INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 12, NO. 4, DECEMBER 2007, 325-330

A Comparison of Genetic Algorithm Techniques for the Cryptanalysis of TEA

Aaron GARRETT, John HAMILTON and Gerry DOZIER

Abstract— **Block cipher cryptanalysis is a very difficult problem for which, so far, no automated solution has been developed. Common approaches, such as linear or differential cryptanalysis, take advantage of the discovered weaknesses exhibited by a cipher. These weaknesses, however, are only discovered through painstaking, intense scrutiny by professional cryptanalysts. Most of the exploits that have been found to break modern ciphers are based on the observation that those ciphers do not produce truly random output. In this paper, we extend the work of Hernandez, et al, in which genetic algorithms are used to solve the problem of determining whether a given cipher produces random output. We show that carefully tailored genetic algorithms are capable of finding efficient distinguishers for ciphers much faster than has previously been reported.**

Index Terms— **Cryptanalysis, block cipher, distinguisher, genetic algorithms, TEA.**

1. INTRODUCTION

AN important part of any cryptanalysis technique involves determining the tendencies of a cipher to produce less than random output. For instance, it has been shown that, if proper precautions are not taken, the stream cipher RC4 is more likely to produce a 0 as its third keystream byte than random chance would allow [1]. Weaknesses like this one have been exploited to break RC4, which forms the basis of Wired Equivalent Privacy (WEP) encryption, a common wireless network encryption technique. Many tools exist than can discover the key used for WEP encryption in a matter of hours. This type of attack is common, but finding the weakest parts of the encryption has traditionally been accomplished through painstaking effort.

Genetic algorithms have shown remarkable aptitude for solving difficult optimization problems. Hernandez, et al, have shown that the problem of determining whether a given cipher produces random output can be interpreted as an optimization problem and that genetic algorithms can be successfully applied to those problems [2], [3]. In this paper, we extend an idea first described in [2], in which a genetic algorithm was used to find efficient distinguishers that are capable of determining the bits at which a cipher fails to produce random output. The basic idea involves evolving a bitmask, m, for the message-key pair, p, such that passing m & p through the cipher maximizes the chi-squared statistic.

The paper is organized as follows: Section 2 provides an introduction to genetic algorithms; Section 3 introduces Feistel networks, a major topic in modern ciphers; Section 4 discusses the Tiny Encryption Algorithm; Section 5 explains the experimental setup that was used; Section 6 provides our results; and Section 7 gives conclusions and directions for future work.

2. GENETIC ALGORITHMS

In the early 1970s, John Holland and his colleagues at the University of Michigan studied particular applications of cellular automata called Genetic Algorithms (GAs). In this early work, Holland attempted to simulate natural evolution by evolving binary strings [4]. These binary strings are called chromosomes, which represent candidate solutions for some problem of interest. In recent years, genetic algorithms have also been developed to operate on non-binary chromosomes. An individual consists of a chromosome and a fitness value, the latter a measure of how well the chromosome solves the problem.

Analogous to the evolution found in the natural world, each chromosome (or set of chromosomes) undergoes some type of modification in order to produce an offspring. These modifications are collectively referred to as genetic operators. At the beginning of a GA, a population of chromosomes is created where each member is randomly generated. This population then goes through a series of generations, or applications of the genetic operators. During each generation, a set of chromosomes from the population is selected to be the parents of the next generation. The parent selection strategy may take on many forms, such as fitness-proportional, tournament, or random selection. Once the parents have been chosen, they are exposed to one or both of the genetic operators - crossover and mutation. During crossover, the chromosomes of two parents are mixed to form one or more offspring.

Various types of crossover operators exist, including single-point, multipoint, uniform, and BLX crossover. During mutation, the chromosome of a single parent is modified to produce an offspring. This modification is typically some type of random change applied to one or more of the alleles (i.e. components) of the chromosome, such as a bit flip for a binary chromosome or the addition of a Gaussian random variable for a real-valued chromosome.

