International Journal of Computer and Information Sciences, Vol. 12, No. 4, 1983

Computationally Efficient Algorithms for a One-Time Pad Scheme

A. G. Akritas,[1] S. S. Iyengar,[2] and A. A. Rampuria[3]

Received November 1982; revised June 1983

The use of cryptography for data protection has received a great deal of attention in recent years. This paper presents computationally efficient algorithms for the implementation of a one-time pad scheme. The algorithms to encipher and decipher text were implemented on a **PDP-11** computer using the programming language **C**. To study the behavior of the keys used to encipher and decipher text, we used the chi-square method, and the test results of two runs are presented with some statistical analysis.

KEY WORDS: Cryptography; ciphers; security; algorithms; data protection; cryptology; one-time pads.

1. INTRODUCTION

The use of cryptography for data protection has received a great deal of attention in recent years (see, e.g., Refs. 7–10 and 17). In today's complex society, as the need for fast electronic communication has grown, so has the need to secure the information being communicated. Furthermore, interest in cryptography is expected to rise with increasing use of the electronic fund transfer system and other applications needing data security and protection.[9]

This paper presents efficient algorithms for the implementation of a one-time pad scheme and is organized as follows. Section 2 describes definitions,

[1] Department of Computer Science, University of Kansas, Lawrence, Kansas 66045. This author was partially supported by the General Research Fund (No. 3230-20-0038) of the University of Kansas.

[2] Department of Computer Science, Louisiana State University, Baton Rouge, Louisiana 70803.

[3] Department of Computer Science, University of Kansas, Lawrence, Kansas 66045.

notations, and preliminary results. Section 3 presents an overview of the proposed scheme. Section 4 describes the mathematical formulation of the scheme. Section 5 presents some empirical results obtained during the study. Section 6 describes computational algorithms of the scheme and Section 7 presents the conclusion of the study and some open questions.

2. DEFINITIONS, NOTATIONS, AND PREVIOUS RESULTS

In this section we review some terminology and some fundamental results concerning one-time pads.[1,2,4]

2.1. Cryptographic Functions

We define a cryptographic function to be one of the form $E = g(k, m)$, where for fixed k, the function $f_k(x) = g(k, x)$ is one-to-one; m is a string of bits (the message to be sent), k is the key, and E is the enciphered message. The key structure determines a sequence $(i_1, ..., i_s)$ and has the following transformation scheme:

$$\begin{aligned} E &= g(k, m) \\ E &= f_{i_1}(f_{i_2}(\dots f_{i_s}(m))\dots) \end{aligned} \tag{1}$$

Normally, one f_i is a mixing transformation. In order to decode the message, the key and the enciphered message are used to undo the transformations f_{i_j} one-by-one in reverse order. For more information on this refer to Refs. 16 and 18.

2.2. Polynomial Equation

In this paper, we define a real root x of a polynomial equation by a continued fraction of the following form:

$$x = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ddots \cfrac{1}{a_m + 1/y}}} \tag{2}$$

Where $a_1, a_2, \ldots, a_m$ are integers (partial quotients). The continued fraction can also be structured as follows:

$$x = \frac{P_m y + P_{m-1}}{Q_m y + Q_{m-1}} \tag{3}$$

where P_k/Q_k is the kth convergent to

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ddots}}$$

and

$$\begin{aligned} P_{k+1} &= a_{k+1} P_k + P_{k-1} \\ Q_{k+1} &= a_{k+1} Q_k + Q_{k-1} \end{aligned} \tag{4}$$

We refer to the right-hand side of Eq. (3) as the "expression for the root of the polynomial."

2.3. Floor Functions

Let x be a positive real number. F is called the floor function of x if

$$F(x) = I$$

where I is the greatest integer such that $I \leqslant x$; $F(x)$ is represented by $\lfloor x \rfloor$.

2.4. Key

The key is used to encipher–decipher text. Without the knowledge of the key the ciphertext "cannot" be deciphered.

2.5. One-Time Pad

This is a cryptographic scheme according to which the ith ciphertext character C_i is obtained by the formula $C_i = M_i + K_i (\text{mod } 26)$, where M_i is the ith plaintext character and K_i is the ith key character. Since the key is never repeated, one-time pads are unconditionally crypto secure; their main drawback, however, is the key management (for obvious reasons).

2.6. Partial Quotients

The integers $a_1, a_2, \ldots, a_m$ in Eq. (2) are called partial quotients.

2.7. Unconditionally Crypto Secure

This is a method for cryptography whose security is totally dependent on the knowledge of the keys used, and without the knowledge of which it is impossible to decipher the intercepted message.

2.8. Previous Results—Akritas' Approach

Akritas[4] proposed a one-time pad scheme where the key management does not present a problem. In this scheme, based on Vincent's theorem,[1,6,20] successive continued fraction transformations are used to isolate and approximate the single, irrational, positive root of a polynomial equation and the partial quotients themselves are used as the key. Thus the key is "concealed" in a polynomial equation that can be easily exchanged using the public key-distribution methods described in Ref. 14.

Before we get into any further discussion, we would like to present an example using Akritas' scheme[4] in order to gain some insight into the problem.

Let us consider "AIKBS" to be the text that party A wishes to communicate to party B. We call this the *plaintext*. Let A be in Washington, D.C., and B in Moscow. Party A does not want to send the plaintext as is, because anybody who intercepts it will also share the same information. So, he sends a different version of the text (ciphertext), from which B can easily retrieve (decipher) the original text (plaintext) by applying some predefined algorithm to obtain the key. Anybody else who intercepts the ciphertext would not be able to retrieve the plaintext without the knowledge of the key.

In our case the key is contained in a polynomial equation with one irrational root; this equation should be securely exchanged between A and B before commencement of communications.

