ALGORITHMS AND SOFTWARE TOOLS FOR TEACHING MATHEMATICAL FUNDAMENTALS OF COMPUTER SECURITY

Vladimir V. Riabov and Bryan J. Higgs
Rivier College
420 S. Main Street • Nashua, NH 03060-5086 • USA
E-mails: vriabov@rivier.edu and bryanhiggs@gmail.com

Java applet-based tools have been developed for exploring mathematical foundations of computer security techniques including modular arithmetic, primes, permutations, combinations, probability, authentication algorithms, and hashes. Tools were used by students to examine MonoAlphabetic and shift substitution ciphertexts, Playfair and Vigenère ciphers, message digests, digital signatures, and public key cryptosystems.

1. Introduction

Many computer security topics involve some familiarity with Math concepts that are not often taught, or inadequately covered, in curricula, including sets, permutations, combinations, and probability; number theory (divisibility, primes, groups, rings, and fields); modular arithmetic; and computability theory (the reasonableness of an algorithm).

These topics form the basis for many security areas, but most students will have limited exposure to them. The challenge here is how to introduce these topics to a typically Math-phobic audience, without eliciting a "deer in the headlights" response. Since this is not a pure Math course, we make no attempt to be mathematically rigorous, nor to expect students to come away with advanced skills in these areas; rather we try to motivate coverage based on simple, real-world applications of these topics.

In this paper, authors share their experience in searching for new approaches in teaching computer security techniques [1] through exploring mathematical foundations of encryption algorithms that include modular arithmetic, primes, permutations, combinations, probability, authentication procedures, and hashes [2]. The developed Java Applets tools were successfully used by students to examine MonoAlphabetic, shift substitution, Playfair and Vigenère ciphertexts, as well as to develop projects [1, 3, 4] on message digests, digital signatures, and public key cryptosystems.

2. Lecture Notes

The class lectures on computer security technologies cover history of cryptography; security concepts; theory of sets, permutations, combinations, and probability; number theory and modular arithmetic; classical cryptosystems; symmetric block ciphers; public key cryptography; an overview of message authentication codes, hashes, and message digests; principles of authentication; network basics; Web security and privacy for users; tunneling and virtual private networks (VPNs); and malware. The instructors discuss with students secure ways of sharing the network resources, issues of confidentiality, medical and personal information security on the Internet, and protection from electronic spam.

This overview helps in introducing complex encryption algorithms such as the RSA Public-Key encryption algorithm [5]. At the same time, it illustrates a strong bond between mathematics and computer science. A student (even if he/she is not familiar with the theory of numbers) can try to solve the problems by a simple experimentation with the Java Applets tools especially designed for these courses.

3. Security Tools

A set of security tools (shown in Fig. 1 [left]) has been developed for these courses by using Java applets. The students have used these tools for deciphering simple shift-substitution ciphertexts (see Fig. 1 [right]), MonoAlphabetic substitution ciphers, Playfair and Vigenère ciphers, as well as for exploring modular arithmetic and message digests. The tools were also used in reviewing the topics on probabilities and combinatorics.

Figure 1. Java Applets security tools (*left*) and Shift Substitution Cipher Breaker (*right*).

4. Course Assignments

The assignments for the Computer Security course include three homework questionnaires, lab, midterm and final exams, and a project paper that covers in depth one of the computer system security technologies. A student can gain extra points towards the final grade for project presentations, lab demonstrations of computer security modeling, and submission of a paper (on the course-related topics) to conferences or journals [1, 3, 4].

Every class starts with a brief discussion of a topic that is related to the homework exercises. After this "warm-up" introduction [6], the instructor offers a discussion on the main topic and asks students for a feedback on lecture materials and their arguments on selecting a competitive strategy for the problem analysis and development. These discussions help students focus on the main point of the class session and stay active in class. Here are a few examples of the homework assignments:

4.1 Assignment 1: Cracking a Simple Cipher

Students are asked to solve simple ciphers by using any method, e.g., the following ciphertexts from the textbook [2] used in this course:

1. ***Si spy net work, big fedjaw iog link kyxogy***
2. ***Cf lqr'xs xsnyctm n eqxxqgsy iqul qf wdcp eqqh, erl lqrx qgt iqul!***