Once the genetic operators have been applied to the parents, a set of offspring is produced. The final step in the generation

Manuscript received May 31, 2007; revised September 25, 2007.

A. Garrett (agarrett@jsu.edu) is with the Department of Mathematics, Computing, and Information Sciences at Jacksonville State University, Jacksonville, AL 36265.

J. Hamilton (hamilton@auburn.edu) is the director of the Information Assurance Center, Auburn University, AL 36830.

G. Dozier (gvdozier@ncat.edu) is Professor and Chair of the Computer Science Department at North Carolina A&T State University, Greensboro, NC 27411.

is to determine which of the offspring (and which of the parents) survive to the next generation. The way this selection is accomplished is typically referred to as the replacement strategy. As with parent selection, this strategy may take on many forms, including those mentioned as selection strategies. The individuals that survive the replacement strategy make up the new population for the next generation of the evolution. For a thorough treatment of genetic algorithms, see [5].

3. Feistel Networks

The basis for nearly all modern cryptographic systems is a design known as a Feistel network. Feistel networks (or Feistel ciphers) are named after IBM cryptographer Horst Feistel, who pioneered the field of modern block ciphers, including the design of the Data Encryption Standard (DES) cipher.

Feistel networks operate on fixed-size blocks of plaintext and process those blocks for several rounds. Each round of processing typically involves several steps. First, the input block is split into right and left halves. Then, the right half is passed through the round function F, along with some key material. The output of this function is then combined with the original left half using exclusive-or (XOR). Finally, the original right half becomes the left half for the next round, while the new left half becomes the right half. Figure 1 displays these characteristics.

The choice of the round function F and the way in which key material is incorporated in each round distinguishes one Feistel cipher from another. Typically, F is designed in such a way that it accomplishes the tasks of key-mixing, substitution, and permutation. The key-mixing step ensures that the ciphertext data is dependent on the key. The substitution step ensures "confusion" between the ciphertext and the key value. The permutation step ensures a "diffusion" of the statistics of the plaintext in the ciphertext [6], [7], [8]. A Feistel cipher makes use of the intuitive notion that even simple confusion and diffusion operators, if used alternately and often, can produce a thoroughly mixed, encrypted output.

For the key-mixing component, Feistel ciphers make use of a concept called a key schedule [6], [7]. Suppose that we have a Feistel network that operates on a block of size of N bytes for R rounds. Rather than use the secret key, K, for the key-mixing step of each round, a set of subkeys, S_i $(1 \leq i \leq R)$, are generated from K.

4. The Tiny Encryption Algorithm

The Tiny Encryption Algorithm (TEA) was created by David Wheeler and Roger Needham in 1994 [9]. It was created to provide a very small, fast, encryption system whose operation could be easily committed to memory. TEA is a Feistel cipher that uses a simple inner function made up of 32-bit additions, bit-shifts, and exclusive-or, and it operates on 64-bit blocks using a 128-bit key.

The operation of one cycle of TEA is depicted in Figure 2. A cycle in TEA corresponds to two Feistel rounds. The creators specify that 32 cycles (i.e., 64 rounds) should be used. Here, we see that the rightmost 32-bit word is turned into three different copies. One is shifted to the left by 4 bits and added