Let the polynomial equation be

$$P_1(x) = x^3 - 2 = 0 \tag{5}$$

which has one sign variation in the sequence of its coefficients $1\ 0\ 0\ -2$. Party A does the following:

Step 1: He computes the floor function a_1 of the root of this polynomial; this turns out to be 1. This is the first partial quotient in the continued fraction expansion of the root

$$x = 1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ddots}}$$

Step 2: This a_1 is used as the first key to encipher the first character 'A' of the plaintext as follows:

$$\begin{aligned}\text{cipher symbol } C_1 &= (a_1 + \text{'A'}) \bmod 26 \\ &= (1 + \text{'A'}) \bmod 26 \\ &= \text{'B'}\end{aligned} \tag{6}$$

Step 3: The polynomial $P_1(x)$ is transformed to $P_2(x) \leftarrow P_1(1 + 1/x)$, where after computations we have

$$P_2(x) = -x^3 + 3x^2 + 3x + 1 \tag{7}$$

(The computations involved are explained in Section 4.)

Now, Steps 1–3 are repeated with polynomial $P_2(x)$ in place of $P_1(x)$, and the second character in the plaintext T, this gives us

$$\begin{aligned}a_2 &= 3 \\ C_2 &= \text{'L'} \\ P_3(x) &= 10x^3 - 6x^2 - 6x - 1\end{aligned} \tag{8}$$

By repeating Steps1–3 for each character in the plaintext, we obtain the following:

$$\begin{aligned}&a_3 = 1, \quad && C_3 = \text{'L'}, \quad && P_4(x) = -3x^3 + 12x^2 + 24x + 10 \\ &a_4 = 5, && C_4 = \text{'G'}, && P_5(x) = 55x^3 + 81x^2 + 33x - 3 \\ &a_5 = 1, && C_5 = \text{'T'} &&\end{aligned} \tag{9}$$

Hence, the ciphertext is 'BLLGT' and this is sent to B over an insecure channel.

Now B, who receives this ciphertext, can easily decipher it, starting with the polynomial $P_1(x)$. He performs Steps 1 and 3 exactly as mentioned above. However, Step 2 is modified so that

$$\text{plaintext symbol } P_i = (C_i - a_i) \bmod 26$$

Thus, the original plaintext 'AIKBS' will be recoved.

In the above example, we can see that, after each execution of Step 3, the coefficients of the polynomial are becoming large (in absolute value) and also that out of the five partial quotients, a_i's, three were one. So most of the time the key used to encipher or decipher was one.[13] Hence, to better conceal our message, these keys sould be made more random.

In the present study, we propose some modifications to the above scheme to keep the coefficients within one word of memory and to make the keys as random as possible. Several possibilities are considered and analyzed. Finally, an algorithm is presented that does accomplish our requirements. To do so, we have used the plaintext itself as part of our coding scheme. However, this does not make the system less secure.

3. AN OVERVIEW OF THE PROPOSED SCHEME

In this section we present an overview of our scheme and the mathematics involved. The ideas are further explained and analyzed in full detail with empirical results in subsequent sections.

3.1. One-time Pads in Cryptography

As we saw above, one-time pads are unconditionally crypto secure, i.e., since we use a different key to encipher each character in the plaintext, it is impossible to recover the plaintext without the knowledge of the whole key. For example, consider the scheme $C_i = (M_i + K_i) \bmod 26$, where the ith ciphertext symbol C_i is obtained by adding mod 26 the ith message symbol M_i and the ith key symbol K_i. Clearly, without knowledge of the K_i's it is impossible to recover the M_i's. However, one is faced with the difficult task of generating and distributing enormous amounts of key, an operation that renders the scheme very expensive to use.

3.2. Use of a Polynomial Equation to Generate the Key

A polynomial equation with one sign variation in the sequence of its rational coefficients has exactly one positive root; choose the equation so that the root is a nonquadratic irrational number (this will guarantee that the sequence of partial quotients is never repeated). Akritas, based on Vincent's theorem,[1] uses continued fractions and the idea that each partial quotient is the floot function of the positive root of a polynomial to approximate the real root. As this root can be approximated to any degree of tolerance, and the partial quotients do not repeat, theoretically, we can get an infinite number of them, which can be used as the key. However, we have the following interesting fact.

3.3. Behavior of the Partial Quotients

For almost all real numbers, the probability that the nth partial quotient a_n in the continued fraction expansion of a real number is equal to a positive integer j is given by

$$\log_2 \frac{(j+1)^2}{j(j+2)}$$

For $j = 1$ and almost all numbers, this means that the probability that $a_n = 1$ is approximately 0.41.[13] Therefore, in order to better conceal message, our scheme has to be modified. Instead of using the partial quotients themselves as the key, we now use the number obtained from exclusive-ORing mod p the denominator and numberator of the updated expression for the root of the equation. That is, after the ith transformation of the polynomial equation has been performed, the root is represented by

$$x = n_i/d_i, \qquad \text{where} \quad n_i = P_m y + P_{m-1} \quad \text{and} \quad d_i = Q_m y + Q_{m-1} \tag{10}$$

we then choose $K_i = (n_i \text{ XOR } d_i) \bmod p$ to be our key. This scheme is explained in detail below (see Section 3.6).

3.4. Coefficients of the Transformed Polynomial Equation

In the above-mentioned scheme, to approximate the real root, the polynomial equation $P(x) = 0$ goes through successive transformations of the form $P(x) \leftarrow P(b + 1/x)$, which is equivalent to the pair of transformations $P(x) \leftarrow (b + x)$ and $P(x) \leftarrow P(1/x)$, where b is some partial quotient a_n. In this process, the coefficients of the polynomial start to get large, and eventually it becomes impossible to store them in one computer word of memory. As we intend to implement our method on a reasonably small system, we decided to use modular arithmetic, thus guaranteeing that the coefficients can always be stored in one word of memory (i.e., after each transformation we make sure that the coefficients stay within a given range). By modifying the polynomial in the above manner after each transformation, we found that the polynomial repeats itself after a certain number of transformations and so does our key. The cycle length of the partial quotients was found to be an integer multiple of the cycle length of the polynomial. So, we need some further modifications.

