Steganography and Cryptography Inspired Enhancement of Introductory Programming Courses

Yana Kortsarts
ykortsarts@mail.widener.edu
Computer Science Department
Widener University,
Chester, PA, USA

Yulia Kempner
yuliak@hit.ac.il
Computer Science Department
Holon Institute of Technology
Holon, Israel

Abstract

Steganography is the art and science of concealing communication. The goal of steganography is to hide the very existence of information exchange by embedding messages into unsuspicious digital media covers. Cryptography, or secret writing, is the study of the methods of encryption, decryption and their use in communications protocols. Steganography manipulates data to ensure the security of information, but the concept of steganography differs from cryptography. Cryptography obscures the meaning of a message, but it does not conceal the fact that there *is* a message. The goal of cryptography is to make data unreadable by a third party, whereas the goal of steganography is to hide the data from a third party. We present a way to integrate steganography and cryptology examples into introductory programming courses. This enrichment promotes active involvement in the course and provides opportunity to engage students in experimental problem solving and collaborative learning to enhance critical thinking.

Keywords: Steganography, cryptology, problem solving, active learning, engagement, introductory programming.

1. INTRODUCTION

Steganography is the art and science of concealing communication (Kessler, 2004; Provos & Honeyman, 2003). The goal of steganography is to hide the very existence of information exchange by embedding messages into unsuspicious digital media covers. Cryptography, or secret writing, is the study of the methods of encryption, decryption, and their use in communications protocols. Both techniques manipulate data to ensure the security of information, but the concept of steganography differs from cryptography. Cryptography obscures the meaning of a message, but it does not conceal the fact that there *is* a message. The goal of cryptography is to make data unreadable by a third party, whereas the goal of steganography is to hide the data from a third party. Both techniques have an ancient origin, but the modern field is relatively young. Cryptography and steganography are fundamental components of computer security. Cryptography provides mathematical foundations of computer security and it is a well-developed and highly researched field of computer science. In contrast, the interest in steganography has increased only in recent years, when it was recognized that the use of steganographic technique might become a security threat. Furthermore, the first verified use of steganography for espionage purposes was recently confirmed by FBI in the case of Russian spies (Stier, 2010), who used steganography techniques to hide sensitive information in images on the internet. This accusation by the FBI has made steganography a topic of public interest, and has caused concern regarding the number of images on the internet which could potentially hide secret messages (*Zielinska*, *Mazurczyk* & *Szczypiorski, 2014)*. Due to the crucial importance of cryptography and steganography in computer science, it seems that at least some examples should be integrated into the introductory programming courses - first core courses in the undergraduate computer science (CS) and computer information systems (CIS) curriculum. Furthermore, these concepts provide an opportunity to enhance analytical and critical thinking including creativity and ethical analysis which are fundamental characteristics of the information systems (IS) profession as stated in the latest IS 2010 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems (Topi, et al., 2010). The current paper discusses our experience integrating steganography and cryptography examples into freshman year programming courses taught on Python and C.