These ciphers are simple substitution ciphers of the type with which many people like to amuse themselves trying to solve – e.g., newspapers sometimes publish a daily crypto-puzzle, which readers try to solve, often during their commutes. Students typically find the second cipher easier to solve, probably because there are more 'hints' in the text, and more repetition. Also, if students solve the first cipher and then move onto the second, their mindset is likely already set in such a way that the second seems easier to solve. It is important that the students be required to describe the steps they used to 'crack' each cipher, rather than just provide the solution without explanation Evaluating their efforts based on the description of their strategy is better than simply relying on their answers.

4.2 Assignment 2: Cracking Classic Ciphers

After cracking simple short ciphers, students are asked to explore how cryptographers might actually crack some classic ciphers. The students are encouraged to use various components of Java applets while working on this assignment. They start by exploring a MonoAlphabetic Substitution Cipher (see Figs. 1, 2) that maps individual plaintext letters to individual ciphertext letters, on a 1-to-1 unique basis. (The oldest such cipher known is the Caesar cipher [2], where the mapping involves a simple circular shift within the alphabet). To encipher a message, students take each letter in the plaintext, find that letter in the Plaintext row, and substitute the corresponding letter immediately below it, in the Ciphertext row. For example, using this substitution table, we can take the message:

The enemy plans to attack on Tuesday morning

and encipher it into the following text by selecting the value of the *Shift Key* = 3:

Qeb bkbjv mixkp ql xqqxzh lk Qrbpaxv jlokfkd

To decipher the text, they simply reverse the process – or equivalently, use the negative of the original shift value, e.g., *Shift Key* = 26 – 3 = 23 (see Fig. 1, right). Both encryption and decryption can be done manually, or by using one of the Java Tools, available [7, 8].

Finally, students examine the Letter Frequency Analysis approach [1, 2, 9], which is based on some assumptions about the plaintext:

- That the plaintext consists of characters, not some kind of binary code.
- That it is written in some known natural language (e.g., English).
- That we know the frequency of letters in a typical piece of text in that language.
- That the plaintext is typical of normal English text, and so we expect the same frequencies of letters (approximately, within statistical fluctuations).