3.5. Use of the Plaintext Itself as Key

The plaintext itself can be (and has been) used to generate the key. For example, consider the scheme $C_i = (M_{i-1} + M_i) \bmod 26$, where the M_{i-1}th character is used as the key to encipher the M_ith character in the plaintext. Initially we have $C_0 = (M_0 + K) \bmod 26$, where K is some constant. So if we know K, we can decipher the ciphertext very easily. But, clearly this kind of scheme, which only depends on the plaintext, is very insecure, since K could easily be found by applying all the integers in the range 0–25, and one can easily decipher any message. We use this idea in the following way.

3.6. A Hybrid Method

In our method, we start out with the polynomial equation, and proceed as explained before, but instead of using $K_i = (n_i \text{ XOR } d_i) \bmod p$, we modify it to be

$$K_i = (n_i + m_{i-1}) \text{ XOR} (d_i + m_{i-1}) \bmod p$$

where n_i, d_i, and p are as before and m_{i-1} is the $(i-1)$th character in the plaintext. Initially we can have m_0 be any constant.

Thus, our scheme is secure, because the K_i's cannot be known until the n_i's and d_i's are known or until the polynomial itself is known. Also, the K_i's do not produce any cycle because the M_i's do not. (In a future work we hope to examine the problem of plaintext attack.)

4. MATHEMATICAL STRUCTURE—BACKGROUND

In this section we present in detail the various steps involved in our procedure. We start with presenting the main theorem used to isolate and approximate the real roots of a polynomial equation. The proof of this theorem is quite lengthy, and since it appears in the literature, it will be omitted.

Theorem 4.1. (Vincent–Uspensky–Akritas): Let $P(x) = 0$ be a polynomial equation of degree $n > 1$, with rational coefficients and without multiple roots, and let $\Delta > 0$ be the smallest distance between any two of its roots. Let m be the smallest index such that

$$F_{m-1}\Delta/2 > 1 \qquad \text{and} \qquad F_{m-1}F_m\Delta > 1 + 1/\varepsilon_n$$

where F_k is the kth member of the Fibonacci sequence

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21, \ldots$$

and

$$\varepsilon_n = (1 + 1/n)^{1/(n-1)} - 1$$

Then the transformation [Eq. (2)]

$$x = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \ddots \cfrac{1}{a_m + 1/y}}}}$$

(which is equivalent to the series of successive transformations of the form $x = a_i + 1/y$, $i = 1, 2, \ldots, m$) with arbitrary, positive integral elements $a_1, a_2, \ldots, a_m$, transforms the equation $P(x) = 0$ into the equation $P(y) = 0$, which has not more than one sign variation in the sequence of its coefficients.

The above theorem is applied as follows:

(i) The continued fraction transformation (2) can also be written as [Eq. (3)]

$$x = \frac{P_m y + P_{m-1}}{Q_m y + Q_{m-1}}$$

where P_k/Q_k is the kth convergent to the continued fraction

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + 1 \ddots}}$$

and, as we recall [Eq. (4)]

$$P_{k+1} = a_{k+1} P_k + P_{k-1}$$

$$Q_{k+1} = a_{k+1} Q_k + Q_{k-1}$$

(ii) Provided there are positive roots, when the partial quotients a_i's are properly chosen, $P(x)$ leads to an equation $P(y) = 0$ with exactly one sign variation in the sequence of its coefficients. Then from the Cardano–Descartes rule of signs we know that $P(y) = 0$ has one root in the interval $(0, \infty)$. If y was this positive root, then the corresponding root x of $P(x)$ could be easily obtained from Eq. (3). We only know that y lies in the interval $(0, \infty)$; therefore, substituting y in Eq. (3) once by 0 and once by ∞, we obtain for the positive root x an isolating interval whose unordered end points are P_{m-1}/Q_{m-1} and P_m/Q_m. Note that to each positive root there corresponds a different continued fraction. At most m partial quotients have to be computed for the isolation of any positive root. (Negative roots can be isolated if we replace x by $-x$ in the original equation.)

The calculation of the quantities a_i for the transformation of the form (2) that leads to an equation with exactly one sign variation constitutes the real root isolation procedure. Two methods actually result, Vincent's and one due to Akritas, corresponding to two different ways in which the computation of the a_i's may be performed.

Vincent's method basically consists in computing a particular a_i by a series of unit incrementations; that is, $a_i \leftarrow a_i + 1$, which corresponds to the substitution $x \leftarrow x + 1$. This "brute force" approach results in a method with an exponential behavior. Therefore, Vincent's method is of little practical use.

On the contrary, in the method developed by Akritas a partial quotient is immediately computed as the lower bound b of a positive root of a polynomial[3]; that is, $a_i \leftarrow b$, which corresponds to the substitution $x \leftarrow x + b$ performed on the polynomial under consideration at that stage. It is obvious that this method is independent of the size of the a_i's and results in a method with polynomial computing time bound. Akritas used Cauchy's rule repeatedly to find the lower bound b on the value of positive root. (One can safely conclude that the fllor function of a root is equal to its lower bound b).

In approximating the root the efficiency of the computation of the lower bound can be improved if instead of Cauchy's rule we use the following theorem and its corollary.

Theorem 4.2. If each negative coefficient of a polynomial is taken positively and divided by the sum of all the positive coefficients that precede it, the greatest of all the fractions thus formed increased by unity is an upper bound of the positive roots.

Corollary 4.1. If the polynomial equation has only one sign variation in the sequence of its (positive) coefficients and is represented by

$$P(x) = a_n x^n + \cdots + a_{r+1} x^{r+1} - a_r x^r \cdots - a_0$$

then an upper bound of the positive root is given by

$$b = \left[\max_{0 \leqslant j \leqslant r} \operatorname{abs}(a_j) \Big/ \sum_{i=r+1}^{n} a_i \right] + 1$$

The proof of this corollary is obvious from the above theorem because the denominator is constant. So, for our case, that is, where the polynomial has one sign variation, this is clearly an efficient way to compute the upper bound of the root.