2. GOALS AND OBJECTIVES

Computer security is long recognized as an excellent source of the interesting projects that could be integrated into introductory programming courses. The merit of steganography-oriented assignments was discussed previously by several authors (Courtney, M. & Stix, A., 2006; Hunt, 2005; Stevenson, D., Wick, M., & Ratering, S., 2005; Markham, 2009; Ryder, 2004). Various approaches to teach cryptography courses for undergraduates were documented in several papers (Aly & Akhtart, 2004; Gandhi, Jones, & Mahoney, 2012; Hsin, 2005; Huraj, L. & Siladi, V., 2009). In addition, one of the authors of the current paper had a successful experience integrating these topics into a computer forensics course for non-majors (Kortsarts & Harver, 2007), and both authors had a successful experience integrating a public-key cryptography component into a programming course (Kortsarts & Kempner, 2010). In contrast, the focus of the current work is on the integration of cryptography and steganography concepts into freshman year introductory programming courses that are taught on Python and C without use of any image processing and graphics libraries. We present an idea of designing the course centered around these topics and emphasizing the merit of cryptography and steganography inspired programming assignments to develop and enhance programming and critical thinking skills. The related assignments are integrated into the courses not as a separate module but through the entire curriculum, starting at the very first week of classes from the non-programming computer ethics component. In this paper, we focus on the programming part of the courses and emphasize algorithmic implementations. We design secure communication teamwork to help to promote collaborative learning. Our goal is to link main programming concepts to specific steganography and cryptography technique to promote achievement of the programming proficiency. The proposed enrichment helps to achieve the following goals: (1) engaging students in real world problem solving activities; (2) increasing students' motivation and interest in programming; (3) enhancing students' programming skills. Here we are discussing some known problems drawn from the advanced cryptology and computer security textbooks, as well as less known cryptography techniques, which are not covered in major texts. We are making these problems accessible to novice programmers. Proposed experiments create an enjoyable programming experience, spark students' interest, and increase their engagement in the course. Students show a great interest in discovering and decrypting hidden messages. They become highly motivated in algorithmic implementation of various steganography and cryptography techniques. Some of the coding schemes are revisited several times during the course, and students have an opportunity to observe their growing abilities to tackle more complex problems and design more elegant implementations as course is progressing. Furthermore, the proposed enhancement provides an opportunity to build a solid background for upper level technical electives such as cryptology, computer security and computer forensics.

3. COURSE CURRICULUM SUMMARY

One of our institutions offers an undergraduate program leading to Bachelor of Science degrees in both Computer Information Systems (CIS) and Computer Science (CS). Both majors take the two-course series Introduction to Computer Science 1 (CS 1), taught in Python, and Introduction to Computer Science 2 (CS 2), taught in Python and C, in their first year. The structure of each course is three hours lecture and three hours lab, four credits. The second institution requires students to take two introductory programming courses taught on C during their freshman year. While we do have slightly different course structures, the course curriculum is very similar and allows joint implementation of the proposed enhancement.

As previously mentioned, the first week of classes is devoted to the computer ethics component, which provides an excellent opportunity to start discussing computer security topic. This component is not a subject of this paper, and was previously reported in (Kortsarts & Fischbach, 2013).

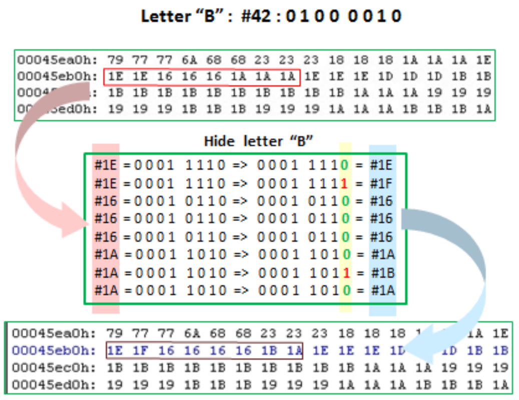
Following computer ethics, we introduce students to the binary number system. We discuss binary, octal, and hexadecimal representations, as well as ASCII code.

The rest of the curriculum is standard for the introductory programming course. Over two courses we cover material including two-dimensional lists and dictionaries in Python, and up to two-dimensional arrays in C. One of our institutions has a more extended curriculum and covers simple data structures, including linked lists and trees in C.

4. STEGANOGRAPHY ENRICHMENT

The concept of steganography is first introduced through non-programming assignments as an effective way to illustrate binary system representation and add relevance to this topic in students' eyes. We discuss the simplest steganography embedding technique – least significant bit (LSB) insertion. To avoid confusion, we provide only limited information regarding various image representations, focusing only on a definition of 24-bits RGB (true color) BMP image format, which is a sequence of binary bits, three bytes per pixel in BLUE, GREEN, RED order. Each byte gives the saturation for that color component. In our approach, which from our experience worked the best for our students, the container file is a string of binary bits and the message to hide is a string of characters. Students use ASCII code to convert a string of characters to binary string, and then replace the last bit of each byte in the container to hide the information. The reverse procedure is applied to uncover the message.