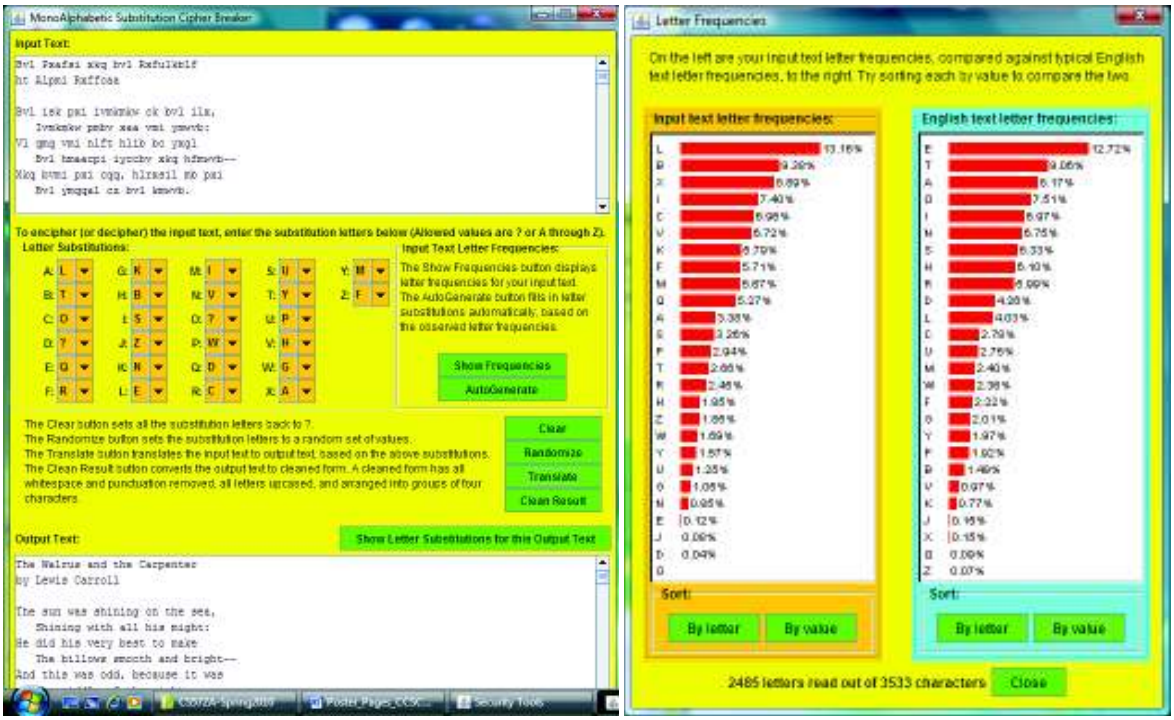

Figure 2. MonoAlphabetic Cipher Breaker (*left*) and letter frequencies in typical English (*right*).

As long as we know that there is a 1-to-1, unique mapping from plaintext to ciphertext (and, therefore, from ciphertext to plaintext), we can employ our knowledge of those letter frequencies to crack a substitution cipher. It is important to note that we need a large enough piece of text to give us some expectation that we have a large statistical sample. The longer the message, the better statistical sample we are likely to have.

Known letter frequencies in typical English text may be found on the web [9]. A typical representation of the letter frequencies in traditional English is shown on the bar chart (see Fig. 2, right). The Java tool allows a student to view the letter frequencies for the ciphertext being examined (Fig. 2, center). Students may display letter frequencies in alphabetic order, or in order by frequency.

The MonoAlphabetic Cipher Breaker Java applet (see Fig. 2, left) was used for deciphering the structured ciphertext (620 words, 2,485 characters), where the original word spacing, punctuation, and style have been retained. The second ciphertext (25,955 words; 103,818 characters) was organized in groups of four letters and word spacing and punctuation have been removed. The absence of the content clues (word spacing and punctuation) makes it more difficult to decipher the ciphertext, while the larger sample

allows greater use of letter frequency analysis. To reduce the time of deciphering the unstructured ciphertext, one student even wrote customized UNIX scripts and a standard UNIX dictionary to help with the mechanics of the solution [1].

4.3 Assignment 3: Exploring the Playfair Cipher

In 1854, Sir Charles Wheatstone invented the *Playfair Cipher* [2], which is a polygram substitution cipher using a block size of 2. Based on the use of a 5 × 5 square matrix of letters, constructed starting from a keyword or keyphrase. Each unique letter from the phrase is inserted into the square, until there are no more letters, and then the remaining letters of the alphabet are added to fill the square. For example, the phrase "Cynicism is the last refuge of the romantic" produces the matrix shown in Fig. 3 below.

