SOME ELEMENTARY CRYPTOGRAPHY INCLUDING RSA WITH THE TI-92 PLUS AND VOYAGE 200

Richard Klima
Department of Mathematical Sciences
Appalachian State University
Boone, North Carolina 28608
klimare@appstate.edu

Neil Sigmon
Department of Mathematics
Radford University
Radford, Virginia 24142
npsigmon@radford.edu

With society becoming more reliant on digital and computing technology, the ability to transfer information in a secure fashion using cryptography has increased in importance. As this reliance grows in the future, people will benefit from having at least some basic knowledge of this important topic. However, most students, especially those majoring in liberal arts and humanities curricula, have no formal training in any of the mathematical techniques used in cryptography. Much of this problem stems from the fact that many cryptographic techniques are too tedious to be done by hand, and we must rely instead on computing technology not readily available to all students for doing realistic examples.

The TI-92 Plus and Voyage 200 graphing calculators can be used to alleviate this problem. Many of the computations involved in doing cryptography are simplistic in nature. As a result, different types of algorithms for performing encryption and decryption can easily be programmed into the calculators. These algorithms include monoalphabetic affine ciphers, block ciphers (specifically, the Hill system), and public key methods involving the RSA cryptosystem. We have successfully taught these topics in mathematics courses ranging from introductory liberal arts courses to graduate level courses designed for teachers. The integration of the TI-92 Plus and Voyage 200 for performing the needed computations has been an essential part of the teaching process.

The purpose of this paper is to show how affine ciphers, the Hill cryptosystem, and the RSA cryptosystem can be implemented on the TI-92 Plus and Voyage 200 calculators. With each system we will also give a preliminary discussion of the user-written codes necessary for executing the commands that follow.

Note Concerning User-Written Codes

The functions **tonumber**, **toletter**, **inverse**, **nextprim**, and **expmod** are user-written procedures that are not part of the standard commands on the TI-92 Plus and Voyage 200. The code for these routines and details describing their use (along with other commands) can be found at the web site http://www.radford.edu/~npsigmon/ticalculator.htm.

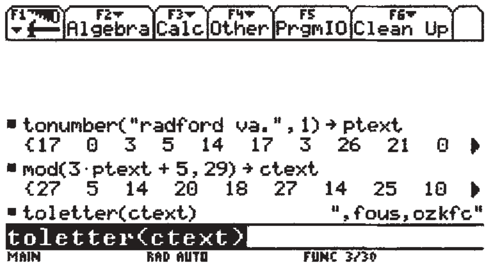
Character and Number Conversion

To convert between letters and numbers, we use the assignment $[a]=00$, $[b]=01$, $[c]=02, \ldots, [z]=25$, $[\ ]=26$, $[,]=27$, and $[.]=28$. On the TI-92 Plus and Voyage

200, the user-written function **tonumber** converts a message written in English into a numerical representation. The command is specified by **tonumber(*message, numchars*)**, where *message* is entered as lower case letters and symbols enclosed in quotation marks, and is converted into a list of integers representing *numchars* characters at a time. For example, the command **tonumber("appalachian state", 5)** converts the message into the numerical representation {15150011 2070800 1326181900 1904}. The function **toletter** converts a number or list of numbers into an alphabetic representation. The command is specified by **toletter(*nums*)**, where *nums* is either a single number or a list of numbers separated by commas and enclosed within curly braces { }. For example, the command **toletter(704111114)** returns "*hello*", and the command **toletter({1304222614, 1711040013, 1827261114, 2008180800, 130028})** returns "*new orleans, louisiana.*".

Affine Ciphers

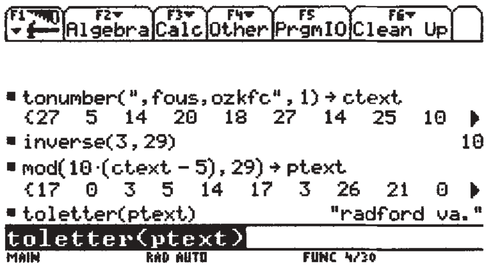
An affine cipher is a monoalphabetic cipher where encryption is performed using a linear function $f(x) = ax + b \bmod n$, with $a, b \in Z_n$, and $\gcd(a, n) = 1$. As an example, suppose we wish to encipher the message "*radford va.*" using the function $f(x) = 3x + 5 \bmod 29$. Each character will be enciphered separately, and Figure 1 below shows the TI-92 Plus or Voyage 200 screen after the corresponding ciphertext "*,fous,ozkfc*" has been determined.



```
■ tonumber(",radford va.", 1) → ptext
  {17  0  3  5  14  17  3  26  21  0 ▶
■ mod(3·ptext + 5, 29) → ctext
  {27  5  14  20  18  27  14  25  10 ▶
■ toletter(ctext)                ",fous,ozkfc"
toletter(ctext)
MAIN            RAD AUTO          FUNC 3/30
```