Ng[15] used the bisection method to find the floor function of the root. We chose to use the hybrid false position method,[16] presented below, to do the same thing because the rate of convergence is much faster than that of the bisection method.

4.1. The Hybrid False Position Method for Computing the Floor Function

Let $P(x) = 0$ be the polynomial equation with one sign variation. We start with two values x_0^+ and x_0^-, such that $P(x_0^+) > 0$ and $P(x_0^-) < 0$. At the $(i+1)$th iteration the algorithm takes the interval endpoints x_i^+ and x_i^- and computes a new endpoint x_{i+1} as the zero of the line going through $P(x_i^+)$ and $P(x_i^-)$ as follows:

$$x_{i+1} = \frac{x_i^+ P(x_i^-) - x_i^- P(x_i^+)}{P(x_i^-) - P(x_i^+)}$$

If $P(x_{i+1}) > 0$, then $x_{i+1}^+ \leftarrow x_{i+1}$ and $x_{i+1}^- \leftarrow x_i^-$, otherwise, $x_{i+1}^- \leftarrow x_{i+1}$ and $x_{i+1}^+ \leftarrow x_i^+$. Thus, the interval $[x_i^+, x_i^-]$ keeps reducing and the method converges. We keep computing the new iterate x_i until $\text{abs}(\lfloor x_i^+ \rfloor - \lfloor x_i^- \rfloor)$ becomes less than or equal to one. Also, at the end of each iterative step, replace the value of $P(x_i^-)$ or $P(x_i^+)$, whichever is not changed, to K times its value, where K is some constant, $0 < K < 1$. By doing so, the speed of convergence is increased. We have chosen K to be $1/2$.

Let $[x_i^-, x_i^+]$ be the interval at the end of this iterative process. In our case x_i^- and x_i^+ are both positive, since we start with the interval $(0, \infty)$. So if $P(\lfloor x_i^+ \rfloor) > 0$, $\lfloor x_i^- \rfloor$ is the lower bound; otherwise, $\lfloor x_i^+ \rfloor$ is the lower bound.

As an example, let us consider the following polynomial $P(x)$ with one sign variation[19]:

$$x^3 - 2x - 5 = 0$$

The upper bound of the root of $P(x)$ is obtained using Corollary 4.1; it is

$$U = \frac{\max(2, 5)}{1} + 1 = 6$$

To compute the floor function (or the lower bound) of the root of $P(x)$ we start with the interval $[0, 6]$ and denote $x_0^- = 0$ and $x_0^+ = 6$.

We have

$$P(0) = -5$$

$$P(6) = 199$$

$$x_1 = \frac{6 * (-5) - 0 * 199}{-5 - 199}$$

$$= \frac{-30}{-204} = 0.147$$

$$P(0.147) = -5.29 < 0$$

Hence, now the interval is $[0.147, 6]$. Since the endpoint 6 is not changed, the value of $P(6)$ is taken to be $K * P(6)$, where K is chosen to be $1/2$. Hence

$$P(6) = \frac{199}{2} = 99.5$$

$$x_2 = \frac{6 * (-5.29) - 0.147 * 99.5}{-5.29 - 99.5}$$

$$= 3.13$$

$$P(3.13) = 19.4$$

Now, the interval becomes $[0.147, 3.13]$. Next we have

$$x_3 = \frac{3.13 * (-5.29) - 0.147 * 19.4}{-5.29 - 19.4}$$

$$= 0.7857$$

$$P(0.7857) = -1.72$$

So the interval is $[0.785, 3.13]$ and $P(3.13)$ becomes $19.4/2 = 9.7$. Next

$$x_4 = \frac{3.13 * (-1.72) - 0.785 * 9.7}{-1.72 - 9.7}$$

$$= 1.13$$

$$P(1.13) = -0.581$$

So, the interval becomes $[1.13, 3.13]$ and $P(3.13)$ becomes $9.7/2 = 4.8$. Next

$$x_5 = \frac{3.13 * (-5.81) - 1.13 * (4.8)}{-5.81 - 4.8}$$

$$= 2.22$$

$$P(2.22) = 1.50$$

So, the interval becomes $[1.13, 2.22]$. Now, $\lfloor 2.22 \rfloor - \lfloor 1.13 \rfloor = 2 - 1 = 1$, and we are ready to decide whether 2 is the lower bound; otherwise, 1 will be chosen. We have

$$P(2) = -1 \qquad \text{and} \qquad P(2.22) = 1.50$$

This implies that the root is in the interval $[2, 2.22]$. Hence 2 is the floor function.

4.2. A Method for the Translation of a Polynomial

As we have seen, we need to perform the following transformation:

$$P(x) \leftarrow P(b + 1/x)$$

The above is equivalent to the following pair:

$$P(x) \leftarrow P(b + x) \qquad \text{and} \qquad P(x) \leftarrow P(1/x)$$

The second transformation can be easily done by simply inverting the order of the coefficients of the polynomial, and so we are mainly concerned about the first one. We have used the Ruffini–Horner method[5]; a straight-line algorithm corresponding to it is the following:

Assume that $P(x) = \sum_{i=0}^{n-1} a_i x^i$. Now,

$$C_{j_0} \leftarrow a_{n-1}$$

$$C_{0,j+1} \leftarrow C_{0,j} * b + a_{n-1-(j+1)} \qquad (j = 0, 1, \ldots, n-2)$$

$$C_{k,j} \leftarrow C_{k,j-1} * b + C_{k-1,j} \qquad (k = 1, 2, \ldots, n-2;\ j = 1, 2, \ldots, n-1-k)$$

Example 1. Translation using the Ruffini–Horner method. Let us take the polynomial

$$P(x) = 3x^3 - 12x^2 - 24x - 10 = 0$$

and suppose that we want to compute

$$P(5 + 1/x)$$

First we do the transformation $P(x) \leftarrow P(x + 5)$:

	3	−12	−24	−10
5	3	3	−9	−55
5	3	18	81	
5	3	33		
5	3			

and we obtain

$$P(x) \leftarrow P(x+5) = 3x^3 + 33x^2 + 81x - 55$$

Now we do the second transformation

$$P(x) \leftarrow P(1/x)$$

which is achieved by inverting the order of the coefficients of the equation, i.e.,

$$P(x) \leftarrow P(1/x) = -55x^3 + 81x^2 + 33x + 3$$

5. EMPIRICAL RESULTS

In this section, we present some of the results obtained during the study. Since it is not possible to include in this report all of the results obtained, we have decided to present a few selected cases.