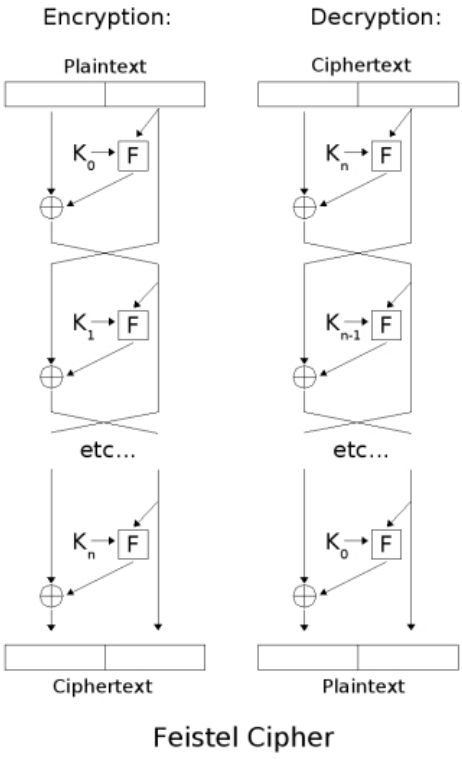

Fig. 1. Generic Feistel Network

to the first word of the key, one is shifted to the right by 5 bits and added to the second word of the key, and the last one is added to the current value of delta, which is a publicly known constant based on the golden ratio and initially set to 0x9E3779B9. (Each cycle, the value of delta is increased by its initial value so that, at every cycle, the value of delta is a multiple of the initial value.) These three values are then combined by exclusive-or and the result is added to the leftmost 32-bit word. Then, in Feistel fashion, the left and right words are swapped. In the second round of each cycle, the only change is that the third and fourth key words are used.

In the late 1990's, the original TEA algorithm was discovered to suffer from equivalent keys and to be vulnerable to related-key attacks [10]. To overcome these weaknesses, a revised version, known as XTEA (eXtended TEA), was created [11]. The operation of this modified algorithm is depicted in Figure 3. In [2], the authors used GAs to attack the original TEA algorithm, so we limit our experiments in this paper to TEA, as well.

5. Experimental Setup

Like the authors in [2], in this paper we attack reduced-round versions of TEA using a genetic algorithm. We consider attacks on up to 3 cycles (6 Feistel rounds). However, each attack on an increased number of cycles must be treated differently. As cycles are added, the cipher is better equipped to thoroughly randomize the data, which makes the distinguishing process more difficult. We follow the lead from

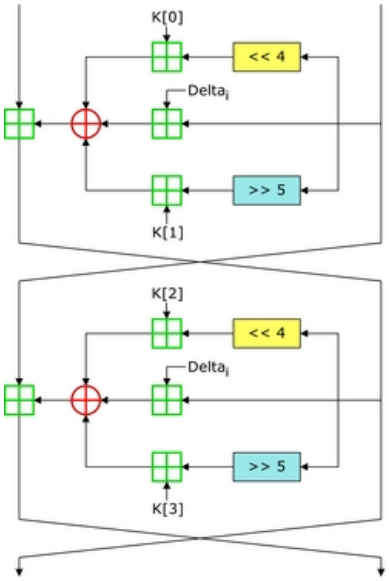

Fig. 2. One Cycle of TEA

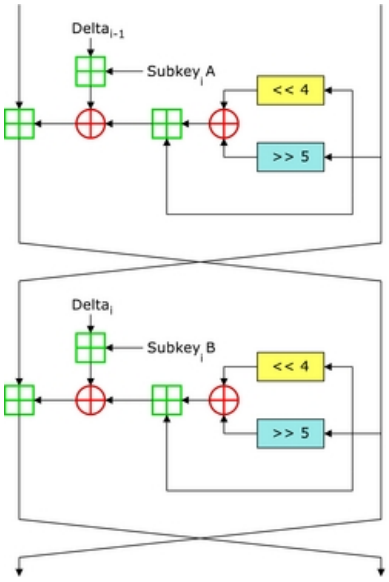

Fig. 3. One Cycle of XTEA

[2] and modify our fitness function as the number of cycles increases.

A binary generational GA was used to evolve bitmasks for random message-key pairs that were to be input for the TEA cipher. As mentioned earlier, the message block size of TEA is 64 bits, while the key size is 128 bits. Therefore, a message-key pair in our work was made up of 192 bits, which determined the length of the bitmasks to be evolved. Each message-key pair was randomly generated using a cryptographically secure pseudo-random number generator based on the SHA-1 cryptographic hash function [12]. To evaluate each candidate bitmask, m, we performed a logical AND with the bitmask and each input pair, p_i, to form m & p_i, $0 \leq i < 2048$. Each of these values was then passed through the cipher to produce a 64-bit output. As in [2], the least significant byte of the most significant word of the output (i.e., $output[0]$ & $0x0000FF$) for each pair p_i was tested using the chi-squared statistic to determine its deviation from a random distribution.

