`

APPROVAL OF HONORS PROGRAM SENIOR PROJECT

Candidate

Alexis Reinert

Project Title

Two Simplified Post-Quantum Cryptosystems Demonstrated on the TI Graphing Calculator

This Senior Project is approved as acceptable

Project Director

Dr. Bill Yankosky

Committee Member

Dr. Brent Dozier

Committee Member

Dr. Jackie Lewis

Honors Program Director

Dr. Bill Yankosky

Honors Program Assistant Director

Dr. Fred Sanborn

November 21, 2023

Two Simplified Post-Quantum Cryptosystems Demonstrated on the TI Graphing Calculator

Alexis Reinert

North Carolina Wesleyan University

HON 402

November 21, 2023

Two Simplified Post-Quantum Cryptosystems Demonstrated on TI Graphing Calculator

Introduction

Mathematics is integrated into nearly every aspect of the modern world, so it should not be surprising that the science of sending secret messages relies heavily on mathematics. Cryptography is the study or technique of securing communication, data, or information through the use of mathematical concepts and codes. It is used interchangeably with cryptology, which is the study of secret writing in all its forms. Although cryptology is often depicted in movies and TV shows as something the treasure hunters need to solve to find their treasure, there are many real-world applications. These real-world applications involve protecting data and information, but as technology advances, the cryptosystems currently used to protect data are in jeopardy, which has led to a push by the National Institute of Standards and Technology for new cryptosystems.

Cryptography Background

For thousands of years, the only form of cryptology was private key ciphers. This required both parties, the sender and the receiver, to securely share the prearranged secret key. However, a major problem with the private key ciphers is that the sender and receiver would have to exchange the prearranged key without outsiders incepting this key, which makes private key encryption risky. In order to combat this extremely risky method of encryption, the idea of public key encryption was created. The idea behind public key ciphers is that there are two keys, one public and one private. The public key allows any individual to encrypt a message and send it to the receiver, but the receiver is the only individual with the private key, which makes them

the only person who can decrypt the messages that are being sent. Removing the need to exchange a private key eliminated the risk of others intercepting it (Singh, 2000, p. 243-292).

One of the most influential public key ciphers is known as the RSA cryptosystem. It is a prime example of a real-world application of cryptology, and it is still widely used today for things like encrypting emails or securing information.

The RSA Cryptosystem

With new technological developments, the security of the RSA cryptosystem could soon be in jeopardy, creating the need for a new cryptosystem to take its place. The math behind RSA utilizes modular arithmetic, which follows the notation used by Lewand, and is based on the concept of factoring (Lewand, 2000, p.19-22). A person selects two prime numbers (typically these prime numbers would be extremely large, but for the purpose of this example let's just choose p = 7 and q = 19). These prime numbers are then multiplied together to get another number, N. Thus, N = 133. Also, they need to select another number that is relatively prime to (p – 1) x (q – 1). So in this case that number can be e = 29. Now (e, N) can be published as the public key. In order to encrypt a message, it must be first converted to the form of an ASCII binary digit number. ASCII stands for the American Standard Code for Information Interchange, and it represents each letter of the alphabet with a 7-digit binary number. Also, each of the binary numbers can also be represented by a decimal number. Below is Table 1 which shows the ASCII binary numbers and the decimal numbers for the 26 capital case letters of the alphabet:

Table 1:

A	1000001	65	**N**	1001110	78
B	1000010	66	**O**	1001111	79
C	1000011	67	**P**	1010000	80
D	1000100	68	**Q**	1010001	81
E	1000101	69	**R**	1010010	82
F	1000110	70	**S**	1010011	83
G	1000111	71	**T**	1010100	84
H	1001000	72	**U**	1010101	85
I	1001001	73	**V**	1010110	86
J	1001010	74	**W**	1010111	87
K	1001011	75	**X**	1011000	88
L	1001100	76	**Y**	1011001	89
M	1001101	77	**Z**	1011010	90

Table 1: The ASCII binary number and decimal number system

The decimal form can be represented by M. In order to encrypt the ciphertext, C, the following formula is used:

$$C = M^e (mod\ N)$$

Suppose we wanted to encrypt the letter R. The ASCII binary form of R is 1010010, which in decimal is 82, which means M = 82. From the published public key we can determine that

$$C = 82^{29} (mod\ 133) = 17$$

We can now send this ciphertext to the person who published the public key. Because exponentials in modular arithmetic are a one-way function, anyone without the knowledge of the p and q values will find it extraordinarily difficult to decrypt the message. This is because by nature one-way functions, in general, are easy to do, but hard to undo. The person with the knowledge of p and q can calculate the decryption key, d, using the following formula:

$$e * d = 1 (mod (p-1) * (q-1))$$

So by substituting in the e, p, and q values, the d value can be computed.

$$29 * d = 1(mod(7-1) * (19-1))$$

$$29d = 1(mod\ 108)$$

$$d = 41$$

Once d is determined the following formula can be used to decrypt the message:

$$M = C^d(mod\ 133)$$

$$M = 17^{41}(mod\ 133) = 82$$

Using the table, 82 can be translated back into the plaintext "R", thus completing the decryption. It is important to note that in practice RSA is performed on groups of multiple letters, rather than on a single letter at a time. Also, we can note that e and d are multiplicative inverses with respect to the modulus $(p-1) * (q-1)$, thus d can be found using the Euclidean Algorithm as long as e is relatively prime to $(p-1) * (q-1)$. For a detailed explanation on the Euclidean Algorithm follow Dudley's notation (Dudley, 2008, p. 18).

Since the RSA algorithm was created, cryptanalysts have been looking for ways to break it. However, there are two main problems that are involved in attempting to break RSA. The first is factoring N into a product of two primes. In order to get d, the decryption key, a person would need the encryption key, e, which is public. However, they would also need to know (p – 1) x (q – 1), which means they need to know both p and q. This would require the factoring of N. The straightforward way to find the values of p and q is to check every prime number, that is less than the square root of N, one by one to see if it divides N (Dudley, 2008, p. 43). However the

real applications of RSA use prime numbers that are extremely large, and any brute-force attempt would take a very long time.

The second problem correlates to the fact that the RSA encryption formula is $C = M^e (mod\ N)$. With the knowledge of M, e, and N it is fairly easy to compute C, but if someone were intercept C, the ciphertext, they would have to work backwards to find the M value. Trying the solve $C = M^e (mod\ N)$, with the knowledge of C, e, and N is known as a discrete logarithm problem, because of the connection to logarithms. Typically, a person could take the log of both sides of the equation and then solve, but logs cannot be done when modular arithmetic is involved. This brings us back to factoring to find the p and q values, which as we know takes a very long time. Over the years, many cryptanalysts have looked for ways to cut down on the time it would take to solve for these factors, but nothing seemed to work.

In 1965, Gordon E. Moore introduced what was known as Moore's law, which addressed the idea that computer performance would double approximately every eighteen months (Moore, 2006). However, despite the fact that computers were constantly getting faster, it was still not enough to break RSA in a timely manner. Since all shortcut attempts to break RSA had failed, cryptanalysts turned to looking for new technology: "If there is no obvious way to reduce the number of steps required for factoring, then cryptanalysts need a technology that will perform these steps more quickly. Silicon chips will continue to get faster as the years pass, doubling in speed roughly every eighteen months, but this is not enough to make a real impact on the speed of factoring—cryptanalysts require a technology that is billions of times faster than current computers. Consequently, cryptanalysts are looking towards a radically new form of computer, the quantum computer" (Singh, 2000, p. 320). This technology would make the current cryptosystems insecure.

Quantum Computers

The basis of quantum computers and what makes them different from ordinary computers is the concept of superposition. Superposition is the idea that something can exist in more than one state at a time. Physicist, Erwin Schrodinger is famous for something known as Schrodinger's cat, which is a hypothetical experiment that explains superposition: "Imagine a cat in a box. There are two possible states for the cat, namely dead or alive. Initially, we know that the cat is definitely in one particular state, because we can see that it is alive. At this point, the cat is not in a superposition of states. Next, we place a vial of cyanide in the box along with the cat and close the lid. We now enter a period of ignorance, because we cannot see or measure the state of the cat" (Singh, 2000, p. 324). Now we have no idea what the state of the cat is. It could have stepped on the cyanide and died, or it could still be alive. However, instead of determining whether the cat is either dead or alive, the concept of superposition allows us to view the cat as both dead and alive. Thus it satisfies all possible states. Obviously, once the box is opened, we will be able to determine the actual state of the cat, but until then quantum theory suggests that the cat is in two states, and this is known as superposition. This correlates to a quantum computer because a quantum bit can also exist in different states at the same time. The concept of superposition does not exist on ordinary computers, which is why they do not have the power to quickly crack cryptosystems: "Imagine that you have two versions of a question. To answer both questions using an ordinary computer you would have to input the first version and wait for the answer, then input the second version and wait for the answer. In other words, an ordinary computer can address only one question at a time, and if there are several questions it has to address them sequentially. However, with a quantum computer, the two questions could be combined as a superposition of two states and inputted simultaneously—the machine itself

would then enter a superposition of two states, one for each question" (Singh, 2000, p. 326-327). This technology would give cryptanalysts a huge advantage in cracking cryptosystems like RSA. In the late 1900's, quantum algorithms began to be developed: "They led directly to the first 'useful' quantum algorithm in 1994, when Peter Shor discovered an algorithm to (probabilistically) factor numbers using a quantum computer faster than any known algorithm for conventional computers. In fact, this algorithm can factor numbers roughly as fast as any algorithm, quantum or conventional, can find large prime numbers. So widespread use of this algorithm would make RSA completely insecure" (Holden, 2017, p. 280).