In Section 2 we mentioned that 41% of the time the partial quotient $a_n = 1$; for example, consider the polynomial equation

$$x^4 - 8x^3 - 3x^2 - 32x - 8 = 0$$

The partial quotients are

$$(11, 7, 3, 2, 1, 1, 1, 1, 20, 5, 11, 1, 7)$$

and the approximating interval of the root is

$$[0.92387953251128634045, 0.92387953251128676332]$$

So, out of 13 partial quotients five are ones, which is 35.71% of the time.

As we mention before, to better conceal the messages, we decided not to use the a_i's as our key, but instead to use the numerator and denominator of the expression [Eq. (10)] for the real root of a polynomial along with the message itself to generate our key.

From here on, we will refer to the above-mentioned numerator as N and the denominator as D. The root is denoted as x, and by p we denote the integer modulo with which we reduce the coefficients of the polynomial, the partial quotients, and N and D.

We used bitwise inclusive-OR, exclusive-OR, and AND of N and D and tried the resulting numbers as our key. We found that bitwise exclusive-OR gives us the best results; the keys obtained were quite random. Some results are given in Table I.[18]

For case 1 in Table I we find that 6.8% is the maximum and 1.2% the minimum number of times any value occurs. Certainly 6.8% is much better when compared to 41%. For case 2, 6.6% is the maximum and 1.2% the minimum number of occurrences, which is again very good. In case 3 we find that the maximum number of occurrences of the value 1 is 21.2%, much higher than in cases 1 and 2, but still better than 41%. Also, in practice, we will be using a larger value of p.

We have mentioned that the polynomials and our key repeat after a certain number of successive translations. Table II gives results with different values of p and different polynomials.

Table I. Values of Exclusive-OR of N and D[a]

Value of N XOR D	No. of occurrences	Percent of times
Case 1. Polynomial equation: $x^2 - 7x - 12 = 0$; modulo $p = 29$		
0	27	5.4
1	34	6.8
2	22	4.4
3	28	5.6
4	20	4.0
5	12	2.4
6	12	2.4
7	24	4.8
8	6	1.2
9	21	4.2
10	16	3.2
11	7	1.4
12	25	5.0
13	15	3.0
14	15	3.0
15	11	2.2
16	17	3.4
17	19	3.8
18	16	3.2
19	11	2.2
20	17	3.2
21	25	5.0
22	18	3.6
23	9	3.8
24	10	2.0
25	14	3.6
26	17	3.4
27	7	1.4
28	26	5.2

Table I. *(continued)*

Value of N XOR D	No. of occurrences	Percent of times
Case 2. Polynomial equation: $x^4 + 23x^3 - 5x^2 - 14x - 4 = 0$; modulo $p = 29$		
0	33	6.6
1	30	6.0
2	22	4.4
3	6	6.0
4	22	4.4
5	24	4.8
6	12	2.4
7	16	3.2
8	8	1.6
9	6	1.2
10	17	3.4
11	8	1.6
12	19	3.8
13	7	1.4
14	15	3.0
15	8	1.6
16	21	4.2
17	25	5.0
18	14	2.8
19	16	3.2
20	19	3.8
21	14	3.8
22	11	2.2
23	17	2.4
24	26	5.2
25	31	6.2
26	14	2.8
27	20	4.0
28	20	4.0
Case 3. Polynomial equation: $x^3 - 2 = 0$; modulo $p = 10$		
0	49	9.8
1	106	21.2
2	36	7.2
3	36	7.2
4	70	14.0
5	94	18.8
6	12	2.4
7	62	12.4
8	12	2.4
9	24	4.8

[a] Without taking the message into consideration. For each of cases 1–3 the number of polynomial updates is 500.

Table II[a]. Study of Cycle Lengths

Value of p	Polynomial cycle length	K	Key cycle length	Polynomial at which repetition starts
Case 1. Polynomial equation: $x^4 + 23x^3 - 5x^2 - 14x - 4 = 0$				
29	152	2	304	37th
26	2	1	2	89th
19	59	8	472	10th
14	12	1	12	170th
11	50	2	100	100th
5	10	4	40	2nd
Case 2. Polynomial equation: $x^3 + 8x^2 - 16x - 24 = 0$				
29	74	1	74	124th
26	2	1	2	15th
19	78	3	234	98th
14	2	3	6	5th
11	6	3	18	25th
5	2	3	6	3rd
Case 3. Polynomial equation: $x^3 - 2 = 0$				
29	74	1	74	144th
26	54	3	162	74th
19	102	1	102	1st
14	52	3	156	6th
11	2	1	2	38th
5	8	3	24	3rd
Case 4. Polynomial equation: $x^2 + 8x - 12 = 0$				
4	14	4	56	2nd
4	7	4	28	2nd
4	9	4	36	2nd
4	7	4	28	2nd
1	8	1	8	2nd
3	12	3	36	2nd
Case 5. Polynomial equation: $16x^4 + 8x^3 - 14x^2 - 24x - 4 = 0$				
29	196	Large, >500	Large, >500	49th
26	36	4	144	15th
19	2	1	2	58th
14	40	2	80	24th
11	24	4	96	65th
5	36	2	72	22nd

[a] K is the integer obtained by dividing the key cycle length by the polynomial cycle length.