A meta-GA [13] was used to determine the specific parameters for each GA (e.g., population size, mutation rate, etc.). The specific meta-GA used to find these parameters was a real-coded, steady-state GA using blend crossover and initialized using the parameters in Table I. The chromosome to be evolved by the meta-GA consisted of four real-coded values for population size (which was rounded up to the nearest integer), crossover usage rate (for binary-coded uniform crossover), mutation usage rate, and mutation rate. The meta-GA was run for a maximum of 400 function evaluations. Each evaluation consisted of creating a binary-coded generational GA with the candidate parameters and running it 5 times for 20000 function evaluations each time. The average best fitness (as defined by the chi-squared statistic) across the 5 runs was then returned as the fitness for those parameters. Figure 4 illustrates the operation of the meta-GA.

TABLE I
PARAMETERS FOR META-GA

Parameter	Value
Population Size	50
BLX-α	0.1
Mutation Range	0.05
Crossover Usage Rate	1.0
Mutation Usage Rate	1.0
Mutation Rate	1.0

6. RESULTS

6.1. One-cycle TEA

For the case of 1-cycle TEA, [2] used a generational GA with a population size of 100, a single-point crossover rate of 0.85, and a mutation rate of 0.005 to evolve the bitmasks for the message-key pairs. The fitness function used in their work is given in Equation 1.

$$fitness = \begin{cases} w^4, & \text{if } \chi^2 = 522,480 \\ \chi^2, & \text{otherwise} \end{cases} \quad (1)$$

In this equation, w represents the weight (or number of 1's) of the bitmask. Here, the authors claim that the χ^2 statistic achieves a maximum of 522,480 for a dataset consisting of 2,048 samples and 256 possible values. Their fitness function can then be interpreted as searching for maximum deviation from a uniform probability distribution, followed by a maximization of the weight of the bitmask.

In our work, we modified the value 522,480 to be 522,240 because we believe that the authors were in error. To elaborate, the χ^2 statistic for an underlying probability distribution is calculated as follows:

$$\chi^2 = \sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i}$$

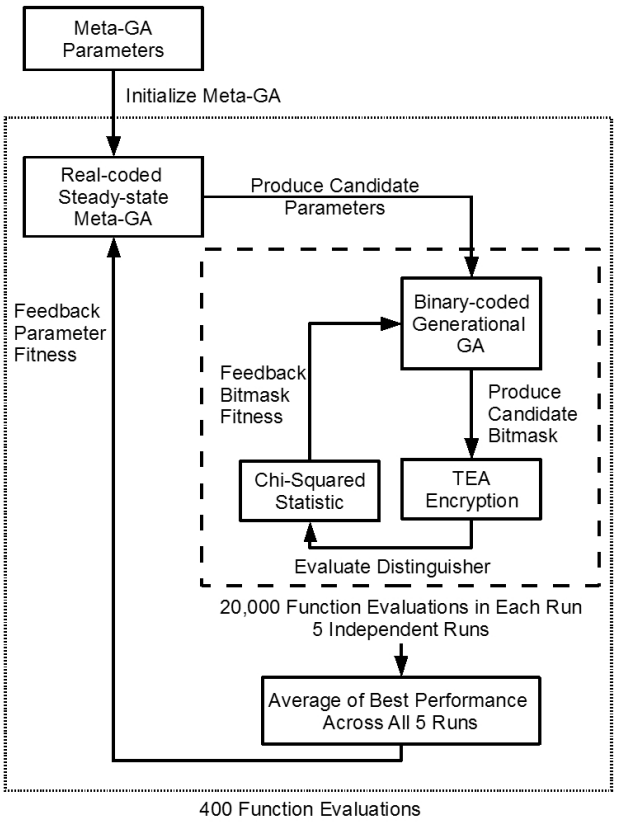

Fig. 4. Operation of Meta-GA

where N is the number of observations and O_i and E_i are the observed and expected probabilities, respectively, for observation i. In our situation, the underlying probability distribution is expected to be uniform (i.e., truly random). Since there are 256 possible values, the maximum deviation from the uniform distribution would occur when all observations take on the same value. In this case, 255 of the possible values would be observed with frequency 0, while one value would be observed with frequency 2,048. The expected frequency for each value would be $\frac{2,048}{256} = 8$. Therefore, the maximum χ^2 statistic would be