While cryptanalysts intend to take advantage of the abilities of quantum computers to crack current cryptosystems, cryptographers also hope to utilize quantum computers to establish new cryptosystems, known as Quantum Cryptography, that can withstand quantum computers: "Furthermore, it is based on quantum theory, the same theory that is the foundation for quantum computers. So while quantum theory is the inspiration for a computer that could crack all current ciphers, it is also at the heart of a new unbreakable cipher called quantum cryptography" (Singh, 2000, p. 332). Quantum cryptosystems have been studied and developed, but the details of those systems are beyond the scope of this paper.

Post-Quantum Cryptography

Once quantum computers are larger scale, more commonly used, and have the ability to crack encrypted information at high speeds, all current public key ciphers will essentially be obsolete. This triggers the need for something known as Post-quantum cryptography. Post-quantum cryptography, also known as quantum-resistant cryptography, are cryptosystems that are not known to be easily solved by a quantum computer but can still be run on ordinary computers. Lattice-based cryptography is the basis for a lot of these new post-quantum

cryptosystems: “A lattice is an evenly spaced grid of points in an n-dimensional space equipped with coordinate axes” (Holden, 2017, p. 282). Within lattice-based problems, there are two main concepts that make it difficult for a quantum computer to solve— the shortest-vector problem and the closest-vector problem. The shortest-vector problem uses specific generators of a lattice to find a point that is located in the lattice and also as close as possible to the axis’s origin. The closest vector problem uses specific generators of a lattice as well as a point located outside of the lattice to find a point located in the lattice that is closest to the point located outside the lattice. Both of these can be depicted in Figure 1 and Figure 2 below, where the lattice is shown by the open circles:

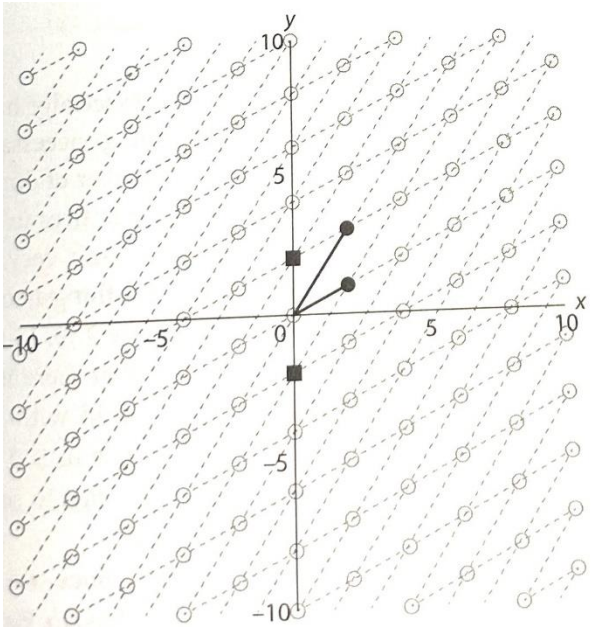


Figure 1: The shortest vector problem: The solid dots represent the generators of the lattice and solid square boxes represent the closest point to the origin that is still in the lattice (Holden, 2017, p. 283)

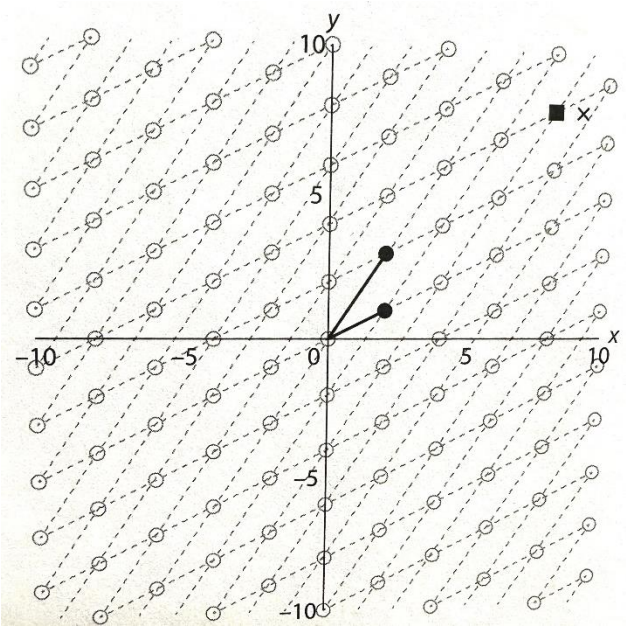


Figure 2: The closest vector problem: The solid dots represent the generators of the lattice, the x is a point not on the lattice, and the solid square box is the closest point in the lattice to the x (Holden, 2017, p. 283).

Increasing the dimensions of a lattice makes these two problems increasingly difficult to solve.

Although there are other forms of post-quantum cryptography, lattice-based cryptography led to the development of two cryptosystems, Frodo, which utilizes matrices, and NewHope, which utilizes polynomials. However, both Frodo and NewHope are based on KYBER, which was developed by the organization, Cryptographic Suite for Algebraic Lattices (Crystals). Crystals-KYBER is a way to secure a key utilizing the learning with errors problem, which is the equation T = AS + E, where A, E, and T can be used to solve for S (Holden, 2022). These cryptosystems originated from NIST's call for new cryptosystems: "the National Institute of Standards and Technology (NIST) started in 2016 an international call to determine new standard algorithms for the so called post-quantum cryptography (PQC), i.e., to propose sufficiently

secure algorithms to resist the attacks from quantum computers. This process is being developed in several rounds. The first round received 69 submissions, but only 15 of them have reached the current third round" (Gonzalez de la Torre & Dios, 2022). We are going to break down the Frodo and the NewHope cryptosystems and show small-scale examples of how they work. These simplified versions are respectively named Smeagol and Phantom. Also, we are going to demonstrate the high power of the TI-84 graphing calculator to run encryption and decryption programs for each of these cryptosystems.

The Smeagol Cryptosystem

The first cryptosystem known as Smeagol is a public key cryptosystem that is based on the Learning With Errors (LWE) problem, and it utilizes matrices for both encryption and decryption. The private key is a $k\ x\ l$ matrix S and the public key is (A, T) where A is a $k\ x\ k$ public matrix. E is a $k\ x\ l$ error matrix, and $T \equiv AS + E\ (mod\ q)$. After computing matrix T the next step is to encrypt a message. A plaintext message (p) can be represented by a k-bit binary vector, where the different combinations can represent letters of the alphabet. The encryption is done by using two small random errors vectors, e_1 and e_2, and a random vector, r. Then the ciphertext is computed and displayed as (u, v), where:

$$u = (A^T r + e_1)\ mod\ q$$

and

$$v = (T^T r + e_2 + \lfloor q/2 \rceil p)\ mod\ q$$

Note that $\lfloor x \rceil$ means to round x to the nearest integer.

Now suppose two people, Taylor and Travis, wanted to exchange a message using the Smeagol encryption algorithm with the following values: $(k = 5,\ l = 5,\ q = 23)$

Suppose that Taylor starts with random matrices A, S, and E:

$$A \equiv \begin{bmatrix} 7 & 21 & 8 & 18 & 0 \\ 15 & 6 & 14 & 16 & 0 \\ 7 & 7 & 5 & 13 & 0 \\ 20 & 3 & 4 & 13 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S \equiv \begin{bmatrix} 1 & 0 & 2 & 1 & 0 \\ 0 & -1 & 1 & 3 & 0 \\ 1 & 0 & 0 & -1 & 0 \\ 1 & 2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$E \equiv \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

From these matrices, Taylor can compute matrix T:

$$T \equiv AS + E\ (mod\ q)$$

So, $T \equiv \begin{bmatrix} 7 & 21 & 8 & 18 & 0 \\ 15 & 6 & 14 & 16 & 0 \\ 7 & 7 & 5 & 13 & 0 \\ 20 & 3 & 4 & 13 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 & 1 & 0 \\ 0 & -1 & 1 & 3 & 0 \\ 1 & 0 & 0 & -1 & 0 \\ 1 & 2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} (mod\ 23)$

$$T \equiv \begin{bmatrix} 11 & 15 & 8 & 11 & 0 \\ 22 & 4 & 7 & 13 & 0 \\ 3 & 19 & 11 & 13 & 0 \\ 14 & 1 & 10 & 15 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

After matrix T is computed, Taylor can publish (A, T) as the public key.

$$(A,T) = \left(\begin{bmatrix} 7 & 21 & 8 & 18 & 0 \\ 15 & 6 & 14 & 16 & 0 \\ 7 & 7 & 5 & 13 & 0 \\ 20 & 3 & 4 & 13 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 11 & 15 & 8 & 11 & 0 \\ 22 & 4 & 7 & 13 & 0 \\ 3 & 19 & 11 & 13 & 0 \\ 14 & 1 & 10 & 15 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \right)$$

Next, Travis will select a plaintext message to encrypt. Suppose that Travis wants to send Taylor the letter "T". Since k = 5, the p-value will be a 5-bit vector that represents the chosen letter of the alphabet that is being encrypted, in this case, the letter is "T". Table 2 displays the different letters that the 5-bit vectors represent.