Figure 1: Affine cipher encipherment

To decipher the message, we use the inverse $f^{-1}(x) = a^{-1}(x - b) \bmod n$ of the affine function $f(x)$. To compute a^{-1} in Z_n, we can use the **inverse** function. This function, which employs the Euclidean algorithm, is specified by **inverse(*integer, modulus*)**, and returns the multiplicative inverse of *integer* with respect to *modulus*. So, for the encryption function $f(x) = 3x + 5 \bmod 29$, if we enter **inverse(3,29)** on the TI-92 Plus or Voyage 200, the result is 10, yielding the decryption function $f^{-1}(x) = 10(x - 5) \bmod 29$. Figure 2 below shows the commands that are then necessary to recover the message.



```
■ tonumber(",fous,ozkfc", 1) → ctext
  {27  5  14  20  18  27  14  25  10 ▶
■ inverse(3, 29)                          10
■ mod(10·(ctext - 5), 29) → ptext
  {17  0  3  5  14  17  3  26  21  0 ▶
■ toletter(ptext)             "radford va."
toletter(ptext)
MAIN            RAD AUTO          FUNC 4/30
```

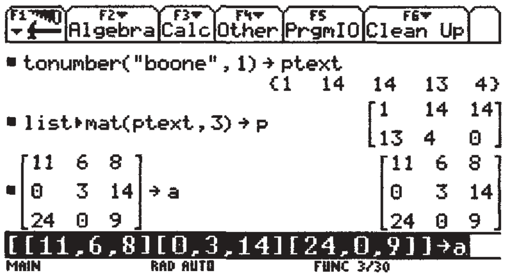
Figure 2: Affine cipher decipherment

The Hill Cryptosystem

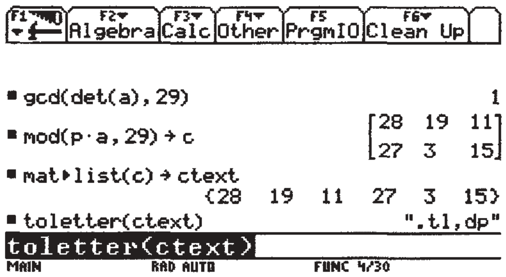
The Hill cryptosystem, which was first published by Lester Hill in 1929 and studied earlier by Jack Levine, uses an invertible matrix as the key and enciphers message characters in blocks instead of separately. Encryption is performed using a function of the form $f(P) = P \cdot A \mod n$, where the rows in the matrix P contain the numerical representations of the characters in the plaintext in order, the number of columns in P is compatible with matrix multiplication by the square key matrix A, and A is chosen to satisfy $\gcd(\det(A), n) = 1$. As an example, suppose we wish to encipher the message "*boone*" using the function $f(P) = P \cdot A \mod 29$ with the following key matrix A.

$$A = \begin{bmatrix} 11 & 6 & 8 \\ 0 & 3 & 14 \\ 24 & 0 & 9 \end{bmatrix}$$

To construct P on the TI-92 Plus or Voyage 200, the existing **list▶mat** command is useful for converting a list into a matrix. The command **list▶mat(*list, numperrow*)** uses the objects in *list* to create a matrix filled row by row from the top with *numperrow* objects within each row. And if there are not enough objects in *list* to completely fill the last row of the matrix, zeros are used. To construct the ciphertext, the existing **mat▶list** command is useful. The command **mat▶list(*matrix*)** converts the objects in *matrix* into a list, using the rows of *matrix* in order. Figure 3 below shows the commands necessary to produce the ciphertext "*.t1,dp*" using the message and key matrix we defined above.