$$\begin{aligned}\chi^2 &= 255\frac{(0-8)^2}{8} + \frac{(2,048-8)^2}{8} \\ &= 2,040 + 520,200 \\ &= 522,240\end{aligned}$$

for the problem described. It is interesting to note that the authors in [2] find their best solution to have this χ^2 value. Therefore, for our work, we modify their fitness function to the one given in Equation 2.

$$fitness = \begin{cases} w^4, & \text{if } \chi^2 = 522,240 \\ \chi^2, & \text{otherwise} \end{cases} \quad (2)$$

We also used different genetic algorithm parameters, as determined by the meta-GA, for our work. We used a non-elitist generational GA with a population size of 185 along with uniform crossover (rather than single-point crossover). Our crossover usage rate was set to 1.0, our mutation usage rate was set to 0.1786, and our mutation rate was set to 0.0829.

For one-cycle TEA, we used our GA to evolve 10 different bitmasks, each to a maximum of 30,000 function evaluations (i.e., evaluations of a candidate bitmask). In each case, we found bitmasks which had maximal deviation from the random distribution (i.e., $\chi^2 = 522,240$) and weights of 154 (3 runs) or 155 (7 runs). In [2], the authors found their best solution in 90,000 function evaluations, and its weight was only 153. We found the best solution to be the following bitmask:

$$\{\text{0xFFFFFF00, 0xFFFFE000, 0xFFFFFF00,}$$
$$\text{0xFFFFFF00, 0xFFFFFFFF, 0xFFFFFFF}\}$$

6.2. *Two-cycle TEA*

For two-cycle TEA, [2] modify their fitness function in order to relax the requirement that maximal deviation is achieved in the χ^2 statistic. For this case, the fitness function is given in Equation 3.

$$fitness = \begin{cases} \frac{1}{w^3} + w^4, & \text{if } \chi^2 \geq 403.4579 \\ \frac{1}{w^3}, & \text{otherwise} \end{cases} \quad (3)$$

Here, the central idea is to force the GA to concentrate first on finding bitmasks that achieve a minimally acceptable χ^2 statistic and then to focus on increasing the weight of the bitmask.

To produce bitmasks for two-cycle TEA, we used a GA with a population size of 80, a crossover usage rate of 0.8943, a mutation usage rate of 0.2555, and a mutation rate of 0.0274. These parameters are different than those used in the one-cycle case. This is because the added cycle complicates the search for adequate solutions, and, therefore, the parameters must be changed to better correspond to the new problem.

As with the one-cycle case, we used our GA to evolve 10 different bitmasks, each to a maximum of 30,000 function evaluations. For each of these bitmasks, we calculated the average χ^2 statistic across 30 different random input-key pairs. The results are given in Table II. We found that the average bitmask weight was 157 with a standard deviation of 4.6188. Likewise, the average χ^2 statistic was 453.3875 with a standard deviation of 26.3766.

TABLE II
RESULTS OF GENETIC ALGORITHM ON TWO-CYCLE TEA

Weight	χ^2
157	459.6417
155	483.6
158	474.8167
145	468.2333
159	451.3917
158	415.8583
160	442.475
157	435.6833
162	488.3333
159	413.8417

In [2], the authors found their best solution in 70,000 function evaluations, and its weight was 155. Additionally, their criteria for the distinguisher is $\chi^2 > 390.0315$. So, if we are given an unknown, black box function that perform TEA encryption, we need only take random input-key pairs, apply any of the bitmasks found by the GA, pass the result into the black box, and calculate the χ^2 statistic on the result. If this statistic is greater than 390.0315, then we know that the black box function is indeed performing TEA encryption.