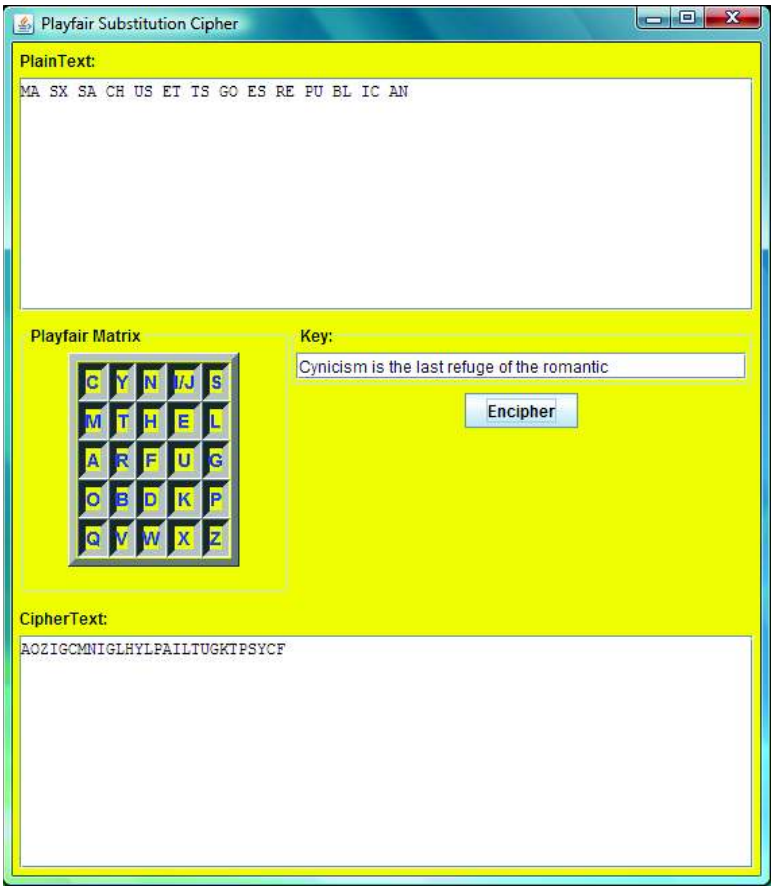

Figure 3. Playfair Substitution Cipher Applet.

Here are the rules to encipher a piece of plaintext:

Massachusetts goes Republican!

First, eliminate all non-letter characters and up-case all letters:

MASSACHUSETTSGOESREPUBLICAN

Then, arrange the plaintext in pairs of letters. If any pair of letters contains the same letter (for example, 'SS'), then insert an 'X':

MA SX SA CH US ET TS GO ES RE PU BL IC AN

If there is a last character not paired, add an 'X'.

For each pair of plaintext characters, call the first *p*, and the second *q*; the corresponding ciphertext characters *c* and *d*:

- If *p* and *q* are in the *same row* of the matrix, *c* is the letter *to the right of p*, and *d* is the letter *to the right of q*, wrapping around if necessary.
- If *p* and *q* are in the *same column* of the matrix, *c* is the letter *below p*, and *d* is the letter *below q*, wrapping around if necessary.
- If *p* and *q* share neither the same row nor column, they define the corners of a square. The other two corners of the square are *c* and *d*, with *c* being the letter in the same column as *p*.

Finally, the ciphertext will be generated (see Fig. 3):

AO ZI GC MN IG LH YL PA IL TU GK TP SY CF

Students are also encouraged to find an effective method to decipher this message.

4.4 Assignment 4: Exploring Probabilities

Starting with the warm-up exercise on simulating a coin toss, students explore factorials, powers, permutations, and combinations by using the Java Applets tool (see Fig. 4).

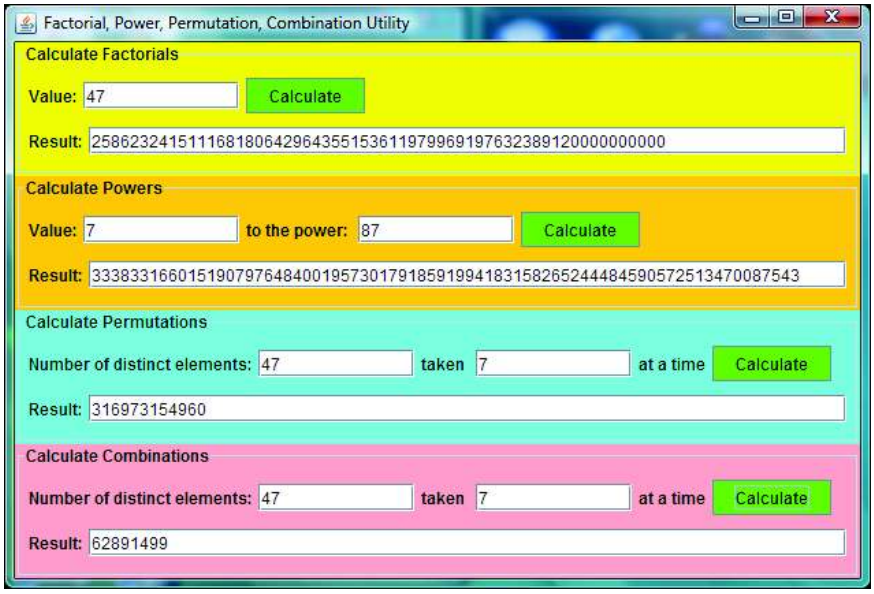

Figure 4. Factorial, Power, Permutation, and Combination Utility of Java Applets.