Table 2:

A	00001	**N**	01110
B	00010	**O**	01111
C	00011	**P**	10000
D	00100	**Q**	10001
E	00101	**R**	10010
F	00110	**S**	10011
G	00111	**T**	10100
H	01000	**U**	10101
I	01001	**V**	10110
J	01010	**W**	10111
K	01011	**X**	11000
L	01100	**Y**	11001
M	01101	**Z**	11010

So $p = [1 \quad 0 \quad 1 \quad 0 \quad 0]^T$

Then, Travis uses random error vector $e_1 = [1 \quad 0 \quad 0 \quad 0 \quad 1]^T$ random error vector

$e_2 = [0 \quad 0 \quad 0 \quad -1 \quad 1]^T$, and random vector $r = [0 \quad 1 \quad -1 \quad 1 \quad 1]^T$, as well as

$p = [1 \quad 0 \quad 1 \quad 0 \quad 0]^T$ to compute u and v vectors.

$$u = (A^T r + e_1) \bmod q$$

$$= \left(\begin{bmatrix} 7 & 21 & 8 & 18 & 0 \\ 15 & 6 & 14 & 16 & 0 \\ 7 & 7 & 5 & 13 & 0 \\ 20 & 3 & 4 & 13 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T [0 \quad 1 \quad -1 \quad 1 \quad 1]^T + [1 \quad 0 \quad 0 \quad 0 \quad 1]^T \right) mod\ 23$$

$$= \left(\begin{bmatrix} 7 & 15 & 7 & 20 & 0 \\ 21 & 6 & 7 & 3 & 0 \\ 8 & 14 & 5 & 4 & 0 \\ 18 & 16 & 13 & 13 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right) mod\ 23$$

$$= \left(\begin{bmatrix} 28 \\ 2 \\ 13 \\ 16 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right) mod\ 23$$

$$= \begin{bmatrix} 29 \\ 2 \\ 13 \\ 16 \\ 2 \end{bmatrix} mod\ 23 = \begin{bmatrix} 6 \\ 2 \\ 13 \\ 16 \\ 2 \end{bmatrix}$$

$$v = (T^T r + e_2 + \lfloor q/2 \rceil p)\ mod\ q$$

$$= \left(\begin{bmatrix} 11 & 15 & 8 & 11 & 0 \\ 22 & 4 & 7 & 13 & 0 \\ 3 & 19 & 11 & 13 & 0 \\ 14 & 1 & 10 & 15 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}^T \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} + \left\lfloor \frac{23}{2} \right\rceil \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right) mod\ 23$$

$$= \left(\begin{bmatrix} 11 & 22 & 3 & 14 & 0 \\ 15 & 4 & 19 & 1 & 0 \\ 8 & 7 & 11 & 10 & 0 \\ 11 & 13 & 13 & 15 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} + 12 \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right) mod\ 23$$

$$= \left(\begin{bmatrix} 33 \\ -14 \\ 6 \\ 15 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 12 \\ 0 \\ 12 \\ 0 \\ 0 \end{bmatrix} \right) mod\ 23$$

$$= \begin{bmatrix} 45 \\ -14 \\ 18 \\ 14 \\ 3 \end{bmatrix} mod\ 23 = \begin{bmatrix} 22 \\ 9 \\ 18 \\ 14 \\ 3 \end{bmatrix}$$

Thus the encryption algorithm gives $u = [6 \quad 2 \quad 13 \quad 16 \quad 2]^T$ and

$v = [22 \quad 9 \quad 18 \quad 14 \quad 3]^T$. Then Travis send the u and v values to Taylor. From this, Taylor can use the decryption algorithm to decrypt the message. The decryption formula is as follows:

$$p' = \left\lfloor \lfloor q/2 \rceil^{-1} \left((v - S^T u) mod\ 23 \right) \right\rceil mod\ 2$$

Decryption Example:

$$p' = \left\lfloor \lfloor 23/2 \rceil^{-1} \left(\left(\begin{bmatrix} 22 \\ 9 \\ 18 \\ 14 \\ 3 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 2 & 1 & 0 \\ 0 & -1 & 1 & 3 & 0 \\ 1 & 0 & 0 & -1 & 0 \\ 1 & 2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T \begin{bmatrix} 6 \\ 2 \\ 13 \\ 16 \\ 2 \end{bmatrix} \right) mod\ 23 \right) \right\rceil mod\ 2$$

$$= \left\lfloor \frac{1}{12} \left(\left(\begin{bmatrix} 22 \\ 9 \\ 18 \\ 14 \\ 3 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & 2 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 1 & 3 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 2 \\ 13 \\ 16 \\ 2 \end{bmatrix} \right) mod\ 23 \right) \right\rceil mod\ 2$$

$$= \left\lfloor \frac{1}{12} \left(\left(\begin{bmatrix} 22 \\ 9 \\ 18 \\ 14 \\ 3 \end{bmatrix} - \begin{bmatrix} 35 \\ 30 \\ 30 \\ 15 \\ 2 \end{bmatrix} \right) mod\ 23 \right) \right\rceil mod\ 2$$

$$= \left\lfloor \frac{1}{12} \left(\begin{bmatrix} -13 \\ -21 \\ -12 \\ -1 \\ 1 \end{bmatrix} mod\ 23 \right) \right\rfloor mod\ 2$$

$$= \left\lfloor \frac{1}{12} \left(\begin{bmatrix} 10 \\ 2 \\ 11 \\ 22 \\ 1 \end{bmatrix} \right) \right\rfloor mod\ 2$$

$$= \left\lfloor \begin{bmatrix} {}^{10}/_{12} \\ {}^{2}/_{12} \\ {}^{11}/_{12} \\ {}^{22}/_{12} \\ {}^{1}/_{12} \end{bmatrix} \right\rfloor mod\ 2$$

$$= \begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \\ 0 \end{bmatrix} mod\ 2 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = [1 \quad 0 \quad 1 \quad 0 \quad 0]^T = \text{The letter “T”}$$

Finally Taylor can see that the message Travis sent her was the letter “T”

Each letter can be encrypted and decrypted using the same process, however this could take a while doing the math by hand. By using the programs that were written on the TI-84 graphing calculator, which can be seen in the appendix, the process can be sped up and encrypted and decrypted faster.

The Smeagol Cryptosystem on the TI Graphing Calculator

There are three programs that can be used to run through a complete example of the Smeagol Cryptosystem. The first program is used to determine the public key, the second

program is used for encryption, and the third program is used for decryption. There are nine matrix storage slots on the TI-84 Plus CE graphing calculator. Before running any of the programs the appropriate matrices and vectors must be pre-entered. For the public key program matrices A, S, and E are required and can be entered by selecting the matrix option then going to the edit tab. Matrices A, S, E, must be entered as 5x5 matrices. Below are images of each of these matrices entered into a different matrix storage slot.

Image 1: Matrix A

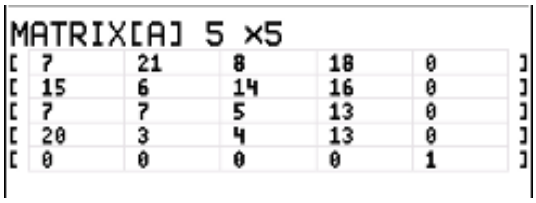


Image 2: Matrix S

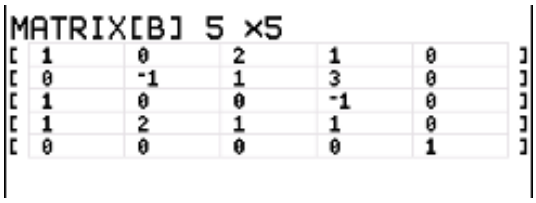
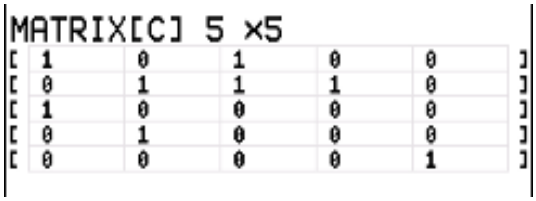
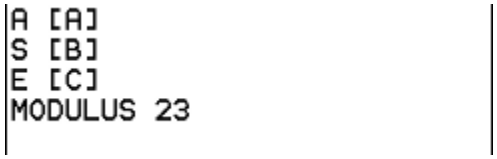


Image 3: Matrix E



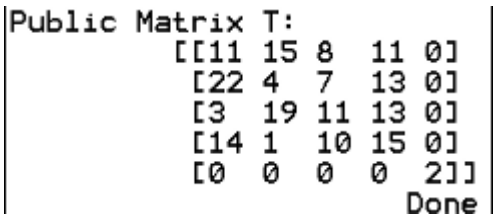
After entering these matrices, the program can be run. It will first ask for the input of each of the matrix slots that hold the values A, S, and E, as well as the modulus, which is 23.

Image 4: Input Screen



Then the program will display Matrix T.

Image 5: Matrix T



Then the public key can be published.

Now using the encryption calculator program let's encrypt the word "MATH". So using Table 2 the P vectors can be determined to be $p_1 = [0 \quad 1 \quad 1 \quad 0 \quad 1]^T$,

$$p_2 = [0 \quad 0 \quad 0 \quad 0 \quad 1]^T, p_3 = [1 \quad 0 \quad 1 \quad 0 \quad 0]^T, and\ p_4 = [0 \quad 1 \quad 0 \quad 0 \quad 0]^T.$$

Before running the program, each of the required matrices and vectors, A, T, e_1, e_2, r, p, must be entered into one of the matrix storage slots. Matrices A and T must be stored as 5x5 matrices and the vectors e_1, e_2, r, and p, can be entered as 5x1 matrices. Below are images of each of the matrices and vectors entered into a different matrix slot on the calculator.