In our case, we were able to find bitmasks that, on average, had a greater weight than those found by [2]. These bitmasks were also able to achieve a large enough χ^2 statistic to be used with the distinguisher given in [2]. We found the best solution to be the following bitmask:

$$\{\text{0xFFFFF3F8, 0xFFFE758A, 0xFFFFF7FA,}$$
$$\text{0xFFFFF5FA, 0xFFFFFF8C, 0xFFFFFF9C}\}$$

This bitmask has a weight of 162 and was able to achieve an average χ^2 statistic of 488.3333 on 30 random input-key pairs.

6.3. Three-cycle TEA

For three-cycle TEA, [2] use the same fitness function as for two-cycle TEA (i.e., Equation 3). To produce bitmasks for three-cycle TEA, we used a GA with a population size of 143, a crossover usage rate of 0.9527, a mutation usage rate of 0.133, and a mutation rate of 0.0202. These parameters are different than those used in the lesser cycle cases because, as before, the problem of finding efficient distinguishers becomes more and more complex as cycles are added.

As with the lesser cycles, we used our GA to evolve 10 different bitmasks, each to a maximum of 30,000 function evaluations. For each of these bitmasks, we calculated the average χ^2 statistic across 30 different random input-key pairs. The results are given in Table III. We found that the average bitmask weight was 100.2 with a standard deviation of 8.8292. Likewise, the average χ^2 statistic was 420.8242 with a standard deviation of 21.233.

TABLE III
RESULTS OF GENETIC ALGORITHM ON THREE-CYCLE TEA

Weight	χ^2
99	427.0
100	432.675
105	413.0417
109	437.7333
104	423.025
93	445.1333
100	396.1917
105	420.7333
79	437.2667
108	375.4417

In [2], the authors found their best solution in 63,000 function evaluations, and its weight was 116. Additionally, their criteria for the distinguisher is $\chi^2 > 330.5197$. Our results show that the bitmasks that were created by our GA can easily pass the distinguisher criteria. However, the weights of the bitmasks that were generated are considerably lower than those found by Hernandez and Isasi. This is most likely due to the lower number of function evaluations that were used to evolve the bitmasks. For lower cycles, it was possible for a more carefully tailored GA to achieve better solutions than shown in [2] when given less than half of their function evaluations. But as the cycles increase, the GA needs more time to find the near-optimal solutions.

7. CONCLUSIONS AND FUTURE WORK

In [2], the authors were unable to find efficient distinguishers for greater than four-cycle TEA. Additionally, as the number of cycles increased, they had to modify the fitness function to ease the burden on the genetic algorithm. In this paper, we have shown significant improvements to their previous work in using genetic algorithms for the task of creating efficient distinguishers for the Tiny Encryption Algorithm with less than three cycles. However, our approach was unable to surpass the approach of Hernandez and Isasi for three-cycle TEA. This could most likely be remedied by allowing the GA to evolve using a larger number of function evaluations (approaching the number used by Hernandez and Isasi).

A major open question and important area for future work is the application of multiobjective optimization to this problem. Finding efficient distinguishers, as presented in this paper, is actually a multiobjective optimization problem where the two objectives are minimzation of the bitmask weight and maximization of the χ^2 statistic. We have followed the approach used in [2], where a criterion-based fitness assignment strategy is used [14]. However, using a Pareto-based fitness assignment may lead to better performance [14].

Another opportunity for improvement may be to experiment with other evolutionary computation techniques (rather than genetic algorithms), which may be more suitable for this problem. However, there are several modifications that could be made to improve the performance of the genetic algorithm. One such approach would be to use uniform crossover, rather than single-point crossover. Another modification is the use of a steady-state GA (or, at the very least, an elitist generational GA) instead of a non-elitist generational GA. These modifications will require the use of Monte Carlo simulations (because all the inputs are randomly generated), which will, in turn, take longer to evolve near-optimal solutions. (The actual number of function evaluations will not change, but the time required to complete one function evaluation will increase.) Such modifications would very likely improve the performance of the system.