For the non-programming assignment, we ask students to hide very short messages, starting from one letter, as shown in Figure 1, and increasing the message to three letter words.



```
Letter "B" : #42 : 0 1 0 0  0 0 1 0

00045ea0h: 79 77 77 6A 68 68 23 23 23 18 18 18 1A 1A 1A 1E
00045eb0h: 1E 1E 16 16 16 1A 1A 1A 1E 1E 1E 1D 1D 1D 1B 1B
0004    : 1B 1B 1B 1B 1B 1B 1B 1B 1B 1B 1A 1A 1A 19 19 19
00  edOh: 19 19 19 1B 1B 1B 19 19 19 1A 1A 1A 1B 1B 1B 1A

Hide letter "B"
#1E = 0001 1110 => 0001 1110 = #1E
#1E = 0001 1110 => 0001 1111 = #1F
#16 = 0001 0110 => 0001 0110 = #16
#16 = 0001 0110 => 0001 0110 = #16
#16 = 0001 0110 => 0001 0110 = #16
#1A = 0001 1010 => 0001 1010 = #1A
#1A = 0001 1010 => 0001 1011 = #1B
#1A = 0001 1010 => 0001 1010 = #1A

00045ea0h: 79 77 77 6A 68 68 23 23 23 18 18 18       1A 1E
00045eb0h: 1E 1F 16 16 16 16 1B 1A 1E 1E 1E 1D    1D 1B 1B
00045ec0h: 1B 1B 1B 1B 1B 1B 1B 1B 1B 1B 1A 1A 1A 19 19 19
00045ed0h: 19 19 19 1B 1B 1B 19 19 19 1A 1A 1A 1B 1B 1B 1A
```

Figure 1: Hiding Letter B

To provide a visual support, we do utilize UltraEdit 32 to show students the binary representation of the bitmap images before and after the hidden message was inserted. We show students various ways to hide the message, starting from implementations that alter the original image. First, the text message is inserted as one block; then we separate the message into letters and insert each letter as a block replacing first byte of original image on each line. Neither of these techniques provides the proper hiding of information, and students can see strange behavior of the original image. We complete our discussion demonstrating the results of LSB embedding which works properly without image distortion, but we do discuss the limitations of this technique as well, when used in real world situations.

We revisit steganography LSB technique again after introducing one-dimensional lists, and this time students write computational implementation of this algorithm. One three-hour laboratory assignment is devoted to write a program, which hides information and recovers the hidden information, again, omitting all details of image representation. Students use UNIX redirection to input/output from/into file, but some years we do introduce file input/output in Python. To combine steganography and cryptography concepts under one umbrella, one of the last assignments is devoted to programming implementation of hiding encrypted message and decrypting recovered message. Students design a menu, which allows choosing from various cryptographic schemes to encrypt the message, which will be embedded into the image container.

5. CRYTOGRAPHY ENRICHMENT

We begin our cryptography journey discussing the process of secure communication scenario between two parties to introduce main cryptography terms. Secure communication between two entities starts with agreement on specific cryptography scheme or cipher, which is an algorithm for performing encryption and decryption. An encryption algorithm modifies the original message, plaintext, in a way that only designated receiver is able to read. The output of the encryption process is called encrypted message or ciphertext. When designated receiver would like to read a message, the ciphertext will be deciphered or decrypted using decryption algorithm. We also introduce students to the process of cryptanalysis, which is used to breach cryptographic security systems and gain access to the contents of encrypted messages without permission. There are additional concepts, such as encryption and decryption keys, which are more easily understood while introduced in the context of the specific cryptographic scheme.

