Analysis and Java Implementation of Tiny Encryption Algorithm (TEA) for Computer Security

Chance J.R. Cantwell, *5th Year Engineering Systems and Computing Undergraduate Student, University of Guelph*

Abstract—**The purpose of this assignment is to implement an encryption algorithm using the Java programming language and evaluate it through testing and experimentation. The Tiny Encryption Algorithm (TEA) was chosen for its simplicity and ease of free use. By implementing the algorithm and performing basic tests, such as encrypting and decrypting values, the algorithm can be analyzed to ensure that it meets specifications from literature. Through experimentation, the execution performance of the algorithm and the difficulty to conduct an exhaustive key search can be determined and analyzed. From the obtained results, it is possible to make recommendations regarding how the algorithm may be used.**

Index Terms—**Tiny, Encryption, Algorithm, TEA, Cryptography, Java, Analysis, Implementation, Cipher, Symmetric, Encryption, Decryption, Security, Communication, Feistel**

I. Introduction

THE Tiny Encryption Algorithm (TEA) is a method used for cryptography, which is important for securing communicated data from being seen by unauthorized personnel. The TEA, found in 1994 by David Wheeler and Roger Needham of the Computer Laboratory at the Cambridge University in England, is a symmetric encryption algorithm, which means that it uses the same key for encryption and decryption processes. The TEA algorithm is a block cipher, which means that the algorithm and an encryption or decryption key work together on a block, or group of bits, of data, rather than a single bit of data. The TEA algorithm is a simple cryptographic tool that is fast, easy to use, and only requires a few lines of code for implementation.

There are some advantages of using the TEA algorithm. One very important advantage is its rapid processing speed, which makes it "ideally suited to modern applications, such as distribution of pay-as-you-go real-time broadband video over broadband internet" [1] connections. In addition, it can be used freely, as it is in the public domain with no active patents filed for it [1]. Overall, the TEA algorithm is useful for applications where speed is a priority, as well as a cost-effective tool for educational or consumer use.

II. Functionality of Tiny encryption algorithm (tea)

The Tiny Encryption Algorithm (TEA) is a simple algorithm used in cryptography. It is a Feistel type cypher in the sense that it "…uses a number of rounds to scramble the data" [1] being encrypted. Wheeler and Needham suggested in their 1994 paper that 16 cycles would be sufficient for encryption and decryption; however it is recommended to use 32 cycles [2]. The algorithm uses a 128 bit key, divided into four 32 bit blocks, and two blocks of 32 bits of data for its inputs.

During one cycle of the TEA algorithm, several steps are performed, which can be seen in the pictorial representation shown in Figure 1. The first 32 bit block of data (Z) is shifted to the left by 4 bits and is added to the first key block (k[0]). Z is added to a key constant called δ (delta), which is 0x9e3779b9 in hexadecimal and calculated using equation 1

Submitted for review with Assignment 2 for CIS*4110 Computer Security, School of Computer Science, University of Guelph, on March 10, 2017.

C.J.R. Cantwell is currently a 5th year Undergraduate Engineering Systems and Computing student at the University of Guelph, 50 Stone Rd. E., Guelph, ON., Canada, N1G 2W1. (email: ccantwel@mail.uoguelph.ca)

[2]:

$$delta = (\sqrt{5} - 1)2^{31}$$

Equation 1: Delta calculation for Tiny Encryption Algorithm [2]

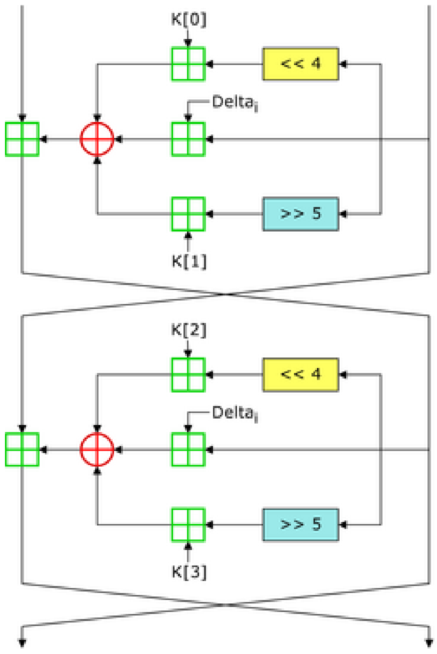

Figure 1: Pictorial representation of TEA algorithm functionality [2]

Z is then shifted to the right by 5 bits and is added to a second key block (k[1]), before an XOR operation takes place on the previous operations. The result of the XOR operation is added to the second 32 bit block of data (Y), which is obtained through a similar procedure but using the second and third key blocks (k[2] and k[3]). The first and second blocks of data trade places for the next cycle of the algorithm (what was Z before is now Y, and what was Y before is now Z).