REFERENCES

[1] D. R. Hankerson, D. G. Hoffman, D. A. Leonard, C. C. Lindner, K. T. Phelps, C. A. Rodger, and J. R. Wall, *Coding Theory and Cryptography: The Essentials*, 2nd ed. New York: Marcel Dekker, Inc., 2000.

[2] J. C. Hernandez and P. Isasi, "Finding efficient distinguishers for cryptographic mappings, with an application to the block cipher TEA," *Compuational Intelligence*, vol. 20, no. 3, pp. 517–525, August 2004.

[3] ——, "Finding efficient distinguishers for cryptographic mappings, with an application to the block cipher TEA," in *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*. IEEE, 2003.
[4] J. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.
[5] G. Dozier, A. Homaifar, E. Tunstel, and D. Battle, "An introduction to evolutionary computation," in *Intelligent Control Systems Using Soft Computing Methodologies*, A. Zilouchian and M. Jamshidi, Eds. CRC Press, 2001, pp. 365–380.
[6] B. Schneier, *Applied Cryptography*. New York: Wiley, 1994.
[7] N. Furguson and B. Schneier, *Practical Cryptography*. New York: Wiley, 2003.
[8] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.
[9] D. J. Wheeler and R. M. Needham, "TEA, a tiny encryption algorithm," in *Fast Software Encryption: Second International Workshop*, B. Preneel, Ed. Springer-Verlag, 1994, pp. 363–366.
[10] V. R. Andem, "A cryptanalysis of the tiny encryption algorithm," Master's thesis, The University of Alabama, 2003.
[11] R. M. Needham and D. J. Wheeler, "TEA extensions," Computer Laboratory, University of Cambridge, Tech. Rep., October 1997.
[12] NIST, "Fips publication 180-1: Secure hash standard (shs)," National Institute of Standards and Technology, Tech. Rep., April 1995.
[13] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986.
[14] E. Zitzler, M. Laumanns, and S. Bleuler, "A tutorial on evolutionary multiobjective optimization," in *Metaheuristics for Multiobjective Optimisation*, X. Gandibleux, M. Sevaux, K. Srensen, and V. T'kindt, Eds. Berlin: Springer, 2004, pp. 3–37.

Aaron Garrett Aaron Garrett is currently completing his Ph.D. in computer science at Auburn University. He received a Bachelors degree in mathematics and a Masters degree in computer science from Jacksonville State University in Jacksonville, AL. In 2001 while pursuing the Masters degree, Aaron became a NASA Summer Graduate Fellow at Ames Research Center, Mountain View, CA. Since 2002, he has served as an instructor in the computer science department at Jacksonville State University.

John Hamilton John A.Drew Hamilton, Jr. is an associate professor at Auburn University and director of the Auburn Information Assurance Center. Previously he served as the Director of the Joint Forces Program Office at the Space and Naval Warfare Systems Command, the Research Director for the Department of Electrical Engineering and Computer Science at the US Military Academy, as Chief of the Ada Joint Program Office, Chief, Officer Training Division at the Computer Science School, Fort Gordon. Dr. Hamilton has a B.A. in Journalism from Texas Tech University, an M.S. in Systems Management from the University of Southern California, an M.S. in Computer Science from Vanderbilt University and a Ph.D. in Computer Science at Texas A&M University. Dr. Hamilton is a graduate of the Naval War College with distinction. CRC Press publishes his book, Distributed Simulation, written with David A. Nash and Dr. Udo W. Pooch.

Gerry Dozier Gerry Vernon Dozier is Professor and Chair of the Computer Science Department at North Carolina A&T State University. He is currently a member of the Technical Committee on Evolutionary Computation of the IEEE Computational Intelligence Society. He is an associate editor for (1) IEEE Transactions on Evolutionary Computation, (2) Intelligent Automation and Soft Computing (AutoSoft), (3) Journal on Education and Information Technologies, and (4) the International Journal of Intelligent Computing and Cybernetics. Gerry is also the director of the Applied Computational Intelligence Lab.