Image 6: Matrix A

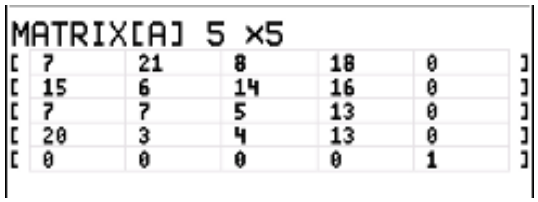


Image 7: Matrix T

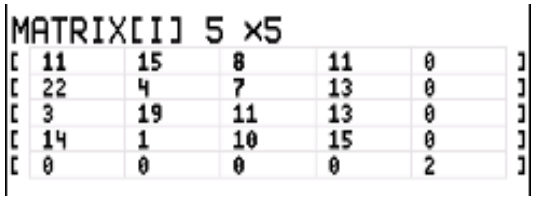


Image 8: Vector e_1

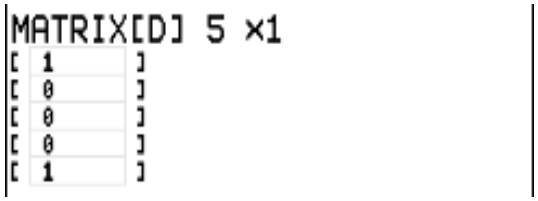


Image 9: Vector e_2

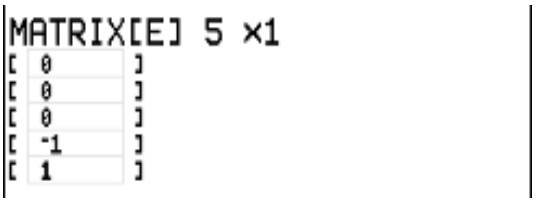


Image 10: Vector r

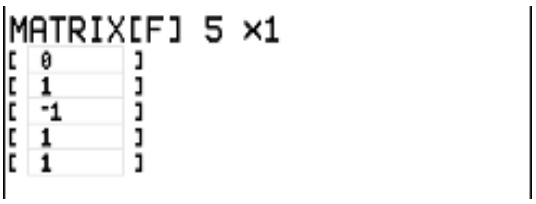
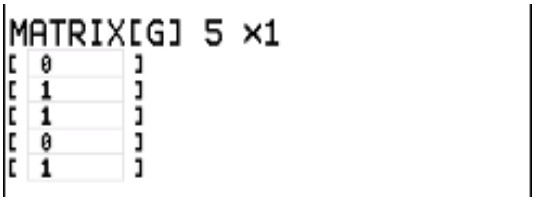
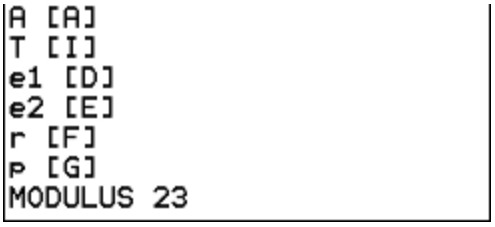


Image 11: Vector p_1



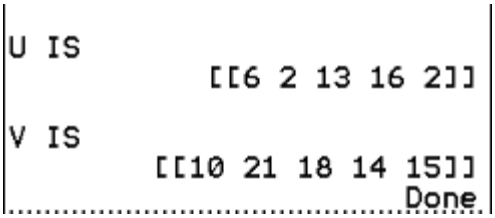
After all the necessary matrices and vectors are entered, then the program can be run. The program will ask for the input of each of the matrix slots that are holding the values A, T, e_1, e_2, r, and p, as well as the modulus value, which in this case is 23.

Image 12: Input Screen



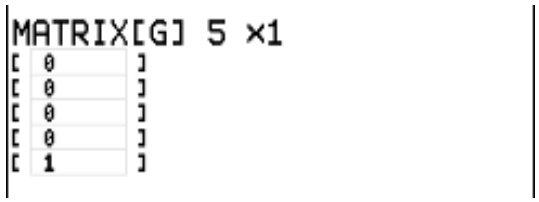
Next the program displays the encrypted u and v values.

Image 13: u and v values



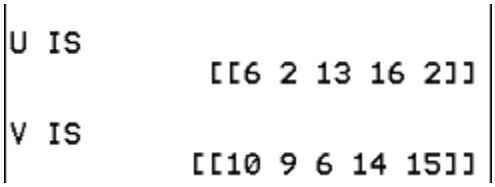
The program can be repeated to encrypt the rest of the plaintext. The only matrix slot that needs to be changed is the p value, which is stored in the matrix G slot. It can be changed to vector p_2.

Image 14: Vector p_2



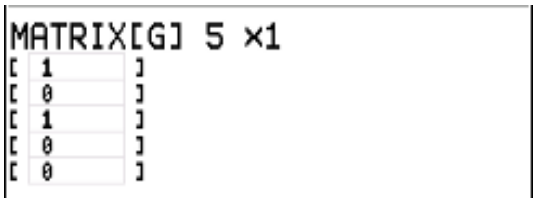
The program can be run using the same steps as before and the following u and v will be displayed:

Image 15: u and v values



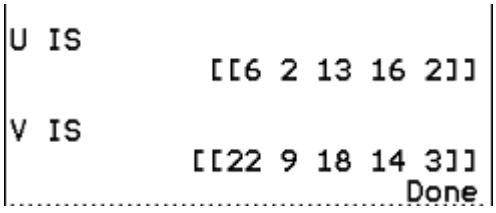
The p value can be replaced again for p_3.

Image 16: Vector p_3



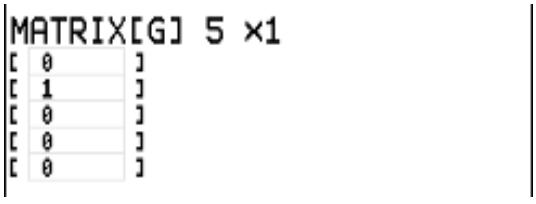
Using the program with p_3 yields the following u and v values:

Image 17: u and v values



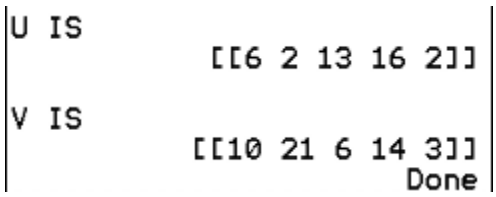
Finally the last p value, p_4, can be entered.

Image 18: Vector p_4:



This gives the following u and v values:

Image 19: u and v values



This concludes the encryption process.

Now using the u and v values from the previous encryption example, the decryption calculator program can be used to recover the plaintext. For each set of u and v values the decryption program must be run. The decryption formula is as follows:

$$p' = \left\lfloor \left\lfloor \frac{q}{2} \right\rfloor^{-1} \left((v - S^T u) mod\ q \right) \right\rfloor mod\ 2,$$

so the decryption program needs the values for v, u, and S entered into matrix storage slots.

The first set of u and v values can be entered:

Image 20: Vector u

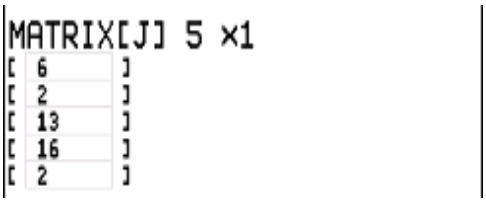
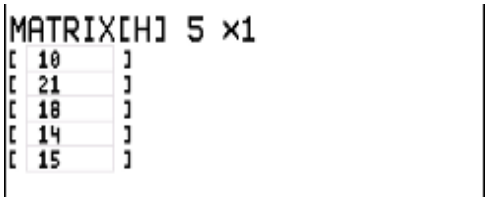


Image 21: Vector v



The S matrix is the same as listed in the encryption example and the q value is the same as well. Then the decryption program can be run and the location where the u, v, and S values are stored can be input.

Image 22: Input Screen

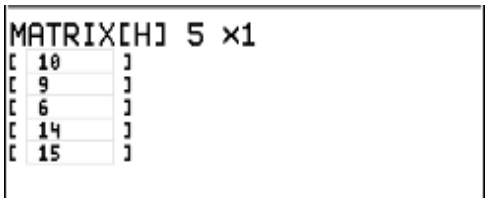
Then the program will display the p' vector, which will be the original p vector, the plaintext.

Image 23: p' value

If we look at Table 2, we can see that this p' value means the plaintext is the letter "M".

The next set of u and v values can be entered into the calculator and the program can be run again. The u vector is the same as the first decryption.

Image 24: Vector v



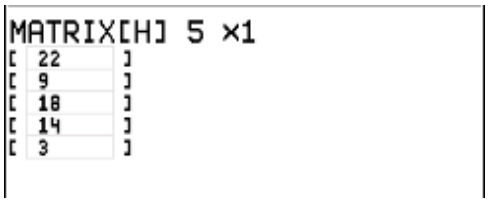
After running the program and following the same steps as before, the following p' is found:

Image 25: Vector p’

Looking at Table 2, we can determine that the plaintext is the letter “A”.

Once again the third set of u and v values can be entered into matrix storage slots. The u value still remains the same.