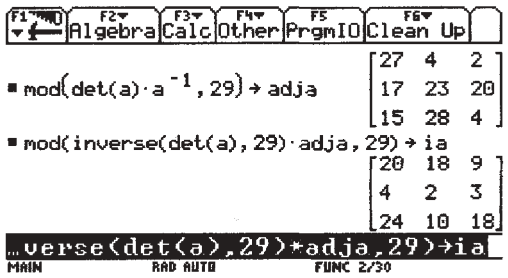
```
■ tonumber("boone", 1) → ptext
                              {1   14   14   13   4}
                                  [1   14  14]
■ list▸mat(ptext, 3) → p          [13  4   0 ]
 [11  6  8 ]                      [11  6   8 ]
■[0   3  14] → a                  [0   3   14]
 [24  0  9 ]                      [24  0   9 ]
[[11,6,8][0,3,14][24,0,9]]→a
MAIN          RAD AUTO          FUNC 3/30
```



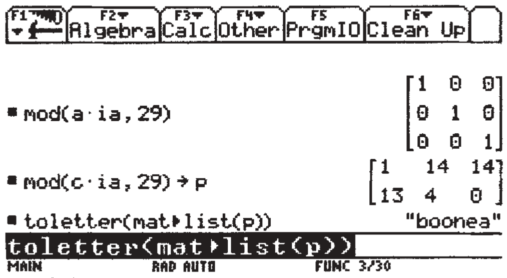
```
■ gcd(det(a), 29)                         1
                                  [28  19  11]
■ mod(p·a, 29) → c                [27  3   15]
■ mat▸list(c) → ctext
                    {28   19   11   27   3   15}
■ toletter(ctext)                     ".t1,dp"
toletter(ctext)
MAIN          RAD AUTO          FUNC 4/30
```

Figure 3: Hill system encipherment

To decipher a message that has been encrypted using the Hill system, we use the inverse function $f^{-1}(C) = C \cdot A^{-1} \mod n$. Thus, we must find the inverse A^{-1} of the key matrix A. To do this, we use the fact from linear algebra that $A^{-1} = \frac{1}{\det(A)} \cdot \operatorname{adj}(A)$. However, a problem arises because the TI-92 Plus and Voyage 200 have no existing commands for finding matrix adjoints. But since adjoints must have integer entries, we can compute $\operatorname{adj}(A) = \det(A) \cdot A^{-1}$. We can then find $\frac{1}{\det(A)}$ using the **inverse** routine, and use this to find $A^{-1} \mod n$ with integer entries. Figure 4 below shows how the inverse A^{-1} of the key matrix A we defined above can be determined, and how the plaintext message can then be recovered from the matrix C we constructed in the message encipherment.



```
■ mod(det(a)·a⁻¹, 29) → adja        [27  4   2 ]
                                    [17  23  20]
                                    [15  28  4 ]
■ mod(inverse(det(a), 29)·adja, 29) → ia
                                    [20  18  9 ]
                                    [4   2   3 ]
                                    [24  10  18]
…verse(det(a),29)*adja,29)→ia
```



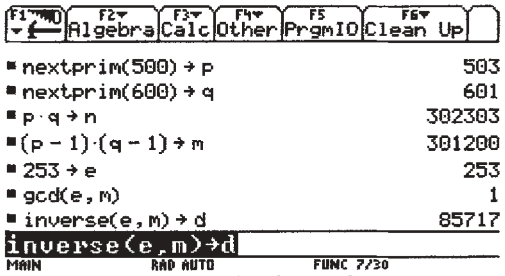
```
                                    [1  0  0]
■ mod(a·ia, 29)                     [0  1  0]
                                    [0  0  1]
                                 [1   14  14]
■ mod(c·ia, 29) → p              [13  4   0 ]
■ toletter(mat▸list(p))             "boonea"
toletter(mat▸list(p))
```

Figure 4: Hill system decipherment

The RSA Cryptosystem

The RSA cryptosystem is named for its developers, Rivest, Shamir, and Adleman, who first published the system in 1978. There is an initial setup stage in the system, in which we choose primes p and q, and form the products $n = pq$ and $m = (p-1)(q-1)$. We then choose a positive integer e with $\gcd(e,m) = 1$, which allows for the computation of the multiplicative inverse d of e modulo m (*i.e.*, such that $e \cdot d = 1 \bmod m$). The integers e and m are made public, making the RSA cryptosystem a public key system.

