

Therefore, it is necessary to have a fast computation of a low(est) degree annihilator of the combiner f.

3.3.2. Why do we Need a Reconfigurable Computer?

Although the computations needed in an attack or a defense can be done by a conventional computer, a significant speedup can be achieved by using a reconfigurable computer. For example, it has been shown that a reconfigurable computer can compute bent functions for use in encryption at a speed that is 60,000 times faster than by a conventional computer [146]. The speedup is achieved because a reconfigurable computer can implement many copies of basic logic elements, like adders and comparators, which can then execute simultaneously. However, in a conventional computer, there are fast logic elements, like adders, but there are relatively few of them. A

conventional computer is limited in what it can efficiently compute because its architecture is limited. As another example, the effectiveness of a reconfigurable computer has been shown in computing Boolean functions with high correlation immunity, which are more immune to correlation immunity attacks [92]. A speedup of about 190 times was achieved.

In this section, we show that a reconfigurable computer is effective in computing the algebraic immunity of a Boolean function. This study represents a departure from the two studies involving a reconfigurable computer described above. That is, compared to nonlinearity and correlation immunity, the computation of the algebraic immunity of a Boolean function is more complex. Given human inclinations, it would seem that algebraic immunity computations should be relegated to conventional computers rather than reconfigurable computers. However, we show that his intuition is wrong; reconfigurable computers still hold an advantage over conventional computers.

3.3.3. Background and Notation

We start with a combinatorial property of a Boolean function that also plays an important role in the cryptographic world (see [313] for more cryptographic properties of Boolean functions).

Definition 3.6. *The* **degree** d *of a* **term** $x_{i_1}x_{i_2}\ldots x_{i_d}$ *is the number of distinct variables in that term, where* $i_j \in \{1, 2, \ldots, n\}$*, and* n *is the total number of variables.*

Definition 3.7. *The algebraic normal form or* **ANF** *of a Boolean function* $f(x_1, x_2, \ldots, x_n)$ *consists of the exclusive OR of terms; specifically,* $f(x_1, x_2, \ldots, x_n) = c_0 \oplus c_1x_1 \oplus c_2x_2 \oplus \ldots \oplus c_{2^n-1}x_1x_2\ldots x_n$*, where* $c_i \in \{0, 1\}$*.*

The ANF of a function is often referred to as the **positive polarity Reed-Muller** form.

Definition 3.8. *The* **degree**, $\deg(f)$*, of function* $f(x_1, x_2, \ldots, x_n)$ *is the largest degree among all the terms in the ANF of* f*.*

Table 3.2. Functions that annihilate the 3-variable majority function f and their degree

$x_1x_2x_3$	f	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}	α_{11}	α_{12}	α_{13}	α_{14}	α_{15}
000	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
001	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
010	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0
011	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
100	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
101	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
110	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
111	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Degree		2	3	3	2	3	2	2	3	3	2	2	3	2	3	3

Example 3.1. *The ANF of the 3-variable majority function is*

$$f(x_1, x_2, x_3) = x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \ .$$

Its degree is 2.

Definition 3.9. *Function $a \neq 0$ is an* **annihilator** *of function f if and only if $a \cdot f = 0$.*

Note that $\bar{f}$ is an annihilator of f. Further, if a is an annihilator of f, so also is α, where $\alpha \leq a$ ('$\leq$' is a partial order on the set of vectors of the same dimension, that is, $(\alpha_i)_i \leq (\beta_i)_i$ if and only if $\alpha_i \leq \beta_i$, for any i).

Definition 3.10. *Function f has* **algebraic immunity** *k where*

$$k = \min\{\deg(a) \mid a \text{ is an annihilator of } f \text{ or } \bar{f}\} \ .$$

Example 3.2. *The annihilators of the 3-variable majority function $f(x_1, x_2, x_3) = x_1x_2 \oplus x_1x_3 \oplus x_2x_3$ include $\bar{f}$ and all g such that $g \leq \bar{f}$, excluding the constant 0 function. In all, there are 15 annihilators of f and 15 annihilators of $\bar{f}$. Among these 30 functions, the minimum degree is 2. Thus, $f(x_1, x_2, x_3) = x_1x_2 \oplus x_1x_3 \oplus x_2x_3$ has algebraic immunity 2. Table 3.2 shows all 15 annihilators (labeled $\alpha_1, \alpha_2, \ldots, \alpha_{15}$) of the 3-variable majority function. Table 3.3 shows all 15 annihilators*

Table 3.3. Functions that annihilate the complement of the 3-variable majority function

$x_1x_2x_3$	$\bar{f}$	β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	β_9	β_{10}	β_{11}	β_{12}	β_{13}	β_{14}	β_{15}
000	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
001	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
010	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
011	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
100	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
101	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0
110	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
111	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Degree		2	3	3	2	3	2	2	3	3	2	2	3	2	3	3

(labeled $\beta_1, \beta_2, \ldots, \beta_{15}$) of the complement of the 3-variable majority function. Across both of these tables, one can verify that the minimum degree among annihilators is 2. It follows that the algebraic immunity of the 3-variable majority function is equal to 2.