Simple substitution ciphers

The first ciphers introduced to students belong to the group of simple substitution ciphers. The main idea behind these ciphers is to substitute one letter by another using a special substitution alphabet rule. We focus on two well-known ciphers belonging to this group: 1) Caesar cipher, in which alphabet is shifted forward three letters for encryption, and three letters backwards for decryption, for example the plaintext *dog* produces the ciphertext *GRJ*; 2) Shift cipher, a general form of Caesar, where the alphabet is shifted K letters forward/backwards, and K is a cipher key (Barr, 2001). For K = 3, we obtain the Caesar cipher. To encrypt, the plaintext letter P is modified using the following formula, C = (P + K) mod 26, and ciphertext letter C is computed. To decrypt, the similar procedure is applied and the desired plaintext letter is computed by the following formula, P = (C - K) mod 26. For instance, for key K = 14 and plaintext *dog*, the ciphertext is RCU. Assignments related to encryption and decryption of a single letter using Caesar and Shift ciphers implementations could be accomplished almost immediately, since they require only limited programming material. We revisit both ciphers after introduction of the loops. At this point students are capable to process a string of characters and output the results of encryption/decryption after each letter, which is still not a complete implementation. The full completion of the computational implementation of both simple substitution ciphers is done after introducing students to one-dimensional lists/arrays and explaining file input/output using programming language or input/output UNIX redirection operators. At this stage, in addition to implementing encryption/decryption algorithms, students are also introduced to the notion of cryptanalysis, specifically, brute force attack and proposed to write a computational implementation of this attack for shift cipher. The brute force approach in this case requires application of all possible shift keys, from 0 to 25, on the ciphertext to find an actual encryption key and the desired plaintext. We ask students to design a simplified interactive implementation, without utilization of the built-in dictionaries. The program finds and displays the decrypted message for each possible key. Students implement sentinel-controlled repetition based on the validity of the displayed message. The program terminates when the valid English sentence is revealed on the screen, which is a desired plaintext. To complete the process, the program also outputs the actual shift key.

Our goal is to apply these assignments to develop and practice programming skills. We explain this material on specific examples, omitting theoretical details and providing final formulas as known fact without mathematical proofs. Implementation of simple substitution ciphers provides an excellent opportunity to practice decision and loop programming structures, and simple processing of the one-dimensional lists/arrays. Since all calculations are performed on numeric values assigned to characters using the rule a/A ← 0, b/B ← 1,…, z/Z ← 25, these examples require working with various data types - characters and integers - switching among them while moving from input to calculations and then to output, which is a struggle for novice programmers. Python provides more flexibility than C, but still requires explicit conversion to avoid any logic errors. While these ciphers are well suited for starting, one should note that they only require modifying the content of the array/list values, leaving the structure of the array/list unchanged. The next example requires array/list manipulations of a higher complexity.