It is important to mention that one weakness of the TEA algorithm is that "[it] has 3 'equivalent keys' for each key, reducing the effective keyspace from 128 bits to 126 bits" [1]. This means that every key value has three equivalent values that can be used just as well as the original. This weakness could make data encrypted using the TEA algorithm more vulnerable to being decrypted by authorized personnel. Also, since the TEA is a symmetric cipher, the same key that was used to encrypt communicated data needs to be sent along with the data so that it can be decrypted by the receiver. If the key somehow gets into the hands of an unauthorized individual, any data encrypted using that key is especially vulnerable, as the unauthorized access would be a breach against the data owner's privacy.

III. Implementation

The program accompanying this report was written in the Java programming language as an educational tool to learn about the functionality of the TEA algorithm. It was created using a command line interface that allows users to select from a series of five options, which are Encrypt New, Encrypt Existing, Decrypt New, Decrypt Existing, and Exit. Selecting Encrypt New or Decrypt New allows users to enter a series of two integers that they would like to encrypt or decrypt, as well as provide a series of four integers to use as encryption or decryption keys. Selecting Encrypt Existing or Decrypt Existing allows users to encrypt or decrypt integer values previously stored in the system, while still being able to provide the series for encryption or decryption keys. Finally, selecting Exit simply closes the program. Testing was performed to ensure that the program functioned according to the specifications of the algorithm. In addition, a second version of the program was created to automate the creation of decryption key values for experimentation purposes.

IV. Testing and Experimenting

For the purpose of Assignment 2 for CIS*4110 Computer Security at the University of Guelph, a testing and experimentation procedure was conducted in order to ensure that the program operated according to specifications, as well as analyze the program to determine effectiveness and performance. The testing and experimentation procedure was conducted using a Toshiba Satellite S50D-A-00G laptop computer with the specifications shown in Table 1.

Table 1: Specifications of Laptop Computer used for Testing and Experimentation

Operating System	Microsoft Windows 8.1
Processor	AMD A10-5745M APU with Radeon(tm) HD Graphics 2.10 GHz
RAM	8.00 GB (7.20 GB usable)
IDE Environment	NetBeans IDE 7.0.1
Java Development Platform	JDK 1.6.0_18
Java Runtime Platform	Java 8 Update 51 (32 and 64 bit)

The testing procedure consisted of running the program using various inputs and ensuring that the program responds correctly. For example, when a user tries to encrypt a series of two integers, and then tries to decrypt the results, the integers obtained should be the same as the values provided for encryption, provided that the same integer key values were supplied for encryption and decryption. Error handling was also tested to ensure that the program did not crash as a result of an exception being thrown, such as when a user tries to encrypt or decrypt existing values when none have been provided. Table 2 shows the tests that were performed, as well as the results obtained from each test. It is important to note that "Expected" in Table 2 means that the output is logical, and meets the specifications of the algorithm.

Table 2: Performed Tests of TEA Algorithm Implementation

Test	Provided Keys	Provided Values	Output	Expected?
Encryption	53687429 84248344 123123 123123	12345678 23456789	1023839550 1455656702	Yes
Decryption	53687429 84248344 123123 123123	1023839550 1455656702	12345678 23456789	Yes
Decryption – Wrong Keys	76421358 91074368 217465 217465	1023839550 1455656702	251359381 1064318256 -- Decryption failed!	Yes
Error Handling – Encrypt Existing	None	None	"No values to encrypt. You must specify values to encrypt!" → Let user input values	Yes
Error Handling – Decrypt Existing	None	None	"No values to decrypt. You must specify values to encrypt!" → Let user input values	Yes

The experimentation procedure used a modified version of the original program in order to analyze its effectiveness and performance. The modified program simulated an attacker trying an exhaustive key search by using a random number generation class to create values to use as decryption keys, which were then used to try and decrypt a series of two integers that were previously encrypted by the program with user provided encryption key values. The goal of this experiment was to determine the amount of attempts needed in order to successfully decrypt the provided integers, and thus determine the effectiveness of the program. The results obtained from this experiment are shown in Table 3.

Table 3: Results obtained from Exhaustive Key Search Experiment

#	Provided Encryption Keys	Provided Values	Attempts	Successful?	Program Running Time
1	53687429 84248344 123123 123123	12345678 23456789	2022355496	Yes, but only for 1 out of 2 Provided Values	15 mins, 54 sec
2	53687429 84248344 123123 123123	679342 108374	7047754953	Yes, but only for 1 out of 2 Provided Values	70 mins, 10 sec
3	53687429 84248344 123123 123123	9314 6872	7079517064	Yes, but only for 1 out of 2 Provided Values	103 mins, 14 sec

In order to analyze the performance of the TEA algorithm, an experiment was conducted in which various integers were submitted to the program. The integers were encrypted and decrypted using the same key values, and the execution time of each process was recorded. Table 4 shows the results obtained from this experiment.