6. PROPOSED SCHEME TO ENCIPHER AND DECIPHER. ALGORITHMS AND IMPLEMENTATION

In this section, we present our complete method to encipher and decipher text together with implementation details. Algorithms for the implementation of our method are presented in the Appendix.

6.1. Selection of the Modulus p

We choose an integer p such that, for any integer x, the following conditions are satisfied:

$$x \leqslant 2^p, \qquad x * x + x \leqslant 2^N - 1$$

where N is the number of bits in a computer word. We do so to ensure that no overflow occurs during our computation. This is necessary in order to produce the same results more than once, so that the ciphertext could correctly be deciphered.

For our computer, **PDP-11**, $N = 16$. Hence,

$$x * x + x \leqslant 2^{15} - 1$$

or

$$2^p * 2^p + 2^p \leqslant 2^{15} - 1$$

or

$$p \leqslant 7$$

or

$$x \leqslant 2^7 = 128$$

6.2. Selection of the Character Set

Obviously, the character set should consist of all the characters that could be used in the plaintext.

In our implementation, we chose two sets: one with 29 characters and another, the ASCII character set, with 128 characters; 128 happens to be the same number as the maximum value of x. We cannot choose a character set consisting of more than $2^7 = 128$ characters because then we will run out of distinct integers to represent them (for our specific computer).

The integer value we use to represent any character is the decimal value of the binary representation of that character according to ASCII standards.

6.3. Description of Our Method

Now, we present our scheme, which is divided into two sections, dealing with the method to encipher and the method to decipher. A polynomial equation with one sign variation (and hence one positive root) and a constant M_0 is exchanged between the communicating parties. If the root of the polynomial is not a nonquadratic irrational number, during the course of the transformations of the polynomial, some coefficients become 0 (first or last). To eliminate this difficulty and to always keep one sign variation in the sequence of the coefficients, we replace this 0 by 1 or -1, as the case may be. (The reader should notice that we have parted from our original assumption of using only nonquadratic irrational numbers as the roots of our polynomials.)

6.3.1. To Encipher

To encipher a plaintext the following steps are involved:

Step 1: The polynomial equation (coefficients only) is read so that it can be used for the generation of the key; also, the expression (N/D) for the root of the polynomial is initialized.

Step 2: Each coefficient C_i of the polynomial is reduced mod p, C_i mod p. If the leading coefficient is negative, all the coefficients are multiplied times -1.

Step 3: The upper bound of the root of the polynomial is computed using Corollary 4.1.

Step 4: The floor function of the root is computed using the hybrid false position method.

Step 5: The expression for the root of the polynomial (N/D) is updated.

Step 6: The next plaintext character M_i is read.

Step 7: The value $K_i = [(N + M_{i-1}) \text{ XOR} (D + M_{i-1})] \bmod p$ is computed, where N and D are obtained from Step 5.

Step 8: The ith ciphertext character is computed, $C_i = (K_i + M_i) \bmod p$.

Step 9: $i \leftarrow i + 1$.

Step 10: Steps 2–9 are repeated until no more characters are left in the plaintext.

6.3.2. To Decipher

To decipher a ciphertext, we basically follow the same steps as above, except for Steps 6–8 and 10, which need to be modified as follows:

Step 6′: The next ciphertext character C_i is read.

Step 7′: From M_{i-1}, the $(i-1)$th plaintext character, computed in the previous cycle, the value of $K_i = [(N + M_{i-1})\,\mathrm{XOR}(D + M_{i-1})]$ is computed.

Step 8′: The ith plaintext character $M_i = (C_i - K_i) \bmod p$ is obtained.

Step 10′: Steps 2–9 are repeated until no more characters are left in the ciphertext.

6.4. Implementation of the Algorithm

The algorithms presented in the previous section were implemented on a **PDP-11** computer using the programming language **C** (see Ref. 11). A listing of programs appears in the Appendix.

In this section, results of two test runs are presented with analysis.[18] To study the behavior of the key obtained, we use the chi-square method,[12] which is briefly summarized below:

The chi-square statistic V of observed quantities is given by

$$V = \frac{1}{n} \sum_{1 \leqslant s \leqslant k} \left(\frac{Y_s^2}{P_s} \right) - n$$

where n is the number of independent observations, P_s is the probability that an observation falls into category s, Y_s is the number of observations actually falling into category s, k is the number of different categories, and V, the randomness of the observations (of the keys in our case), can be obtained from the chi-square distribution Table.[12]

Test 1

Plaintext Used to Encipher

this is a sample text written with a character set of twenty-nine characters. all lower case alphabet and . and new line character. this is stored in fil ptext.c and ciphertext will be stored in ctext.c lets include some characters abcdefghijklmnopqrstuvwxyz.

Polynomail Used for the Key

$$P(x) = x^3 - 2 = 0$$

A Constant Cons Used as the Initial Character of Plaintext to Encipher–decipher

$$\text{Cons} = 0$$

Character Set Used

We used a character set of 29 characters for this test. The character set was [a–z, dot, blank, new line character]. The key generated by the polynomial (with the help, of course, of the plaintext to be enciphered) was as follows (in groups of five):

5 1 28 17 2	27 27 3 23 4	24 12 9 20 8	28 24 7 0 9
1 1 10 13 19	22 6 12 2 14	2 28 0 17 27	11 13 6 16 0
13 10 10 24 2	25 14 25 19 5	5 6 8 25 5	2 15 1 1 4
7 20 22 8 2	3 23 9 2 10	21 13 23 5 23	2 8 26 25 4
0 21 19 5 23	14 2 13 27 1	14 19 28 27 24	6 21 2 13 4
5 19 19 11 16	12 12 9 0 27	11 11 2 23 21	1 18 1 23 0
1 13 20 8 14	24 3 18 18 1	12 12 20 24 15	1 14 5 16 19
5 4 3 1 2	1 7 12 26 2	14 17 11 19 0	28 23 27 13 11
4 15 15 15 1	8 1 17 0 5	8 26 15 14 26	7 19 22 1 20
10 28 3 2 9	23 19 13 24 1	4 0 3 24 9 2	2 15 19 13 13
28 24 20 3 4	17 17 18 12 25	5 16 16 3 4	5 1 4 3 13
4 26 2 14 0	27 22 0 23 20	0 26 15 4 11	7 9 3 8 1
17 5 2 24 6	20 26 26 15 16	21 14 23 25 1	0 6 1 23 19
3 27 0 12 6	16 0		