Example 3.3. *Let $n = 4$. The function $f = x_1x_2x_3x_4$ has the highest degree, that is equal to 4. Function $a = \bar{x}_1 = x_1 \oplus 1$ annihilates f, since $a \cdot f = \bar{x}_1x_1x_2x_3x_4 = 0$. Since, $\bar{x}_1$ has degree 1 and there exists no annihilator of f of degree 0, the algebraic immunity of f is equal to 1. To reach this conclusion, it is not necessary to check the annihilators of $\bar{f}$, since the only annihilator of $\bar{f}$ is f, which has degree 4.*

We can immediately state the following lemmas.

Lemma 3.6. *The algebraic immunity of a function $f(x_1, x_2, \ldots, x_n)$ is identical to the algebraic immunity of $\bar{f}$.*

Lemma 3.7. *The algebraic immunity of a function $f(x_1, x_2, \ldots, x_n)$ is* $\min\{\deg(\alpha) | \alpha \leq \bar{f}$ *or* $\alpha \leq f\}$.

Proof. The hypothesis follows immediately from the observation that $\{\alpha | \alpha \leq \bar{f}$ or $\alpha \leq f\}$ is the set of all annihilators of f. □

Lemma 3.7 is similar to Definition 3.10. However, there is an important difference. Lemma 3.7 admits an algorithm for determining the algebraic immunity of a function f. Specifically, examine the degree of each function α such that $\alpha \leq \bar{f}$ and determine the minimum degree of the ANF among all α. This requires the examination of $2^{2^n - wt(f)} - 1$ functions, where $wt(f)$ is the number of 1's in the truth table of f, since $\bar{f}$ has $2^n - wt(f)$ 1's in its truth table. In forming an annihilator, each 1 can be retained or set to 0. The '-1' accounts for the case where all 1's are set to 0, which is not an annihilator. The following result is essential to the *efficient* computation of algebraic immunity.

Lemma 3.8. *[222, 330] The algebraic immunity AI(f) of a function* $f(x_1, x_2, \ldots, x_n)$ *is bounded above; specifically, AI(f)* $\leq \lceil \frac{n}{2} \rceil$.

3.3.4. Computation of Algebraic Immunity

Row Echelon Reduction Method. The computation of algebraic immunity is more complex than nonlinearity and correlation immunity, two cryptographic properties that have been previously computed by a reconfigurable computer. There are several methods for computing the algebraic immunity of a Boolean function f. In the brute force method, one checks every function to see if it is an annihilator of the given f, or its complement. In another approach, one can identify directly the annihilators with high degree. Since our attempt is to implement the algebraic immunity computation on a reconfigurable computer, we have not implemented the more recent algorithm of Armchnecht et al. [94], which also deals with *fast* algebraic attack issues. Our approach is based on a simpler version of that linear algebra approach. This enabled us to implement it on the SRC-6 reconfigurable computer and to display the algebraic immunity profiles for all functions on n variables, for $2 \leq n \leq 5$. A similar approach has been used to compute algebraic immunity on a conventional processor [222, 330]. Our implementation is the first known using Verilog on an FPGA.

Here, we create the ANF of a minterm corresponding to each 1 in the truth table of the function. Our approach to solving this system is to

express this in reduced row echelon form using Gaussian elimination. It is based on two elementary row operations: (1) interchange two rows, and (2) add one row to another row. A simple test applied to the reduced row echelon form determines if there is an annihilator of some specified degree.

Our approach is to express the system of linear equations in reduced row echelon form, and, from this, determine if there exists a solution of some specified degree. A matrix is in *row echelon form* if it satisfies the following conditions:

1. the first nonzero element (leading entry) in each row is 1

2. each leading entry is in a column to the right of the leading entry in the previous row

3. rows with all zero elements (if any) are below rows having a nonzero element.

To put the matrix in *reduced* row echelon form, one simply uses elimination on nonzero entries above each pivot.