Image 26: Vector v



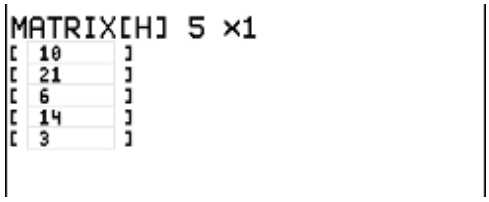
By running the program we can get the following p’ value.

Image 27: Vector p’

This vector translates to the plaintext of the letter “T”.

Finally, the last set of u and v can be entered. The u vector is the same as before.

Image 28: Vector v



This set of u and v vectors yield the following p' vector:

Image 29: Vector p'

This p' vector represents the letter "H". So by putting all the plaintext letter together, we get the decrypted message "MATH".

The Phantom Cryptosystem

The second example is known as the Phantom cryptosystem. This algorithm involves performing modular arithmetic on polynomials. Performing modular arithmetic on polynomials means using polynomial long division. For further explanation of polynomial long division refer to Gallian's book (Gallian, 2017, p. 280-281). This private key is represented by s(x) and the public key is published as (a(x), t(x)), where

$$t(x) \equiv \big((a(x)s(x) + e(x))(mod\ x^n + 1)\big)(mod\ q)$$

In this equation the e(x) is a random error polynomial with small coefficients. In order to encrypt, three random vector polynomials are needed, r(x), $e_1(x)$, and $e_2(x)$, as well as a plaintext vector, p(x). Then the ciphertext is computed and displayed as (u, v), where:

$$u(x) = \Big(\big(a(x)r(x) + e_1(x)\big)\ mod(x^n + 1)\Big) mod\ q$$

and

$$v(x) = \left(\left(t(x)r(x) + e_2(x) + \left\lfloor\frac{q}{2}\right\rceil p(x)\right) mod(x^n + 1)\right) mod\ q$$

Suppose that Taylor and Travis want to exchange a message using the Phantom cryptosystem and the following values: (n=5, q=23)

Suppose Taylor starts with the polynomials a(x), s(x), and e(x):

$$a(x) = 18x^3 + 10x^2 + 22x + 6$$

$$s(x) = x^3 - x^2 - x - 1$$

$$e(x) = x^2 + x$$

From these polynomials, Taylor can compute t(x):

$t(x) \equiv ((a(x)s(x) + e(x))(mod\ x^n + 1))(mod\ q)$

$= (((18x^3 + 10x^2 + 22x + 6)(x^3 - x^2 - x - 1) + (x^2 + x))mod(x^5 + 1))mod\ 23$

$= (\big((18x^6 - 8x^5 - 6x^4 - 44x^3 - 38x^2 - 28x - 6) + (x^2 + x)\big)mod(x^5 + 1))mod\ 23$

$= ((18x^6 - 8x^5 - 6x^4 - 44x^3 - 37x^2 - 27x - 6)mod(x^5 + 1))mod\ 23$

$= (-6x^4 - 44x^3 - 37x^2 - 45x - 2)\ mod\ 23$

$= 17x^4 + 2x^3 + 9x^2 + x + 2$

After t(x) is computed (a(t), t(x)) is published as the public key.

$$\big(a(x), t(x)\big) = ((18x^3 + 10x^2 + 22x + 6), (17x^4 + 2x^3 + 9x^2 + x + 2))$$

Next, Travis will select a plaintext message to encrypt. Suppose that Travis wants to send Taylor the letter “T” again. The 5-bit binary plaintext vector that represents the letter “T” can be determine from Table 2, $p(x) = 1x^4 + 0x^3 + 1x^2 + 0x + 0 = x^4 + x^2$.

Travis then uses random vectors $r(x) = x^3 + 1$, $e_1(x) = -x$, and $e_2(x) = -x^3 + x^2 - 1$, as well as the 5-bit binary plaintext vector, $p(x) = 1x^4 + 0x^3 + 1x^2 + 0x + 0 = x^4 + x^2$, to compute u(x) and v(x).

$$u(x) = (\big(a(x)r(x) + e_1(x)\big)\ mod(x^n + 1))mod\ q$$

$$= (\big((18x^3 + 10x^2 + 22x + 6)(x^3 + 1) + (-x)\big)\ mod(x^5 + 1))mod\ 23$$

$$= \Big(\big((18x^6 + 10x^5 + 22x^4 + 24x^3 + 10x^2 + 22x + 6) + (-x)\big)mod(x^5 + 1)\Big)\ mod\ 23$$

$$= (\big((18x^6 + 10x^5 + 22x^4 + 24x^3 + 10x^2 + 21x + 6)\big)\ mod(x^5 + 1))mod\ 23$$

$$= (22x^4 + 24x^3 + 10x^2 + 3x - 4)\ mod\ 23$$

$$= (22x^4 + x^3 + 10x^2 + 3x + 19)$$

$$v(x) = \left(\left(t(x)r(x) + e_2(x) + \left\lfloor\frac{q}{2}\right\rfloor p(x)\right)mod(x^n + 1)\right)mod\ q$$

$$= \left(\left((17x^4 + 2x^3 + 9x^2 + x + 2)(x^3 + 1) + (-x^3 + x^2 - 1)\right.\right.$$

$$\left.\left. + \left\lfloor\frac{23}{2}\right\rfloor (x^4 + x^2)\right)\ mod(x^5 + 1)\right)mod\ 23$$

$$= \left(\left((17x^7 + 2x^6 + 9x^5 + 18x^4 + 4x^3 + 9x^2 + x + 2) + (-x^3 + x^2 - 1)\right.\right.$$

$$\left.\left. + \left\lfloor\frac{23}{2}\right\rfloor (x^4 + x^2)\right)mod(x^5 + 1)\right)mod\ 23$$

$$= \big((17x^7 + 2x^6 + 9x^5 + 30x^4 + 3x^3 + 22x^2 + x + 1))mod(x^5 + 1)\big)mod\ 23$$

$$= (30x^4 + 3x^3 + 5x^2 - x - 8) mod\ 23$$

$$= (7x^4 + 3x^3 + 5x^2 + 22x + 15)$$

Thus the encryption algorithm gives

$u(x) = (22x^4 + x^3 + 10x^2 + 3x + 19)$ and $v(x) = (7x^4 + 3x^3 + 5x^2 + 22x + 15)$.

Travis then sends these values to Taylor, and she uses the decryption algorithm to translate the plaintext message. The decryption algorithm is as follows:

$$p'(x) = \left\lfloor \frac{\left((v(x) - s(x)u(x)) mod (x^n + 1)\right) mod\ q}{\lfloor q/2 \rfloor} \right\rceil mod\ 2,$$

where v(x) and u(x) are the ciphertext, s(x) is the secret key shown before, n = 5, and q = 23.

Decryption Example:

$$p'(x) = \left\lfloor \frac{\left(\left((7x^4+3x^3+5x^2+22x+15) - (x^3-x^2-x-1)\left((22x^4+x^3+10x^2+3x+19)\right)\right) mod (x^5+1)\right) mod\ 23}{\lfloor 23/2 \rfloor} \right\rceil mod\ 2$$

$$= \left\lfloor \frac{\left((7x^4+3x^3+5x^2+22x+15) - (22x^7-21x^6-13x^5-30x^4+5x^3-32x^2-22x-19)\right) mod (x^5+1)\right) mod\ 23}{\lfloor 23/2 \rfloor} \right\rceil mod\ 2$$

$$= \left\lfloor \frac{\left((-22x^7+21x^6+13x^5+37x^4-2x^3+37x^2+44x+34) mod (x^5+1)\right) mod\ 23}{\lfloor 23/2 \rfloor} \right\rceil mod\ 2$$

$$= \left\lfloor \frac{(37x^4-2x^3+59x^2+23x+21) mod\ 23}{\lfloor 23/2 \rfloor} \right\rceil mod\ 2$$

$$= \left\lfloor \frac{(14x^4+21x^3+13x^2+0x+21)}{\lfloor 23/2 \rfloor} \right\rceil mod\ 2$$

$$= \left\lfloor \frac{(14x^4+21x^3+13x^2+0x+21)}{12} \right\rceil mod\ 2$$

$$= \left|\left(\frac{14}{12}x^4 + \frac{21}{12}x^3 + \frac{13}{12}x^2 + \frac{0}{12}x + \frac{21}{12}\right)\right| mod\ 2$$

$$= (1x^4 + 2x^3 + 1x^2 + 0x + 2)\ mod\ 2$$

$$= (1x^4 + 0x^3 + 1x^2 + 0x + 0) = x^4 + x^2 = \text{The letter "T"}$$

Phantom Cryptosystem on the TI Graphing Calculator

Now let's use the Phantom Cryptosystem calculator program to encrypt the word "SWIFT". Once again the letters can be determined from Table 2. The 5- bit vectors for each letter is as follows:

$$S = p_1(x) = 1x^4 + 0x^3 + 0x^2 + 1x + 1 = x^4 + x + 1$$

$$W = p_2(x) = 1x^4 + 0x^3 + 1x^2 + 1x + 1 = x^4 + x^2 + x + 1$$

$$I = p_3(x) = 0x^4 + 1x^3 + 0x^2 + 0x + 1 = x^3 + 1$$

$$F = p_4(x) = 0x^4 + 0x^3 + 1x^2 + 1x + 0 = x^2 + x$$

$$T = p_5(x) = 1x^4 + 0x^3 + 1x^2 + 0x + 0 = x^4 + x^2$$

The Phantom cryptosystem utilizes three programs. The first program calculates the public key, the second program encrypts the plaintext, and the last program decrypts. Before running the program the necessary polynomials must be entered as vectors in the list storage spaces on the TI-84 Graphing Calculator. The polynomials can be entered into the list by the coefficients from largest degree to smallest degree. The following polynomials are required to be entered as lists for the public key program:

$$e(x) = x^2 + x$$

Modulus Polynomial: $x^5 + 1$

The image below shows each of the polynomials in vector form stored as lists:

Image 30: e(x) in list

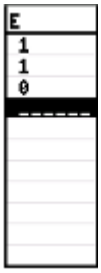


Image 31: Modulus Polynomial

Now we can run the program.