Count of Occurrence of Each Key

Note that count j means number of occurrence of key j:

count0 =	16	count15 =	9
count1 =	21	count16 =	7
count2 =	18	count17 =	8
count3 =	11	count18 =	4
count4 =	12	count19 =	13
count5 =	14	count20 =	9
count6 =	7	count21 =	5
count7 =	5	count22 =	3
count8 =	8	count23 =	13
count9 =	6	count24 =	10
count10 =	5	count25 =	6
count11 =	7	count26 =	8
count12 =	9	count27 =	9
count13 =	12	count28 =	7
count14 =	10		

The ciphertext obtained is displayed below (in groups of five; note that an asterisk represents new line character):

```
eugzu   .gvwe   xbjdx   k gtn   yujgh   bzcg*   bvihf   qlmix
abkmo   gnnox   yeuny   yytou    ge v   gcvsb   m nlf   zvmob
 vbqw   qihgb   .doid   .yrhj   nfjjs   larlg*  yobhz   xrmcn
fmwpo   mduif   ajsxo   uvnfs   nwctv   pyqab   wbkyi   k*. b
ijfmd   hbbse   kfbvb   ysmfo   a*zku   fso a   wtrmn   fgxdm
ps ow   btqpk   yqugv   dnogo   ehbbd   ok. z   *aopg   bl.vh
bshf    l.efy   .d.hk   qqxtn   kyuxh   cn
```

Plaintext Obtained Back from the Ciphertext

this is a sample text written with a character set of twenty-nine characters. all lower case alphabet and . and new line character. this is stored in file ptext.c and cipher text will be stored in ctext.c lets include some characters abcdefghijklmnopqrstuvwxyz.

Analysis

Number of characters in plaintext, 273; cardinality of character set, 29; value of chi-square statistic $V = 50$. From the chi-square distribution table we found that this value of V was satisfactory.

Test 2

Plaintext Used to Cipher

This is sample text to test our program "CRYPT". This text is stored in file 'ptext.c'. The coded message is stored in file 'ctext.c'. The program will ask the user if he wants to code or decode and take appropriate action. It will also ask the user to type in a polynomial to be used as key. It can code all ASCII characters (128).

Chi-square test is done on the keys generated to encipher or decipher this message. All the keys, number of times they occur and chi-square results are then printed. Total number of characters are also printed. For, chi-square test number of characters should be at least five times cardinality of our character set (128).

In file ctext.c, new lines, even new page start at unpredictable places because we don't know which character could be enciphered to new line or new page control character. So, if we try to print ctext.c, we get some garbage. But, if that file is deciphered, we get our plaintext i.e. this file without any problem.

Polynomial Equation Used for the Key

$$P(x) = x^3 - 5 = 0$$

A Constant Cons Used as Initial Plaintext Symbol to Cipher–Decipher

$$\text{Cons} = 0$$

Character Set Used

ASCII character set; 128 characters.

Key Obtained to Cipher–Decipher Text Symbol

Do you want to code?(y or n) – y
Type in key polynomial coefficients

```
1 0 0 -5
0 3 6 11 106 39 83 76 15
69 78 81 53 36 63 109 122 87 17
0 19 49 119 11 61 46 61 59 70
21 28 65 99 95 100 43 62 41 89
11 34 109 26 89 88 17 107 71 54
91 96 119 117 58 65 29 41 13 116
33 118 107 9 26 83 71 52 79 127
15 79 117 116 47 63 6 101 15 44
83 66 85 27 95 30 123 84 115 7
115 70 31 70 57 63 104 105 14 111
104 95 32 57 57 27 112 117 31 24
47 125 66 113 115 91 122 37 31 63
17 61 7 17 44 63 73 112 23 124
21 114 109 79 36 127 114 55 127 79
8 51 94 101 116 13 111 73 112 89
107 63 44 61 120 69 29 28 107 74
97 117 52 107 47 109 0 37 101 119
60 31 1 102 43 112 117 71 106 121
83 35 18 17 13 116 119 45 78 21
57 19 34 43 61 49 9 10 61 0
13 9 52 123 33 76 55 76 25 66
123 27 44 11 98 25 84 123 109 111
30 9 123 84 51 63 118 127 100 87
19 89 90 35 7 100 51 113 83 66
```

57 105 87 110 55 16 3 51 74 29
89 45 50 89 125 118 5 17 43 86
105 42 79 21 48 83 7 74 79 59
2 77 67 75 122 117 83 32 81 17
66 21 105 125 42 9 17 94 127 101
32 29 109 106 11 30 125 37 120 19
123 102 9 120 127 27 67 116 17 31
64 59 81 71 18 109 103 99 43 51
107 10 103 61 68 27 115 13 6 29
115 70 75 43 102 29 41 67 111 54
115 125 51 62 55 61 18 49 93 49
92 45 24 5 87 55 44 5 127 73
40 111 49 70 51 77 77 16 21 5
121 50 121 76 99 39 97 123 54 123
49 24 1 39 38 119 87 102 61 115
12 15 11 18 23 97 84 55 125 101
98 73 66 59 69 123 105 28 69 33
98 7 47 80 29 67 66 1 67 11
46 63 124 15 108 63 75 86 9 109
67 103 113 62 79 7 64 11 77 52
127 24 105 92 89 103 18 115 101 92
107 87 123 100 63 127 46 21 87 64
53 122 123 19 76 29 114 33 124 69
34 9 89 112 57 121 85 42 125 60
15 73 70 85 4 11 63 48 75 45
46 1 39 81 122 71 103 21 125 78
7 65 72 119 119 110 85 80 27 71
74 59 3 33 96 65 34 13 105 44
55 15 81 92 11 92 49 34 51 113
59 81 8 57 65 69 49 24 33 57
90 19 76 97 109 14 25 110 33 77
110 101 127 20 99 19 86 31 64 7
57 76 127 120 77 18 89 68 33 63
111 80 75 113 7 125 102 9 89 80
55 34 13 120 45 72 53 15 91 86
109 34 97 1 96 65 98 75 98 107
125 107 8 39 4 119 102 91 97 36
75 113 58 35 109 82 15 71 78 39
53 49 68 125 13 53 26 29 127 71