Dynamic substitution cipher – Chaocipher
Recently, we also integrated into the course curriculum, a less known and more complex cipher, chaocipher, (Byrne, 1918; Rubin, 2010), belonging to the group of dynamic substitution ciphers. In these ciphers, the substitution rule is changed after each character is encrypted. To decrypt, the reverse procedure is applied. The chaocipher was originally invented by John F. Byrne in 1918, who claimed that the cipher is unbreakable. Unfortunately, the cipher didn't receive any recognition from US officials. Frustrated by the lack of interest, Byrne published four plaintext-ciphertext challenges in his autobiography, *Silent Years* in 1953 (Byrne, 1953). The cipher details were kept secret for many years. Things changed in 2010 when the National Cryptologic Museum library received archives from the members of the Byrne family with the explanation of the chaocipher algorithm, and there are direct links to many items of interest donated by Byrne family posted on the museum website (http://www.nsa.gov/about/cryptologic_heritage /museum/index.shtml). In our approach we closely follow the description of the algorithm published in July 2010 by Moshe Rubin (2010), providing further adaptation and clarifications for novice programmers. The chaocipher method uses two alphabets that are connected to each other. The encryption/decryption algorithm essentially consists of three parts, encryption/decryption of the letter, permutation of the left alphabet using specific rules, and permutation of the right alphabet using specific rules. These steps are performed continuously until the input (plaintext or ciphertext) is exhausted. This cipher requires swapping array/list elements, shifting blocks of the elements several positions left and right, and shifting all elements cyclically until certain conditions are satisfied. These operations are more complex compared to the processing done for the simple substitution ciphers, and require a higher level of algorithmic thinking. To ease the transition and increase the difficulty level gradually, we first permit students to use additional array/list storage, increasing the space complexity of the algorithm. As a complete implementation, students are required to implement all these array/list manipulations with minimal additional space usage. To avoid any attempts at plagiarism, we provide only cipher description and all necessary details to design a computational implementation. We emphasize the mystery around this cipher to keep students motivated and excited. We reveal the name and history of the cipher only after students complete writing the program, but before the collaborative testing step of the assignment. The mystery around this cipher and the interesting history attract students' attention. This cipher provides an opportunity to practice complex manipulations of one-dimensional arrays and lists data structures, utilizing a wide range of built-in Python lists methods and functions, and writing custom functions in C. From the best of our knowledge, this cipher is not covered in any cryptography textbooks.

Block ciphers, Hill cipher
While there are plenty of ciphers with witch to practice one-dimensional lists/arrays data structures, the options are limited when it comes to two-dimensional lists/arrays. The assignment based on Hill cipher provides an efficient way to integrate programming and cryptography topics. This cipher belongs to the group of block ciphers, in which the encryption and decryption process is applied to a block of characters rather than to single character. Hill cipher was invented by Lester S. Hill in 1929 (Hill, 1929; Barr, 2001). The key for this cipher is a square matrix of integers of size n, satisfying several special conditions: 1) all elements of the matrix are numbers between 0 and 25, since the size of the English alphabet is 26; 2) the determinant of the key matrix must be relatively prime to 26. For the encryption process, the plaintext is divided into a block of n letters and for each block of n letters; multiplication of the key matrix by vector is applied to obtain the block of n ciphertext letters. The process is repeated for all blocks. To decrypt, the same multiplication procedure is applied to the block of n ciphertext letters, but instead, substituting the original key matrix with its modular inverse.

We start this programming assignment with the matrix key of size 2, processing the blocks of size 2, gradually increasing the size of the matrix and the correspondent size of the blocks. For instance, we would like to encrypt the following word, ***code***, using Hill cipher and the matrix key

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 5 & 6 \end{pmatrix}$$

To encrypt, the plaintext ***code*** is divided into two blocks.

$$\begin{pmatrix} c \\ o \end{pmatrix} \begin{pmatrix} d \\ e \end{pmatrix}$$

After replacing the characters by their numeric values using the rule mentioned earlier, c ← 2, o ← 14, d ← 3, e ← 4 , each block is encrypted in the following manner:

$$A\begin{pmatrix}2\\14\end{pmatrix}=\begin{pmatrix}2&3\\5&6\end{pmatrix}\begin{pmatrix}2\\14\end{pmatrix}(\bmod 26)=\begin{pmatrix}20\\16\end{pmatrix}=\begin{pmatrix}U\\Q\end{pmatrix}$$

$$A\begin{pmatrix}3\\4\end{pmatrix}=\begin{pmatrix}2&3\\5&6\end{pmatrix}\begin{pmatrix}3\\4\end{pmatrix}(\bmod 26)=\begin{pmatrix}18\\13\end{pmatrix}=\begin{pmatrix}S\\N\end{pmatrix}$$