These exercises help students work on the fourth assignment of reviewing the theory of probabilities that plays an important part in many areas of security. In an attempt to overcome the common "Math-phobia" of students, some standard statistical/probability problems were re-cast using scenarios that were more 'security-related', and perhaps more in keeping with current events: 1) "CIA Hiring"; 2) "Brobdingnag Battles"; 3) "Delta Force"; and 4) "Ethnic Dispute".

5. Exploring Prime Numbers and Modular Arithmetic

Figure 5. The Sieve of Eratosthenes Java Applet for searching prime numbers.

Modern encryption algorithms are based on applications of modular arithmetic [10] and prime numbers. To explore different topics of the number theory (e.g., divisibility, prime numbers, groups, rings, and fields) and modular arithmetic, students used Java applets that were created using recommendations described in [8, 11]. The Sieve of Eratosthenes (see Fig. 5) was used for generating the prime numbers.

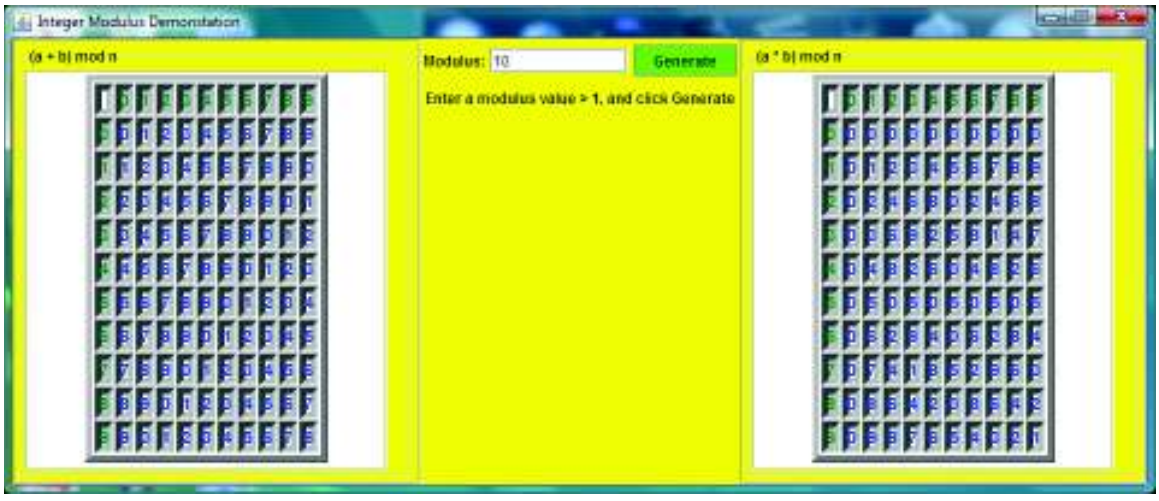

Figure 6. The Integer Modulus Demonstrator Java Applet.

The modular arithmetic operations could be done without worrying about whether we will exceed some large arithmetic bound; therefore, such calculations can be carried out on computers, even for large integer values. The Modular Arithmetic Applet (see Fig. 6) has been useful in analyses of multiplicative inverses [10] that are critical for applying the decryption algorithms [2, 5, 8, 11].

Sometimes the modular multiplicative inverse has a solution, and sometimes it does not. For example, the inverses of 2, 4, 5, 6, and 8 (mod 10) do not exist (see Fig. 6, right diagram; the rows related to integers 2, 4, 5, 6, and 8 do not contain a value 1 there). It turns out that $a^{-1} \equiv x \pmod{p}$ that has a solution iff a and p are relatively prime. In the considered case, the only rows that contain a 1 are for values that are relatively prime to $p = 10$: 1, 3, 7, 9. This example could be used in introducing the finite field of order p, known as the *Galois Field* [10], GF(p), which is defined as the set $\mathbf{Z}_p$ of integers {0, 1, ..., p - 1}, together with the arithmetic operations modulo p. The subset $\mathbf{Z}_p^*$ is defined as the set of (mod p) integers that are relatively prime to p. In this study case, $p = 10$ and $\mathbf{Z}_p^* = \{1, 3, 7, 9\}$. Every element in $\mathbf{Z}_{10}^*$ is present in the multiplicative table based on these four elements only, and no other elements other than those are present. Furthermore, every element in $\mathbf{Z}_{10}^*$ is present in every row of the table. It turns out that this is true for all p; therefore, $\mathbf{Z}_p^*$ is closed under multiplication (mod p).