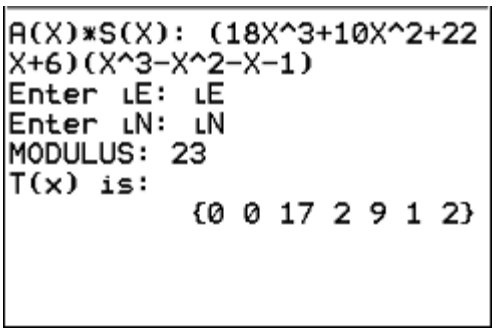
Remember that

$$a(x) = 18x^3 + 10x^2 + 22x + 6$$

$$s(x) = x^3 - x^2 - x - 1$$

First the program will ask us to input $A(x) * S(x)$. Then it will ask for List E, List N because that is where polynomial e and the modulus polynomial, x^5+1, are stored. Also we must enter in the modulus number, q = 23. Then the t(x) is displayed.

Image 32: t(x) display screen



After computing t(x), the public key can be published as (a(x), t(x)). This completes the public key calculator program.

The next program encrypts the plaintext, p(x). Before running the program, the following polynomials must be entered as lists:

Image 33: $e_1(x)$ in list

Image 34: $e_2(x)$ in list

Image 35: Modulus Polynomial

Image 36: $p_1(x)$ in list

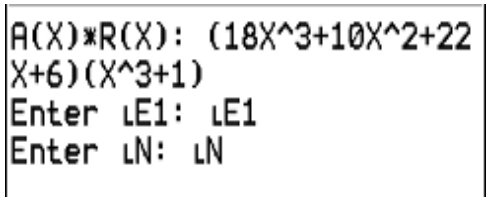
Then the program will ask us to input $A(x) * R(x)$, List E1, and List N.

Remember that

$$a(x) = 18x^3 + 10x^2 + 22x + 6$$

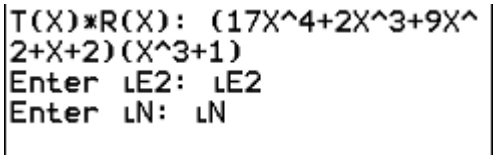
$$r(x) = x^3 + 1$$

Image 37: Input Screen



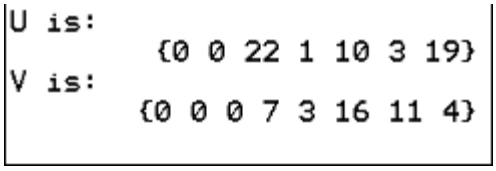
Then the program will display the T(x) list again so we can reenter the polynomial when the program asks for $T(x) * R(x)$, and then we need to enter List E2 and List N.

Image 38: Input Screen



This will display the u and v values, which is the ciphertext.

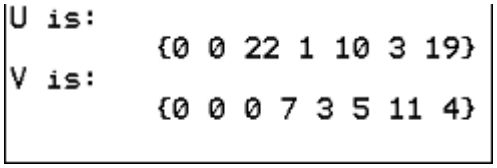
Image 39: u and v values



These lists can be put back into polynomial form by starting on the right with x^0 and moving to the left add one to the exponent. So $U(x) = 22x^4 + x^3 + 10x^2 + 3x + 19$ and $V(x) = 7x^4 + 3x^3 + 16x^2 + 11x + 4$. We can now rerun the program for the other plaintext letters.

W gives the following u and v values:

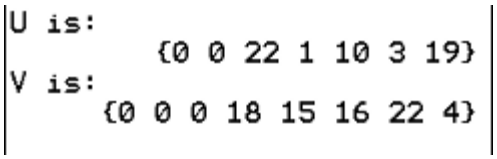
Image 40: u and v values



So $u(x) = 22x^4 + x^3 + 10x^2 + 3x + 19$ and $v(x) = 7x^4 + 3x^3 + 5x^2 + 11x + 4$.

I gives the following u and v values:

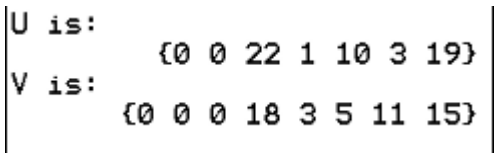
Image 41: u and v values



So $u(x) = 22x^4 + x^3 + 10x^2 + 3x + 19$ and $v(x) = 18x^4 + 15x^3 + 16x^2 + 22x + 4$.

F gives the following u and v values:

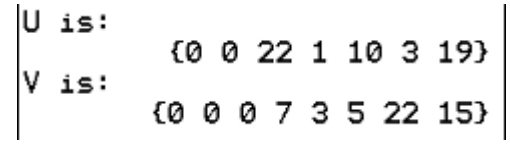
Image 42: u and v values



So $u(x) = 22x^4 + x^3 + 10x^2 + 3x + 19$ and $v(x) = 18x^4 + 3x^3 + 5x^2 + 11x + 15$.

T gives the following u and v values:

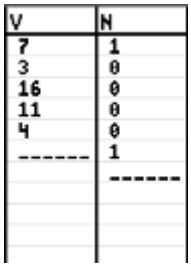
Image 43: u and v values



So $u(x) = 22x^4 + x^3 + 10x^2 + 3x + 19$ and $v(x) = 7x^4 + 3x^3 + 5x^2 + 22x + 15$.

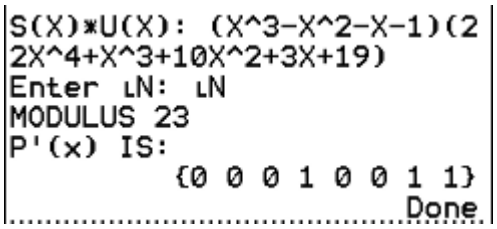
Now that we have the ciphertext, the decryption calculator program can be used to return to the plaintext message. Before running the program we must make sure that the first v value is entered into the List V storage space, we also will be using the modulus polynomial, x^5+1, which can be stored in the List N.

Image 44: List Storage



Now we can run the program, which will first ask for $S(x) * U(x)$. Then the program will require the input of List N (the modulus polynomial), and then the input of the modulus number.

Image 45: p'(x) value

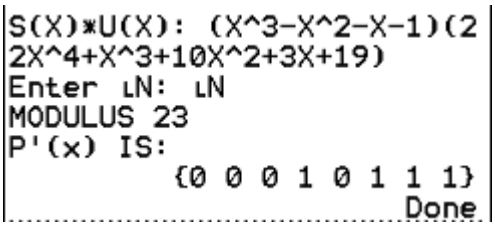


From this we can determine using Table 2 that P'(x) gives the letter S.

We can now change the v value stored in List V and rerun the program. This process can be repeated until all the ciphertext is decrypted. By doing this we get the following results:

For $v(x) = 7x^4 + 3x^3 + 5x^2 + 11x + 4$:

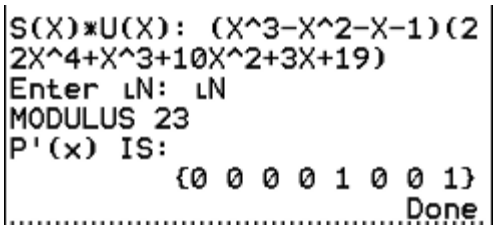
Image 46: p'(x) value



This p'(x) translates to the letter W.

For $v(x) = 18x^4 + 15x^3 + 16x^2 + 22x + 4$:

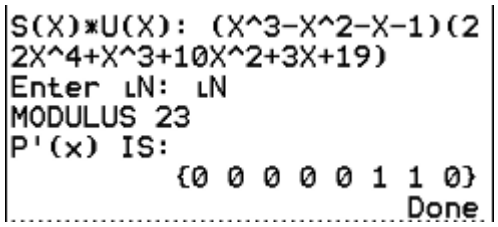
Image 47: p'(x) value



This p'(x) translates to the letter I.

For $v(x) = 18x^4 + 3x^3 + 5x^2 + 11x + 15$:

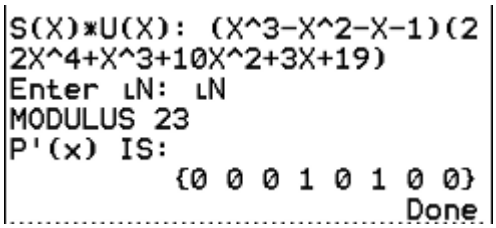
Image 48: p'(x) value



This p'(x) translates to the letter F.

For $v(x) = 7x^4 + 3x^3 + 5x^2 + 22x + 15$:

Image 49: p'(x) value



This p'(x) translates to the letter T.

If we put all the plaintext letters together, we get the word "SWIFT".