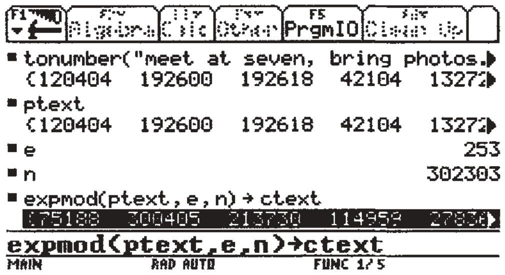
To assist in the RSA setup stage on the TI-92 Plus and Voyage 200, we can use the user-written **nextprim** command. This command is specified by **nextprim(*integer*)**, and returns the smallest prime integer that is larger than the input *integer*. Figure 5 below shows an RSA scheme that was created using the primes $p = 503$ and $q = 601$.



```
■ nextprim(500) → p                 503
■ nextprim(600) → q                 601
■ p·q → n                        302303
■ (p - 1)·(q - 1) → m            301200
■ 253 → e                           253
■ gcd(e, m)                           1
■ inverse(e, m) → d               85717
inverse(e,m)→d
```

Figure 5: RSA Setup Stage

Encryption in the RSA system is done using a function of the form $f(x) = x^e \bmod n$. The modulus n can be of any size, and thus message numbers do not need to be enciphered separately; they can be joined together to form larger numbers, provided the larger numbers stay smaller than n. For example, suppose we wish to encipher the message "*meet at seven, bring photos.*". In the previous two systems, we would convert these message characters separately, yielding the list 12, 04, 04, 19, … , 18, 28. But since $n = 302303$ in our RSA system, these message numbers can be joined together to form larger numbers corresponding to three characters at a time, yielding 120404, 192600, … , 191418, 28. This can be accomplished on the TI-92 Plus and Voyage 200 by entering the command **tonumber("meet at seven, bring photos.",3) →ptext**. Each of the resulting numbers will then be enciphered separately. Thus, with $e = 253$ and $n = 302303$, we encipher the first number by computing $f(120404) = 120404^{253} \bmod 302303$. To do this exponentiation in an efficient manner, we can use the user-written **expmod** command, which employs the well-known technique of repeated squaring and multiplying. This command is specified by **expmod(*base*, *exponent*, *modulus*)**, which returns the result of raising *base* to the power *exponent* and reducing the result modulo *modulus*. And as before, *base* can be either a single number or a list of numbers separated by commas and enclosed in curly braces { }. Figure 6 below shows how we can encipher our message.

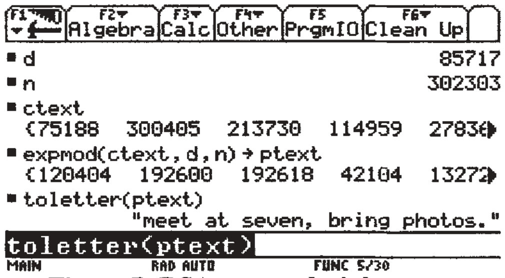


```
■ tonumber("meet at seven, bring photos.▶
  {120404  192600  192618  42104  13272▶
■ ptext
  {120404  192600  192618  42104  13272▶
■ e                                  253
■ n                               302303
■ expmod(ptext, e, n) → ctext
  {75188  300405  213730  114959  27836▶
expmod(ptext,e,n)→ctext
MAIN            RAD AUTO          FUNC 1/5
```

Figure 6: RSA system encipherment

Thus, we obtain the ciphertext 75188, 300405, 213730, 114959, … . And we would just leave this ciphertext in this numerical form since many of the resulting two-digit numerical values would have no corresponding letter in our alphabet assignment.

Fermat's Little Theorem can be used to show that $f^{-1}(x) = x^d \bmod n$ is the function necessary for deciphering messages. Figure 7 below shows how we can decipher the ciphertext we determined above using our values of $d = 85717$ and $n = 302303$.



```
■ d                                85717
■ n                               302303
■ ctext
  {75188  300405  213730  114959  27836▶
■ expmod(ctext, d, n) → ptext
  {120404  192600  192618  42104  13272▶
■ toletter(ptext)
            "meet at seven, bring photos."
toletter(ptext)
MAIN            RAD AUTO          FUNC 5/30
```

Figure 7: RSA system decipherment

Conclusion

In this paper, we have demonstrated how the TI-92 Plus and Voyage 200 calculators can be used to assist in teaching several different types of cryptographic systems. As we have shown, the TI-92 Plus and Voyage 200 allow the instructor to demonstrate realistic examples without the overhead of tedious hand computations.

This technology has been successfully used in many types of learning environments. Additional information can be obtained on classroom use by contacting the authors.