This fundamental property of relative primes allows introducing Euler's totient function, $\varphi(p)$, which is defined [10] as the number of positive integers less than p, that are relatively prime to p. The $\varphi(p)$ function has the following properties:

$$\varphi(1) = 1$$
$$\varphi(p) = p - 1 \text{ (for } p \text{ prime)}$$
$$\varphi(m) < m - 1 \text{ (for } m \text{ composite)}$$

These properties lead to a conclusion that $\varphi(p)$ is just the number of elements in $\mathbf{Z}_p^*$. This fact laid the foundation to various modern encryption algorithms [2, 11], including the RSA public key encryption [5].

6. The Advanced Encryption Standard (AES)

In January 1997, the National Institute of Standards (NIST) announced a contest to select a new encryption standard to be used for protecting sensitive, non-classified, U.S. government information. After rigorous reviews of 5 final proposals, NIST chose a submission called *"Rijndael"* by two Belgian cryptographers – Joan Daemen and Vincent Rijmen [12]. *Rijndael* uses arithmetic in the *Galois Field* GF(2^8), the finite field of order 256. It can be shown [10] that the order of a finite field (number of elements in the field) must be a power of a prime, p^n, where n is a positive integer. Therefore, in *Rijndael* $n = 8$, and each element of the field can be represented by an octet. The bits in the octet are the coefficients of a polynomial over $\mathbf{Z}_2$ modulo the *irreducible* $\mathbf{Z}_2$ *polynomial* [10].

Byte values are represented as polynomials with the least significant bit being the coefficient of x^0, and the most significant bit the coefficient of x^7, e.g., {10100011}

identifies the specific field element: $x^7 + x^5 + x + 1$. Some finite field operations involve one additional bit to the left of an 8-bit byte. When this extra bit is present, it appears as {01} to the left of the other 8 bits: {01} {00011011}. Addition in a finite field is achieved by "adding" the coefficients for the corresponding powers in the polynomials for the two elements. This operation of addition is performed using an XOR operation denoted by $\oplus$. For example, all notations below are equivalent:

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 + 0$ [polynomial notation];
{01010111} $\oplus$ {10000011} = {11010100} [binary notation].

Multiplication in *Rijndael* is the multiplication of polynomials *modulo the irreducible polynomial* [10]. For example, in the polynomial notation:

$(x^6 + x^4 + x^2 + x + 1) \bullet (x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$, and

$(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \bmod (x^6 + x^4 + x^2 + x + 1) = x^7 + x + 1$.

The modular reduction by $m(x)$ ensures that the result will be a binary polynomial of degree less than 8, and thus can be represented in a byte. This multiplication is *associative*, and the element {01} is the *multiplicative identity*. For any non-zero binary polynomial $b(x)$ of degree less than 8, the multiplicative inverse of $b(x)$, denoted by $b^{-1}(x)$ can be found using the Extended Euclidean algorithm [10]. As it follows from the above, the set of 256 possible byte values, with XOR used as addition, and the multiplication defined as above, has the structure of the finite field GF(2^8). The detail description of the AES algorithm can be found in [4, 12].