Example 3.4. *To illustrate, consider solving the algebraic immunity of the majority function* $f(x_1, x_2, x_3) = x_1x_2 \oplus x_1x_3 \oplus x_2x_3$. *The top half of Table 3.4 shows the minterm canonical form of* $\bar{f}$. *Here, the first (leftmost) column represents all binary three tuples on three variables. The second column contains the truth table of the complement function* $\bar{f}$, *which is expressed as* $\bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3$, *or more compactly, as the sum of minterms* $m_0 \vee m_1 \vee m_2 \vee m_4$. *This represents the annihilator of* f *with the most* 1*'s.*

The columns are labeled by all possible terms in the ANF of an annihilator. Then, 1's are inserted into the table to represent the ANF of the minterms. For example, since the top minterm, $m_0 = \bar{x}_1\bar{x}_2\bar{x}_3 = (x_1 \oplus 1)(x_2 \oplus 1)(x_3 \oplus 1) = x_1x_2x_3 \oplus x_2x_3 \oplus x_1x_3 \oplus x_1x_2 \oplus x_3 \oplus x_2 \oplus x_1 \oplus 1$, *its ANF has all possible terms, and so, there is a 1 in every column of this row.*

Note that we can obtain the ANF of some combination of minterms as the exclusive OR of various rows in the top half of Table 3.4. This

Table 3.4. Functions that annihilate the 3-variable majority function

ANF Coefficient →			c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0	Minterms
Index	$x_1x_2x_3$	$\bar{f}$	$x_1x_2x_3$	x_2x_3	x_1x_3	x_1x_2	x_3	x_2	x_1	1	
Original											
0	000	1	1	1	1	1	1	1	1	1	m_0
1	001	1	1	1	1	0	1	0	0	0	m_1
2	010	1	1	1	0	1	0	1	0	0	m_2
3	011	0	0	0	0	0	0	0	0	0	
4	100	1	1	0	1	1	0	0	1	0	m_4
5	101	0	0	0	0	0	0	0	0	0	
6	110	0	0	0	0	0	0	0	0	0	
7	111	0	0	0	0	0	0	0	0	0	
Reduced Row Echelon Form											
0	-	-	1	0	0	0	1	1	1	0	$m_1 \oplus m_2 \oplus m_4$
1	-	-	0	1	0	0	1	1	0	1	$m_0 \oplus m_4$
2	-	-	0	0	1	0	1	0	1	1	$m_0 \oplus m_2$
3	-	-	0	0	0	1	0	1	1	1	$m_0 \oplus m_1$
4	-	-	0	0	0	0	0	0	0	0	
5	-	-	0	0	0	0	0	0	0	0	
6	-	-	0	0	0	0	0	0	0	0	
7	-	-	0	0	0	0	0	0	0	0	

follows from the observation that $m_i \vee m_j = m_i \oplus m_j$ *due to the orthogonality of the minterms. For example, one annihilator* a *is*

$$a = \bar{x}_1 x_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 \ ,$$

and so the ANF of a *is generated by simply exclusive ORing the rows associated with these two minterms.*

Elementary Row Operations. Consider a $0-1$ matrix and two row operations:

1. interchange one row with another, and

2. replace one row by the exclusive OR of that row with any other row.

Using elementary row operations, we seek to create columns, starting with the left column with *only one* 1 (called a *pivot*).

Definition 3.11. *A* 0−1 *matrix is in* **row echelon form** *if and only if all nonzero rows (if they exist) are above any rows of all zeroes, and the leading coefficient (pivot) of a nonzero row is always strictly to the right of the leading coefficient of the row above it.*

Definition 3.12. *[255] A* 0−1 *matrix is in* **reduced row echelon form** *if and only if it is in row echelon form and each leading* 1 *(pivot) is the only* 1 *in its column.*

Consider Table 3.4. The top half shows the truth table of $\bar{f}$. For each value 1 (minterm – m_0, m_1, m_2, and m_4) in the $\bar{f}$ column (third column), the ANF of that minterm is expressed across the rows. To form an annihilator of f, we must combine one or more minterms using the exclusive OR operation. The bottom half of Table 3.4 shows the reduced row echelon form of the top half. The row operations we used to derive the bottom half from the top half can be inferred from the rightmost column. For instance, the entry $m_0 \oplus m_1$ in the bottom half of the table indicates that the rows labeled m_0 and m_1 in the top half of the table were combined using the exclusive OR operation.

Note that, like the top half of Table 3.4, the rows in the reduced row echelon form combine to form *any* annihilator of the original function. This follows from the fact that any single minterm can be formed as the exclusive OR of rows in the reduced row echelon form. For example, m_1 is obtained as the exclusive OR of the top three rows of the reduced row echelon form.