The resulting ciphertext is UQSN. To decrypt, the similar process is applied on the ciphertext, substituting key matrix A with its modular inverse. The general formula for inverse matrix 2x2 reads as follows:

$$A^{-1}=(\det(A))^{-1}\begin{pmatrix}a_{22}&-a_{12}\\-a_{21}&a_{11}\end{pmatrix}(\bmod 26)$$

In our case, we perform the following calculations, to obtain the modular inverse of the key matrix:

$$A^{-1}=(\det(A))^{-1}\begin{pmatrix}6&-3\\-5&2\end{pmatrix}(\bmod 26)$$

$$A^{-1}=(23)^{-1}\begin{pmatrix}6&-3\\-5&2\end{pmatrix}(\bmod 26)$$

$$A^{-1}=17\begin{pmatrix}6&-3\\-5&2\end{pmatrix}(\bmod 26)$$

$$A^{-1}=\begin{pmatrix}24&1\\19&8\end{pmatrix}$$

$$AA^{-1}=I=\begin{pmatrix}1&0\\0&1\end{pmatrix}(\bmod 26)$$

To decrypt and find a desired plaintext, the ciphertext is divided into blocks of two letters and each block is multiplied by the modular inverse of matrix key A.

$$\begin{pmatrix}24&1\\19&8\end{pmatrix}\begin{pmatrix}20\\16\end{pmatrix}(\bmod 26)=\begin{pmatrix}2\\14\end{pmatrix}=\begin{pmatrix}c\\o\end{pmatrix}$$

$$\begin{pmatrix}24&1\\19&8\end{pmatrix}\begin{pmatrix}18\\13\end{pmatrix}(\bmod 26)=\begin{pmatrix}3\\4\end{pmatrix}=\begin{pmatrix}d\\e\end{pmatrix}$$

Note, that all calculations are performed modulo 26. As in previous examples, we provide students with all of the necessary mathematical background. For this assignment, there is a substantial increase in complexity of mathematics, and consequently the level of the programming required, while progressing from smallest matrix size to the higher sizes. The maximal size of the matrix key and the block of letters in our lab assignment is four.

Two dimensional lists/arrays is not an easy topic to comprehend. To ease computational implementation, students are provided with a detailed top-down design, and structured guidance for each function. To implement the Hill cipher encryption and decryption algorithms, students compute the matrix determinant, check validity of the matrix key to ensure that the matrix is invertible modulo 26, and then compute the key matrix modular inverse, which differs slightly from a regular matrix inverse, with multiplicative inverse of determinant modulo 26 substituting for a regular inverse of the determinant. In addition, students implement the matrix by vector multiplication. After all preparation steps are complete, students are ready to process the plaintext/ciphertext in blocks of 2, 3, or 4 letters, based on the size of the key matrix, to produce a final result.

Hill cipher provides an excellent opportunity for students to become proficient in basic processing of two-dimensional lists/arrays data structures.

6. TEAM COLLABORATION

We incorporate two team work routines. For some assignments, students work in teams for the entire laboratory session. We pair weaker students with stronger students to promote active learning. Students work together on programming implementation as well as testing and submit a joint lab report. While this approach has certain advantages, such as exchange of knowledge and the possibility to improve for weaker students and to further improve through teaching for stronger students, we found that in order to be able to perform on the required level, students must work independently during most laboratory sessions. Since the level of prior programming experience varies substantially from student to student, in recent years we avoid team work during the programming step of the assignment to make sure students are not taking advantages of their peers. We mostly apply the second collaborative learning approach in which students work in teams only to test their programs and to write a lab report. The testing process begins with secure communication session. Students exchange encrypted messages between team members and then decrypt messages using their own program. Successful decryption indicates a first step toward fully accomplished assignment. We found that the collaborative work during the testing and revision step of the assignment enhances students' understanding and creates an engaging environment. We ask students to submit their program twice, before, as well as after the testing and revision step. This allows proper grading and ability to track students' error corrections in order to gather information about the most common mistakes and address these issues before the next assignment.