Conclusion

Once relevant quantum computers are built, working cryptosystems, like RSA, will no longer be secure. This provides a vulnerability for currently encrypted data; however it also opens the door for new possibilities and new cryptosystems to take over. NIST has been driving the search for these new cryptosystems, but the mathematics that are utilized are actually

accessible to undergraduate students. Not only is the mathematics behind these cryptosystems accessible, but they can also be used to demonstrate the power of the TI-84 graphing calculators. Although the cryptosystems that we utilized were only small scale, the TI graphing calculator was able to run a program that could accurately encrypt and decrypt. However, for real-world security these cryptosystems would require the length of p, which dictates the size of the matrices and the polynomials, to be at least 256 bits, instead of just 5 bits like the examples used in this paper. More interestingly, the large-scale versions could have real application in the cryptographical world and could be the key to securing future data. This paper demonstrated that post-quantum cryptography might just be the answer to the future security of data.

Appendix: The six programs for the Smeagol cryptosystem and the Phantom cryptosystem written with the TI Basic programming language.

Smeagol Public Key Program

```
Input "A ",[A]
Input "S ",[B]
Input "E ",[C]
dim([A])→L₁
Input "MODULUS ",N
[A]*[B]+[C]→[H]
1→K
For(I,1,L₁(1))
For(J,1,L₁(2))
[H](I,J)→L₂(K)
K+1→K
End
End
L₂-N*int(L₂/N)→L₃
[H]→[I]
1→L
For(I,1,L₁(1))
For(J,1,L₁(2))
L₃(L)→[I](I,J)
L+1→L
End
End
Disp "Public Matrix T: "
Disp [I]
```

Smeagol Encryption Program

```
Input "A ",[A]
Input "T ",[I]
Input "e1 ",[D]
Input "e2 ",[E]
Input "r ",[F]
Input "p ",[G]
Input "MODULUS ",N
[A]ᵀ*[F]+[D]→[H]
dim([H])→L₄
1→G
For(I,1,L₄(1))
For(J,1,L₄(2))
[H](I,J)→Ls(G)
G+1→G
End
End
Ls-N*int(Ls/N)→L₆
[H]→[J]
1→H
For(I,1,L₄(1))
For(J,1,L₄(2))
L₆(H)→[J](I,J)
H+1→H
End
End
[I]ᵀ*[F]+[E]+round(N/2,Ø)*[G]→[C]
dim([C])→L₄
1→E
For(I,1,L₄(1))
For(J,1,L₄(2))
[C](I,J)→Ls(E)
E+1→E
End
End
Ls-N*int(Ls/N)→L₆
1→F
For(I,1,L₄(1))
For(J,1,L₄(2))
L₆(F)→[H](I,J)
F+1→F
End
End
Disp " "
Disp "U IS ",[J]ᵀ
Disp " "
Disp "V IS ",[H]ᵀ
```

Smeagol Decryption Program

```
Input "u ",[J]
Input "v ",[H]
Input "S ",[B]
Input "Modulus ",N
1/(round(N/2,Ø))*([H]-[B]ᵀ[J])→[I]
round([I],Ø)→[I]
dim([I])→L₁
1→K
For(I,1,L₁(1))
For(J,1,L₁(2))
[I](I,J)→L₂(K)
K+1→K
End
End
L₂-2*int(L₂/2)→L₃
[I]→[E]
1→L
For(I,1,L₁(1))
For(J,1,L₁(2))
L₃(L)→[E](I,J)
L+1→L
End
End
Disp " "
Pause "p' is ",[E]
```

Phantom Public Key Program

```
ClrHome
Input "A(X)*S(X): ",Str0
If inString("+-",sub(Str0,1,1
Then
Disp "ERROR SIGN
Stop
End
0→dim(L2
L2→L3
L3→L4
L4→L5
0→L
" →Str9
1→I
1→U
While sub(Str0,I,1)≠"(
I+1→I
End
If I>1
Then
sub(Str0,1,I-1→Str1
If Str1="-
Str1+"1"→Str1
expr(Str1→U
End
For(I,I,length(Str0
sub(Str0,I,1→Str1
If Str1="(
I+1→J
If inString("ABCDEFGHIJKLMNOPQRSTUVWXYZθ",Str1
Then
If Str9="
Str1→Str9
If Str1≠Str9
Then
Disp "ERROR VARIABLES
Stop
End
If Str1≠"X
Then
"X→Str1
sub(Str0,1,I-1)+Str1+sub(Str0,I+1,length(Str0)-I→Str0
End
End
If inString("+-)",Str1
Then
expr(sub(Str0,J,I-J→L2(1+dim(L2
If sub(Str0,J-1,1)="-
-L2(dim(L2→L2(dim(L2
If not(inString(sub(Str0,J,I-J),"X
0→L3(1+dim(L3
I+1→J
If L
Then
I→M
While inString(sub(Str0,K,M-K),"*") or inString(sub(Str0,K,M-K),"/
M-1→M
End
expr(sub(Str0,K,M-K→L3(1+dim(L3
0→L
```

```
End
End
If Str1="X
Then
If not(inString("²³^",sub(Str0,I+1,1
1→L₃(1+dim(L₃
If sub(Str0,I+1,1)="²
2→L₃(1+dim(L₃
If sub(Str0,I+1,1)="³
3→L₃(1+dim(L₃
If sub(Str0,I+1,1)="^
Then
I+2→K
1→L
End
End
If Str1=")
Then
0→dim(L₁
1+max(L₃→dim(L₁
For(A,1,dim(L₂
L₂(A)+L₁(dim(L₁)-L₃(A→L₁(dim(L₁)-L₃(A
End
0→dim(L₂
L₂→L₃
If dim(L₄
Then
0→dim(L₅
dim(L₁)+dim(L₄)-1→dim(L₅
For(B,1,dim(L₁
For(C,1,dim(L₄
L₅(B+C-1)+L₁(B)L₄(C→L₅(B+C-1
End
End
L₅→L₄
Else
L₁→L₄
End
End
End
If not(dim(L₅
L₁→L₅
ULs→L₅

dim(L₅)→M
ClrList L₄
For(I,1,M)
0→L₄(I)
End
Input "Enter ιE: ",ιE
dim(ιE)→N
For(J,1,N)
ιE(N-J+1)→L₄(M-J+1)
End
L₅+L₄→ιR

Input "Enter ιN: ",L₂
dim(ιR)→M
dim(L₂)→N
ClrList L₃,L₄,L₅,L₆
```

```
ιR→L₆
For(I,1,M-N+1)
int(ιR(I)/L₂(1)→L₃(I)
L₃(I)*L₂→L₅
For(K,1,N)
L₆(I+K-1)-L₅(K)→L₆(I+K-1)
End
End

Input "MODULUS: ",Q
L₆-Q*int(L₆/Q)→ιT
Disp "T(x) is: "
Disp ιT
Pause
```