The advantage of the reduced row echelon form is that we can simply inspect the rows to determine the annihilators of lowest degree. For example, in the reduced row echelon form, the top row represents an annihilator of degree 3, since there is a value 1 in the column associated with $x_1x_2x_3$. Since the pivot point has the *only* value 1 in this row, the only way to form an annihilator of degree 3 is to include this row.

The other three rows each have a pivot in a column associated with a degree 2 term. And, the only way to have a degree 2 term is to involve at least one of these rows. Since there are no other rows with

a pivot point in a degree 1 or 0 term, we can conclude that there exist *no* annihilators of degree 1 or 0. Thus, the lowest degree of an annihilator of f $(= x_1x_2 \oplus x_1x_3 \oplus x_2x_3)$ is equal to 2.

Steps to Reduce the Computation Time. The matrices for which we seek a reduced row echelon form can be large. For example, each matrix has 2^n rows, of which we manipulate only those with 1's in the function. Potentially, there are also 2^n columns. However, we can reduce the columns we need to examine by a few observations. Recall that no function has an AI greater than $\lceil \frac{n}{2} \rceil$. Thus, we need consider only those columns corresponding to ANF terms where there are $\lceil \frac{n}{2} \rceil$ or fewer variables. However, it is not necessary to consider columns corresponding to terms with exactly $\lceil \frac{n}{2} \rceil$ variables. This is because if no annihilators are found for a function f (or its complement) of degree $\lceil \frac{n}{2} \rceil - 1$ or less, it must have an AI of $\lceil \frac{n}{2} \rceil$.

We can reduce the computation of the AI of a function by another observation. If a degree 1 annihilator for a function is found, there is no need to analyze its complement. Even if the complement has no annihilators of degree 1, the function itself has AI of 1. On the other hand, finding an annihilator of degree $\lceil \frac{n}{2} \rceil$ requires the analysis of its complement for annihilators of smaller degree.

3.3.5. Results and Comments

Approach. A Verilog program was written to implement the row echelon conversion process described above. It runs on an SRC-6 reconfigurable computer from SRC Computers, Inc. and uses the Xilinx Virtex2p (Virtex2 Pro) XC2VP100 FPGA with Package FF1696 and Speed Grade -5. Table 3.5 compares the average time in computing the AI of an n-variable function on this FPGA with that of a typical microprocessor. In this case, we chose the Intel®Core™2 Duo P8400 processor running at 2.26 GHz. This processor runs Windows 7 and has 4 GB of RAM. The code was compiled using Code::Blocks 10.05. The data shown is from a C program that also implements the row echelon conversion process. For ease of presentation, we compute the rate of computation, as measured by the number of functions per second.

Table 3.5. Comparison of the computation times for enumerating the AI of n-variable functions on the SRC-6 reconfigurable computer versus an Intel® Core™2 Duo P8400 microprocessor

SRC-6 Reconfigurable Comp.				
n	Clocks per function	Functions per second	# of samples	Speedup
2	46.3	2,162,162	16,000,000*	0.5
3	70.7	1,414,130	25,600,000*	1.1
4	75.5	880,558	65,536,000*	1.9
5	348.4	287,012	4,294,967,296*	4.9
6	78.0	12,823	25,000,000	0.7
Intel®Processor				
2		4,186,290	16,000,000*	
3		1,317,076	25,600,000*	
4		458,274	65,536,000*	
5		59,029	4,294,967,296*	
6		17,699	500,000,000	

* Exhaustive enumeration of all n-variable functions

Computation Times. Table 3.5 compares the computation times for AI when done on the SRC-6 reconfigurable computer and on an Intel® Core™2 Duo P8400 processor.

The second, third, and fourth columns in the upper part of Table 3.5 show the performance of the SRC-6 and the middle two columns in the lower part of this table show the performance on the Intel® Core™2 Duo P8400 processor. The last column shows the speedup of the SRC-6 over the Intel® Core™2 Duo P8400 processor.

The second column shows the average number of 100 MHz clocks needed by the SRC-6. The third column shows the average number of functions per second. The fourth column shows the number of functions. The term *samples* is used here to indicate that, for smaller n, we had to repeat the computation of the same function to achieve a sufficient number of functions between each measuring point so that

Table 3.6. Comparing the brute force method with the row echelon method on 4-variable functions

	Brute Force	Row Echelon
# Functions	65,536	65,536
Total Time (sec.)	0.807	0.050
Total Clocks	80,748,733	4,946,111
Clocks Per Function	1,232.1	75.5
Functions Per Second	81,160	1,325,000

the time of computation was accurate. In the case of $n \leq 5$, all functions were enumerated, and in the case of $n = 6$, a subset of random functions was enumerated.