7. SUMMARY

This paper presents our experience teaching an introductory programming course sequence using a computer security theme. Students practiced the main programming concepts on assignments inspired by steganography and cryptography. To assess the students' experience, we applied an indirect assessment tool and designed a short post-survey that included several open-ended questions eliciting and asked student feedback. Students commented on the level of their engagement, interest, curiosity, and active learning opportunities during the laboratory assignments related to computer security topics. We also asked students to comment on the effectiveness of these assignments to enhance programming skills compared to the various assignments related to other topics we to during the course. Overall, students provided positive feedback, especially emphasizing the impact of the team collaboration during the testing step. Students commented that the requirement to find logical errors in their peers' programs significantly promoted comprehension of the main programming concepts. They also commented on their excitement of finding proper testing inputs to discover tricky logical bugs. Based on the students' post-survey results, informal discussions, and comments from teaching evaluations, we could state that the proposed enhancement of the introductory programming courses was a successful addition to our previous positive experience with enrichment of the freshman programming course (Kortsarts & Kempner 2012). Current enrichment expands the pool of interesting and engaging assignments for this course, and we are planning to continue to work in this direction in the future. Some of the ciphers described above allow variations and modifications, and taking in account the mathematical background of the students, additional examples, such as Affine cipher and polyalphabetic substitution Vigenere (Barr, 2001) cipher could be a great addition to the already proposed set of ciphers. In addition, we propose to combine several ciphers and encrypt messages in a two-step process and then to apply a simplified cryptanalysis approach to decipher the message. We believe in changing course laboratory assignments often, and computer security based assignments provide further opportunities for successful course implementations.

8. FURTHER OPPORTUNITIES

We built our current project upon successful implementations of the single components over several years. Merkle-Hellman knapsack cryptosystem (Merkle & Hellman, 1978) was the first algorithm we introduced in the introductory programming courses sequence. We introduced additive knapsack, expanded to multiplicative knapsack, and finally discussed various cryptanalysis techniques. Programming assignments focused on encryption and decryption computational implementations, and on a dynamic programming algorithm to accomplish cryptanalysis attack. The detailed description of this project component was published in 2010 (Kortsarts & Kempner, 2010). In recent years we found that it is more efficient to cover this material in sophomore algorithms course, and focus on symmetric key cryptography schemes described above in the freshman introductory programming courses. In sophomore algorithms course students are better prepared to comprehend conceptually more difficult group of ciphers such as public key or asymmetric ciphers. Some examples of the ciphers that work well are RSA (Rivest, Shamir, & Adleman, 1978) and flipping coins over the phone, which uses a similar protocol, introduced by Blum in 1983 (Blum, 1983; Trappe & Washington, 2006), and it is based on the Rabin cryptosystem (1979). While students are capable of completing computational implementation of these ciphers and games in the freshman programming courses, conceptually, these algorithms require a higher level of maturity and appropriate mathematical background for successful integration into the course curriculum. By the end of the sophomore year, most students complete a discrete mathematics two-course sequence, ensuring their abilities to comprehend these less intuitive cryptography schemes. While these ciphers provide fewer benefits to accomplish our goals in freshman programming courses compared to symmetric ciphers described above, they are an excellent enhancement for the sophomore courses.

9. REFERENCES

Aly, A. & Akhtar, S. (2004), Cryptography and security protocols course for undergraduate IT students, *Newsletter, ACM SIGCSE Bulletin*, 36(2), 44-47

Barr, T. (2001), Invitation to Cryptology, Pearson

Blum, M. (1983), Coin flipping by telephone a protocol for solving impossible problems, *Newsletter, ACM SIGACT News – A special Issue on cryptography*, 15(1), 23, - 27

Byrne, J. (1953), Silent Years, An Autobiography with Memoirs of James Joyce and Our Ireland. New York: Farrar, Straus, and Young (Reprinted in 1975 by Octagon Books, a division of Farrar, Straus, and Giroux)