Table 4: Results obtained from Execution Time Experiment

#	Encryption and Decryption Keys	Provided Values	Encryption Time	Decryption Time
1	53687429 84248344 123123 123123	12345678 23456789	9282 ns	16121 ns
2	53687429 84248344 123123 123123	679342 108374	7817 ns	17588 ns
3	53687429 84248344 123123 123123	9314 6872	8305 ns	17098 ns

V. Analysis

From the basic testing that has been performed, one can see that the TEA algorithm implementation works according to specifications. It was discovered that the created Java program worked as expected, and achieved the desired results rapidly. Error handling, consisting of error messages shown to user and permitting them to solve the problem by inputting values, was also dealt with in such a way that the program did not crash or throw exception errors. Overall, the test results prove that the program is robust and accurate in its ability to decrypt and encrypt integer values.

The first experiment performed on the implementation permitted the analysis of the effectiveness of the TEA algorithm. Obtained from the results of the three trials performed for the Exhaustive Key Search Experiment (Table C), the average amount of attempts required for integer decryption was 5383209171, and the average program running time was 63 minutes and six seconds. However, during each trial, it was discovered that only one of the values was correct on a “successful” decryption attempt. The cause of this was within the method for testing the encryption to determine if the values inputted for encryption matched the output; more specifically, the method would return true for a successful attempt if the second value matched the input but the first did not. This problem was solved by modifying the method to stop executing a loop to iterate through the values when a decrypted value did not match its corresponding input value, however, it was not possible to test this resolution due to time constraints. The obtained results show the difficulty required to conduct a brute force attack, due to a maximum of 8589934584 (4*(Maximum integer range in Java)) possible key combinations to try. Depending on computing resources, trying this many key combinations could take anywhere from several minutes to several days, especially if the decryption keys are being inputted manually into the program at each attempt.

A second experiment was conducted in order to determine the execution times of the encryption and decryption processes (Table D). These values were determined to be 8468 ns for encryption and 16936 ns for decryption on average. The results from this experiment showed high execution performance of the encryption and methods in terms of speed of execution, on the specific testing platform used. This proves the theory that the TEA algorithm is fast at its ability to encrypt an input plaintext value and decrypt an output ciphertext value.

VI. Conclusions and Recommendations

The Tiny Encryption Algorithm (TEA) is a free to use block cipher used in cryptography. It is a fast and accurate method of scrambling, or encrypting data that can be implemented easily using a few lines of code. This was proved by implementing the algorithm using the Java programming language. Through testing the accompanying program, it was ensured that the implementation works according to specifications without errors or program crashing. The experiments performed with the implementation showed that the algorithm displayed high execution performance on the testing platform, and that the algorithm is difficult to attack using an exhaustive key search attack. Due to the fact that it is a symmetric cypher, and its effective key range is only 126 bits, it is recommended that the Tiny Encryption Algorithm is not used for encrypting data that may be sent to another individual, or for encrypting highly confidential data (e.g. Social Insurance Numbers) without proper storage of the encryption key.

Appendix

Encryption Algorithm in C code [3]

```
void code(long* v, long* k)  {
unsigned long y=v[0],z=v[1], sum=0,   /* set up */
 delta=0x9e3779b9, n=32 ;             /* a key schedule constant */
while (n-->0) {                       /* basic cycle start */
  sum += delta ;
    y += (z<<4)+k[0] ^ z+sum ^ (z>>5)+k[1] ;
    z += (y<<4)+k[2] ^ y+sum ^ (y>>5)+k[3] ;   /* end cycle */
              }
v[0]=y ; v[1]=z ; }
```

Decryption Algorithm in C code [3]

```
void decode(long* v,long* k)  {
 unsigned long n=32, sum, y=v[0], z=v[1],
 delta=0x9e3779b9 ;
sum=delta<<5 ;
                       /* start cycle */
while (n-->0) {
    z-= (y<<4)+k[2] ^ y+sum ^ (y>>5)+k[3] ;
    y-= (z<<4)+k[0] ^ z+sum ^ (z>>5)+k[1] ;
   sum-=delta ;  }
                       /* end cycle */
v[0]=y ; v[1]=z ;  }
```

References

[1] S. J. Shepherd, “The Tiny Encryption Algorithm,” Cryptologia, vol. 31, no. 3, pp. 233–245, May 2007 [Online]. Available: http://www.tandfonline.com.subzero.lib.uoguelph.ca/doi/pdf/10.1080/01611190601090606? needAccess=true. [Accessed: 04-Mar-2017]

[2] “Tiny Encryption Algorithm,” Wikipedia, 08-Nov-2016. [Online]. Available: https://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm. [Accessed: 04-Mar-2017]

[3] D. J. Wheeler and R. M. Needham, “TEA, a tiny encryption algorithm,” Fast Software Encryption Lecture Notes in Computer Science, pp. 363–366, 1995 [Online]. Available: http://citeseer.ist.psu.edu/viewdoc/download;jsessionid=C08E8409ADF484095568965A1EBF3E5E?doi=10.1.1.45.281&rep=rep1&type=pdf. [Accessed: 04-Mar-2017]