The second column in the lower part of Table 3.5 shows the average number of functions per second on the Intel® Core™2 Duo P8400 processor, while the third column of this table shows the number of functions. The last column shows the speedup of the reconfigurable computer over the Intel® Core™2 Duo P8400 processor.

For example, for $n = 5$, the SRC-6 reconfigurable computer is 4.9 times faster than the Intel® processor. For $n = 4$, the SRC-6 is 1.9 times faster. However, for $n = 6$, the processor is actually faster than the reconfigurable computer. In the case of $n = 6$, a sample size of 25,000,000 was used for the SRC-6 and 500,000,000 for the Intel® Core™2 Duo P8400 processor. For all lower values of n, exhaustive enumeration was performed.

Comparing the Row Echelon Method to Brute Force. Table 3.6 compares the row echelon method, which involves the solution of simultaneous equations with the brute force method discussed earlier for the case of $n = 4$.

In both cases, 65,536 functions were considered, all 4-variable functions. The last row shows that the row echelon method is able to process 1,325,000 functions per second verses 81,160 functions per second for the brute force method, resulting in 16.3 times the throughput.

Table 3.7. The number of n-variable functions distributed according to algebraic immunity for $2 \leq n \leq 6$

AI\n	2	3	4	5
0	2	2	2	2
1	14	198	10,582	7,666,550
2	0	56	54,952	**4,089,535,624**
3	0	0	0	**197,765,120**
Total	16	256	65,536	4,294,967,296

AI\n	6
0	(2) 0
1	(1,081,682,871,734) *1,114,183,342,052*
2	***1,269,840,659,739,507,264***
3	***17,176,902,299,786,702,300***
Total	18,446,744,073,709,551,616

Bold entries are previously unknown.
Bold and italicized entries are estimates to previously unknown values.

Distribution of Algebraic Immunity to Functions. Table 3.7 shows the number of functions with various algebraic immunities for $2 \leq n \leq 6$. This extends the results of [346] to $n = 5$. In our case, the use of a reconfigurable computer allows this extension.

The entries shown in **bold** in the column for $AI = 5$ are exact values for previously unknown values. The entries shown in ***bold and italics*** for $AI = 6$ are approximate values for previously unknown values. In this case, the approximate values were determined by a Monte Carlo method in which 500,000,000 random 6-variable functions were generated (or $2.7 \times 10^{-9}\%$ of the total number of functions) and their algebraic immunity computed. For $n = 5$ and $n = 6$, the number of functions with algebraic immunity 1 are known. However, Table 3.7 shows the value 0 for the number of functions with algebraic immunity 0 (there are actually 2, the exclusive OR function and its complement). This is because the Monte Carlo method produced no functions with

Table 3.8. Frequency and resources used to realize the AI computation on the SRC-6's Xilinx Virtex2p (Virtex2 Pro) XC2VP100 FPGA

n	Freqency (MHz)	# of LUTs	Total # of slice FFs	# of occupied slices
2	103.1	2,066(2%)	2,977(3%)	2,089(4%)
3	113.0	2,199(2%)	3,011(3%)	2,157(4%)
4	109.4	2,343(2%)	2,760(3%)	2,120(4%)
5	100.9	5,037(5%)	4,110(4%)	3,780(8%)
6	87.5	8,990(10%)	3,235(3%)	5,060(11%)

an AI of 0. The *italicized* value, *1,114,183,342,052*, in Table 3.7 is an estimate of the number of 6-variable functions with algebraic immunity 1. To show the accuracy of the Monte Carlo method, compare this to the previously known exact value 1,081,682,871,734 [346]. The estimated value is 3% greater than the exact value.

Resources Used. Table 3.8 shows the frequency achieved on the SRC-6 and the number of LUTs and flip-flops needed in the realization of the AI computation for various n. The frequency ranges from 113.0 MHz at $n = 3$ to 87.5 MHz for $n = 6$. Since the SRC-6 runs at 100 MHz, the 87.5 MHz value is cause for concern. However, the system works well at this frequency. For all values of n, the number of LUTs, slice flip-flops, and occupied slices were well within FPGA limits. Indeed, among all three parameters and all values of n, the highest percentage was 11%.

Cryptographic Properties. We show that a reconfigurable computer can be programmed to efficiently compute the algebraic immunity of a logic function. Specifically, we show a 4.9 times speedup over the computation time of a conventional processor. This is encouraging given that algebraic immunity is one of the most complex cryptographic properties to compute. This is the third cryptographic property we have concentrated on that has benefited from the highly efficient, parallel nature of the reconfigurable computer. The interested reader may wish to consult two previous papers on nonlinearity [146] and correlation immunity [92].