Courtney, M. & Stix, A. (2006), Building a Steganography Program Including How to Load, Process, and Save JPEG and PNG Files in Java, *Mathematics and Computer Education*, 40(1), 19-35

Gandhi, R., Jones, C., & Mahoney, W. (2012), A freshman level course on information assurance: can it be done? Here is how. ACM Inroads, 3(3), 50-61

Hsin, W. (2005), Teaching cryptography to undergraduate students in small liberal art schools, **InfoSecCD '05:** *Proceedings of the 2nd annual conference on Information security curriculum development*, 38-42

Hunt, K. (2005), A Java framework for experimentation with steganography, *SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on Computer science education*, 282-286

Huraj, L. & Siladi, V. (2009), Cryptography as aparadigm proposal for building the computer science knowledge, ICCOMP'09: Proceedings of the WSEAES 13th international conference on Computers, 357-361

Kessler, G. (2004), Overview of Steganography for the computer forensics examiner, Forensics Science Communication, 6(3)

Kortsarts, Y & Harver. W., (2007), Introduction to Computer Forensics for Non-Majors, The Proceedings of ISECON 2007: #3142, ISSN: 1542-7382

Kortsarts, Y. & Kempner, Y., (2010), Merkle-Hellman Knapsack Cryptosystem in Undergraduate Computer Science Curriculum, Proceedings of the 2010 International Conference on Frontiers in Education: Computer Science & Computer Engineering, FECS 2010, CSREA Press 2010, ISBN 1-60132-143-0

Kortsarts, Y. & Kempner, Y., (2012), Enriching Introductory Programming Courses with Non-Intuitive Probability Experiments Component, ITiCSE '12: Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, 128-131

Kortsarts, Y. & Fischbach, J., (2013), Incorporating Professional Ethics into an Introductory Computer Science Course, Journal for Computing Science in Colleges, 29(3), 35-42

Markham, S, (2009), Expanding security awareness in introductory computer science courses, InfoSecCD '09: 2009 Information Security Curriculum Development Conference, 27 - 31

Merkle, R. & Hellman, M. (1978). Hiding information and signatures in trapdoor knapsacks. Information Theory, IEEE Transactions on Information Theory, 24 (5), 525–530

National Cryptologic Museum website, http://www.nsa.gov/about/cryptologic_heritage/museum/index.shtml

Provos, N. & Honeyman, P. (2003). Hide and seek: An Introduction to Steganography, IEEE Security and Privacy Magazine

Rivest, R., Shamir, A., & Adleman, L. (1978), A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 21 (2), 120–126.

Rabin, M. (1979), Digitalized Signatures and Public-Key Functions as Intractable as Factorization, Technical Report, MIT Laboratory for Computer Science

Rubin, M. (2010), Chaocipher revealed: the algorithm, Progress report #17 from http://www.mountainvistasoft.com/chaocipher/

Ryder, J (2004), Steganography may increase learning everywhere. Journal of Computing Sciences in Colleges, 19(5), 154-162

Stevenson, D., Wick, M., & Ratering, S. (2005), Steganography and cartography: interesting assignments that reinforce machine representation, bit manipulation, and discrete structure concepts, SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on Computer science education, 277-281

Stier, C. (2010), Russian spy ring hid secret messages on the Web, from http://www.newscientist.com/ article/dn19126-russian-spy-ring-hid-secret-messageson-the-web.html.

Topi, H., Valacich, J., Wright, R., Kaiser, K. Nunamaker, J., Sipior, J., & Vreede, G. (2010), IS 2010 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems, Communications of the Association of Information Systems, 26.

Trappe, W. & Washington, L. (2006), Introduction to cryptography with Coding Theory, Second Edition, Pearson Prentice Hall

Zielinska, E., Mazurczyk, W., & Szczypiorski, K., 2014). Trends in Steganography, Communications of the ACM, 57(3), 86-9