Phantom Encryption Program

```
DelVar L₁DelVar L₂DelVar L₃DelVar L₄DelVar L₅1→X
ClrHome
Input "A(X)*R(X): ",Str0
If inString("+-",sub(Str0,1,1
Then
Disp "ERROR SIGN
Stop
End
0→dim(L₂
L₂→L₃
L₃→L₄
L₄→L₅
0→L
" →Str9
1→I
1→U
While sub(Str0,I,1)≠"(
I+1→I
End
If I>1
Then
sub(Str0,1,I-1→Str1
If Str1="-
Str1+"1"→Str1
expr(Str1→U
End
For(I,I,length(Str0
sub(Str0,I,1→Str1
If Str1="(
I+1→J
If inString("ABCDEFGHIJKLMNOPQRSTUVWXYZθ",Str1
Then
If Str9="
Str1→Str9
If Str1≠Str9
Then
Disp "ERROR VARIABLES
Stop
End
If Str1≠"X
Then
"X→Str1
sub(Str0,1,I-1)+Str1+sub(Str0,I+1,length(Str0)-I→Str0
End
End
If inString("+-)",Str1
Then
expr(sub(Str0,J,I-J→L₂(1+dim(L₂
If sub(Str0,J-1,1)="-
-L₂(dim(L₂→L₂(dim(L₂
If not(inString(sub(Str0,J,I-J),"X
0→L₃(1+dim(L₃
I+1→J
If L
Then
I→M
While inString(sub(Str0,K,M-K),"*") or inString(sub(Str0,K,M-K),"/
M-1→M
End
expr(sub(Str0,K,M-K→L₃(1+dim(L₃
```

```
Ø→L
End
End
If Str1="X
Then
If not(inString("²³^",sub(StrØ,I+1,1
1→L₃(1+dim(L₃
If sub(StrØ,I+1,1)="²
2→L₃(1+dim(L₃
If sub(StrØ,I+1,1)="³
3→L₃(1+dim(L₃
If sub(StrØ,I+1,1)="^
Then
I+2→K
1→L
End
End
If Str1=")
Then
Ø→dim(L₁
1+max(L₃→dim(L₁
For(A,1,dim(L₂
L₂(A)+L₁(dim(L₁)-L₃(A→L₁(dim(L₁)-L₃(A
End
Ø→dim(L₂
L₂→L₃
If dim(L₄
Then
Ø→dim(L₅
dim(L₁)+dim(L₄)-1→dim(L₅
For(B,1,dim(L₁
For(C,1,dim(L₄
L₅(B+C-1)+L₁(B)L₄(C→L₅(B+C-1
End
End
L₅→L₄
Else
L₁→L₄
End
End
End
If not(dim(L₅
L₁→L₅
ULs→L₅

dim(L₅)→M
ClrList L₄
For(I,1,M)
Ø→L₄(I)
End
Input "Enter ιE1: ",ιE1
dim(ιE1)→N
For(J,1,N)
ιE1(N-J+1)→L₄(M-J+1)
End
L₅+L₄→ιS

Input "Enter ιN: ",L₂
dim(ιS)→M
dim(L₂)→N
```

```
ClrList L3,L4,L5,L6
ιS→L6
For(I,1,M-N+1)
int(ιS(I)/L2(1)→L3(I)
L3(I)*L2→L5
For(K,1,N)
L6(I+K-1)-L5(K)→L6(I+K-1)
End
End

L6-Q*int(L6/Q)→ιU

DelVar L1DelVar L2DelVar L3DelVar L4DelVar L51→X
ClrHome
Disp "T(x) is: "
Disp ιT
Input "T(X)*R(X): ",Str0
If inString("+-",sub(Str0,1,1
Then
Disp "ERROR SIGN
Stop
End
0→dim(L2
L2→L3
L3→L4
L4→L5
0→L
" →Str9
1→I
1→U
While sub(Str0,I,1)≠"(
I+1→I
End
If I>1
Then
sub(Str0,1,I-1→Str1
If Str1="-
Str1+"1"→Str1
expr(Str1→U
End
For(I,I,length(Str0
sub(Str0,I,1→Str1
If Str1="(
I+1→J
If inString("ABCDEFGHIJKLMNOPQRSTUVWXYZθ",Str1
Then
If Str9="
Str1→Str9
If Str1≠Str9
Then
Disp "ERROR VARIABLES
Stop
End
If Str1≠"X
Then
"X→Str1
sub(Str0,1,I-1)+Str1+sub(Str0,I+1,length(Str0)-I→Str0
End
End
If inString("+-)",Str1
```

```
Then
expr(sub(Str0,J,I-J→L2(1+dim(L2
If sub(Str0,J-1,1)="-
-L2(dim(L2→L2(dim(L2
If not(inString(sub(Str0,J,I-J),"X
0→L3(1+dim(L3
I+1→J
If L
Then
I→M
While inString(sub(Str0,K,M-K),"*") or inString(sub(Str0,K,M-K),"/
M-1→M
End
expr(sub(Str0,K,M-K→L3(1+dim(L3
0→L
End
End
If Str1="X
Then
If not(inString("²³^",sub(Str0,I+1,1
1→L3(1+dim(L3
If sub(Str0,I+1,1)="²
2→L3(1+dim(L3
If sub(Str0,I+1,1)="³
3→L3(1+dim(L3
If sub(Str0,I+1,1)="^
Then
I+2→K
1→L
End
End
If Str1=")
Then
0→dim(L1
1+max(L3→dim(L1
For(A,1,dim(L2
L2(A)+L1(dim(L1)-L3(A→L1(dim(L1)-L3(A
End
0→dim(L2
L2→L3
If dim(L4
Then
0→dim(L5
dim(L1)+dim(L4)-1→dim(L5
For(B,1,dim(L1
For(C,1,dim(L4
L5(B+C-1)+L1(B)L4(C→L5(B+C-1
End
End
L5→L4
Else
L1→L4
End
End
End
If not(dim(L5
L1→L5
UL5→L5

dim(L5)→M
```

```
ClrList L4
For(I,1,M)
Ø→L4(I)
End
Input "Enter ιE2: ",ιE2
dim(ιE2)→N
For(J,1,N)
ιE2(N-J+1)→L4(M-J+1)
End
L5+L4→ιQ
dim(ιQ)→S
For(I,1,S)
Ø→ιY(I)
End
(round(Q/2,Ø)*ιP)→ιV
dim(ιV)→T
For(J,1,T)
ιV(T-J+1)→ιY(S-J+1)
End
ιQ+ιY→ιZ

Input "Enter ιN: ",L2
dim(ιZ)→M
dim(L2)→N
ClrList L3,L4,L5,L6
ιZ→L6
For(I,1,M-N+1)
int(ιZ(I)/L2(1)→L3(I)
L3(I)*L2→L5
For(K,1,N)
L6(I+K-1)-L5(K)→L6(I+K-1)
End
End

L6-Q*int(L6/Q)→ιA
dim(ιA)→N
dim(ιZ)→M
For(I,1,M)
Ø→ιG(I)
End
For(J,1,N)
ιA(N-J+1)→ιG(M-J+1)
End
ClrList ιV
ιG→ιV
Disp "U is: "
Disp ιU
Disp "V is: "
Disp ιV
Pause
```

Phantom Decryption Program

```
DelVar L₁DelVar L₂DelVar L₃DelVar L₄DelVar L₅1→X
ClrHome
Input "S(X)*U(X): ",Str0
If inString("+-",sub(Str0,1,1
Then
Disp "ERROR SIGN
Stop
End
0→dim(L₂
L₂→L₃
L₃→L₄
L₄→L₅
0→L
" →Str9
1→I
1→U
While sub(Str0,I,1)≠"(
I+1→I
End
If I>1
Then
sub(Str0,1,I-1→Str1
If Str1="-
Str1+"1"→Str1
expr(Str1→U
End
For(I,I,length(Str0
sub(Str0,I,1→Str1
If Str1="(
I+1→J
If inString("ABCDEFGHIJKLMNOPQRSTUVWXYZθ",Str1
Then
If Str9="
Str1→Str9
If Str1≠Str9
Then
Disp "ERROR VARIABLES
Stop
End
If Str1≠"X
Then
"X→Str1
sub(Str0,1,I-1)+Str1+sub(Str0,I+1,length(Str0)-I→Str0
End
End
If inString("+-)",Str1
Then
expr(sub(Str0,J,I-J→L₂(1+dim(L₂
If sub(Str0,J-1,1)="-
-L₂(dim(L₂→L₂(dim(L₂
If not(inString(sub(Str0,J,I-J),"X
0→L₃(1+dim(L₃
I+1→J
If L
Then
I→M
While inString(sub(Str0,K,M-K),"*") or inString(sub(Str0,K,M-K),"/
M-1→M
End
expr(sub(Str0,K,M-K→L₃(1+dim(L₃
```

```
Ø→L
End
End
If Str1="X
Then
If not(inString("²³^",sub(StrØ,I+1,1
1→L₃(1+dim(L₃
If sub(StrØ,I+1,1)="²
2→L₃(1+dim(L₃
If sub(StrØ,I+1,1)="³
3→L₃(1+dim(L₃
If sub(StrØ,I+1,1)="^
Then
I+2→K
1→L
End
End
If Str1=")
Then
Ø→dim(L₁
1+max(L₃→dim(L₁
For(A,1,dim(L₂
L₂(A)+L₁(dim(L₁)-L₃(A→L₁(dim(L₁)-L₃(A
End
Ø→dim(L₂
L₂→L₃
If dim(L₄
Then
Ø→dim(L₅
dim(L₁)+dim(L₄)-1→dim(L₅
For(B,1,dim(L₁
For(C,1,dim(L₄
L₅(B+C-1)+L₁(B)L₄(C→L₅(B+C-1
End
End
L₅→L₄
Else
L₁→L₄
End
End
End
If not(dim(L₅
L₁→L₅
ULs→L₅

dim(L₅)→M
ClrList L₄
For(I,1,M)
Ø→L₄(I)
End
dim(ιV)→N
For(J,1,N)
ιV(N-J+1)→L₄(M-J+1)
End

L₄-L₅→ιB

Input "Enter ιN: ",L₂
dim(ιB)→M
dim(L₂)→N
```

```
ClrList L3,L4,L5,L6
ιB→L6
For(I,1,M-N+1)
int(ιB(I)/L2(1)→L3(I)
L3(I)*L2→L5
For(K,1,N)
L6(I+K-1)-L5(K)→L6(I+K-1)
End
End

Input "MODULUS ",Q
L6-Q*int(L6/Q)→ιC
ιC/(round(Q/2,Ø))→ιD
round(ιD,Ø)→ιD
ιD-2*int(ιD/2)→ιF
Disp "P'(x) IS: "
Disp ιF
```

References

Dudley, U. (2008). Elementary Number Theory (2nd ed.). Dover Publications Inc. https://www.google.com/books/edition/Elementary_Number_Theory/tr7SzBTsk1UC?hl=en&gbpv=1&dq=Underwood+dudley+elementary+number+theory+dudley+pdf&printsec=frontcover

Gallian, J. A. (2017). Contemporary Abstract Algebra (9th ed.). https://handoutset.com/wp-content/uploads/2022/07/Contemporary-Abstract-Algebra-Joseph-A.-Gallian.pdf

Gonzalez de la Torre, M. A., & Dios, A. Q. (2022). About the FrodoKEM Lattice-Based Algorithm.

Holden, J. (2017). *The Mathematics of Secrets: Cryptography from Caesar Ciphers to Digital Encryption*. Princeton University Press.

Holden, J. (2022). *RSA is Dead, Long Live PQC! Teaching Cryptography in the Quantum Era*. *Mathematical Association of America*.

Lewand, R. E. (2000). Cryptological Mathematics. The Mathematical Association of America.

Moore, G. E. (2006). Cramming more components onto integrated circuits, reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. IEEE Solid-State Circuits Society Newsletter, 11(3), 33–35. https://doi.org/10.1109/n-ssc.2006.4785860

Polynomial Multiplication. ticalc.org. (2019). https://www.ticalc.org/archives/files/fileinfo/458/45814.html

Singh, S. (2000). *The Code Book: The Science of Secrecy from Egypt to Quantum Cryptography*. Anchor Books.