7. Exploring Message Digests

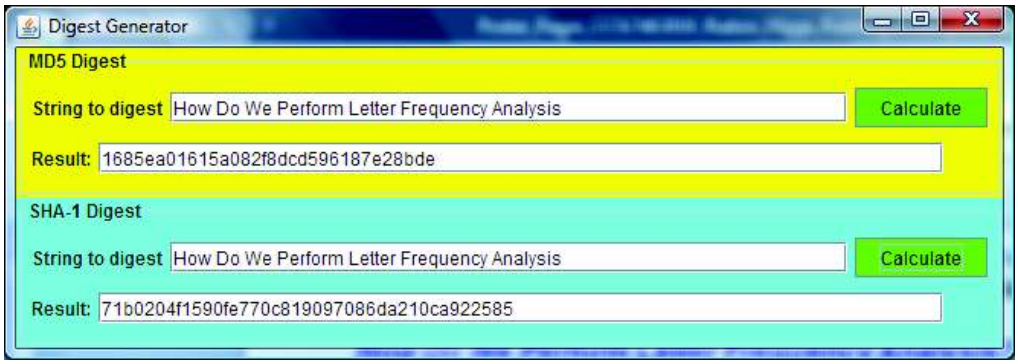

Figure 7. Message Digest Generator Applet.

The drive for hash/message digest algorithms began with public key cryptography. RSA was invented, but it was slow enough at that time to make it impractical when used alone. A cryptographically secure message digest algorithm with high performance would make RSA much more useful. After several attempts and improvements, R. Rivest came up with MD5 [13], a message digest algorithm that produces a one-way hash function that maps a message of any length into a fixed-size hash (128-bit) authenticator value. The example of creating the MD5 digest from the text message is shown in Fig. 7.

The National Institute of Standards approved a Secure Hash Algorithm, SHA-1 [14], for computing a condensed representation of a message. When a message of any length $< 2^{64}$ bits is input, the SHA-1 produces a 160-bit output called a message digest. The message digest can then be input to the Digital Signature Algorithm (DSA) which generates or verifies the signature for the message. Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller in size than the message. The same hash algorithm must be used by the verifier of a digital signature as was used by the creator of the digital signature. Students explored this technique by using the Digest Generator Java applet shown in Fig. 7.

8. Conclusions

The authors have described some algorithms, tools, and experience of using the Java Applets in computer security courses for seniors and graduate students. The experience has been in general a very positive one, while at the same time providing useful lessons learned. The authors believe that this algorithm-exploration and project-based approach with the Java Applets can be effectively applied to courses of a similar nature in academia, and the model can be extended to other areas of applied mathematics.

References

[1] Riabov, V. V. and Higgs, B. J. *J. Comput. Small Coll.*, 2010; 25(6): 245-247.
[2] Kaufman, C., Perlman, R., and Speciner, M. *Network Security: Private Communication in a Public World*, 2nd edition. Upper Saddle River, NJ: Prentice Hall, 2002.
[3] Grossman, J. Coding Theory: Introduction to Linear Codes and Applications. [Online] http://www.rivier.edu/journal/RCOAJ-Fall-2008_table.htm.
[4] Selent, D. Advanced Encryption Standard. *Rivier Academic Journal*, 2010, 6(2). [Online] http://www.rivier.edu/journal/archive/.
[5] Rivest, R., Shamir, A., and Adleman, L. *Communications of the ACM*, 1978; 21(2): 120-126.
[6] Riabov, V. V. *J. Comput. Small Coll.*, 2006; 21(6): 88-99.
[7] Linton, T. Shift Substitution Cipher Tool: ShiftApplet.java. [Online] http://pages.central.edu/emp/LintonT/classes/spring01/cryptography/java/Shift.html.
[8] Bishop, D. *Introduction to Cryptography with Java Applets*. Sudbury, MA: Jones & Bartlett Learning, 2002.
[9] Frequencies. [Online] http://www.simonsingh.net/The_Black_Chamber/frequencyanalysis.html.
[10] Graham, R. L., Knuth D. E., and Patashnik, O. *Concrete Mathematics*. Reading, MA: Addison-Wesley, 1994.
[11] Ferguson, N., and Schneier, B. *Practical Cryptography*. Hoboken, NJ: Wiley, 2002.
[12] Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard, FIPS-197. [Online] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[13] The MD5 Message-Digest Algorithm. *Request For Comments*, No. 1321. [Online] http://www.faqs.org/rfcs/rfc1321.html
[14] Secure Hash Standard. *Federal Information Processing Standards Publication 180-1*. National Institute of Standards. [Online] http://www.itl.nist.gov/fipspubs/fip180-1.htm