13 98 99 92 121 51 14 65 54 87
83 66 107 25 71 63 78 39 115 81
119 12 109 28 99 54 85 40 85 5
87 60 51 21 56 35 109 66 111 53
53 64 119 110 5 27 69 94 77 21
80 127 26 51 88 25 59 63 56 53
69 78 49 103 87 52 105 82 55 29
57 101 124 49 111 115 10 55 10 25
12 63 77 121 107 113 72 23 67 49
82 103 61 104 101 64 37 61 25 30
115 55 8 125 75 14 83 32 17 77
108 99 18 9 59 4 55 111 72 125
23 44 13 95 122 57 60 73 11 36
5 99 58 63 45 122 9 56 33 71
28 49 37 83 11 45 84 5 91 94
51 117 73 50 119 14 121 30 105 7
105 120 7 13 15 8 19 34 19 21
26 45 109 112 95 31 78 23 70 37
97 4 35 47 49 40 81 57 6 59
69 83 80 3 83 64 47 98 79 61
88 35 37 62 107 38 13 17 11 28
71 35 95 99 28 27 29 120 13 70
35 37 24 51 8 127 44 49 30 23
122 29 64 35 37 119 61 84 115 97
18 119 71 8 69 79 84 1 5 111
8 43 102 111 5 41 81 120 121 21
124 77 30 59 63 127 61 111 126 113
112 33 51 60 103 31 77 56 95 14
49 119 96 49 113 2 21 27 86 77
113 118 9 61 38 13 123 116 1 109
101 68 119 38 79 35 123 83 24 71
51 52 109 57 16 59 115 106 51 35
18 9 69 21 112 109 103 68 119 49
25 24 17 55 61 23 48 87 54 39
85 113 38 101 69 39 38 113 109 4
99 66 123 85 108 15 65 19 80 99
81 88 31 100 121 101 45 99 103 6
83 104 121 75 20 67 85

Count of Occurrence of Each Different Key

Note that count j means number of occurrences of key j:

count0 = 4	count42 = 3	count84 = 7
count1 = 7	count43 = 7	count85 = 9
count2 = 2	count44 = 8	count86 = 5
count3 = 4	count45 = 9	count87 = 11
count4 = 5	count46 = 4	count88 = 4
count5 = 9	count47 = 6	count89 = 10
count6 = 5	count48 = 3	count90 = 2
count7 = 11	count49 = 18	count91 = 5
count8 = 6	count50 = 3	count92 = 6
count9 = 13	count51 = 16	count93 = 1
count10 = 4	count52 = 6	count94 = 4
count11 = 13	count53 = 8	count95 = 7
count12 = 3	count54 = 6	count96 = 4
count13 = 14	count55 = 14	count97 = 8
count14 = 6	count56 = 4	count98 = 7
count15 = 11	count57 = 13	count99 = 12
count16 = 3	count58 = 3	count100 = 5
count17 = 12	count59 = 11	count101 = 12
count18 = 9	count60 = 5	count102 = 7
count19 = 10	count61 = 16	count103 = 10
count20 = 2	count62 = 4	count104 = 4
count21 = 14	count63 = 17	count105 = 10
count22 = 0	count64 = 8	count106 = 4
count23 = 7	count65 = 8	count107 = 12
count24 = 8	count66 = 10	count108 = 3
count25 = 8	count67 = 8	count109 = 18
count26 = 5	count68 = 5	count110 = 5
count27 = 9	count69 = 12	count111 = 12
count28 = 7	count70 = 8	count112 = 8
count29 = 12	count71 = 13	count113 = 12
count30 = 7	count72 = 4	count114 = 3
count31 = 9	count73 = 7	count115 = 13
count32 = 4	count74 = 4	count116 = 6
count33 = 10	count75 = 9	count117 = 7
count34 = 8	count76 = 7	count118 = 4
count35 = 11	count77 = 12	count119 = 16
count36 = 4	count78 = 7	count120 = 8
count37 = 9	count79 = 10	count121 = 10
count38 = 6	count80 = 7	count122 = 6
count39 = 9	count81 = 10	count123 = 14
count40 = 3	count82 = 3	count124 = 5
count41 = 4	count83 = 14	count125 = 13
		count126 = 1
		count127 = 15

Ciphertext Obtained

It was not possible to print the ciphertext file in a manner so that is could be included here, because it had unpredictable new line and new page characters, which caused the printer to act strange. But we did get the plaintext back.

Plaintext Obtained Back from Ciphertext

This is sample text to test our program "CRYPT." This text is stored in file ptext.c. The coded message is stored in file ctext.c. The program will ask the user if he wants to code or decode and take appropriate action. It will also ask the user to type in a polynomial to be used as key. It can code all ASCII characters(128).

Chi-square test is done on the keys generated to encipher or decipher this message. All the keys, number of times they occur and chi-square results are then printed. Total number of characters are also printed. For, chi-square test number of characters should be at least five times cardinality of our character set(128).

In file ctext.c, new lines, even new page start at unpredictable places because we don't know which character could be enciphered to new line or new page control character. So, if we try to print ctext.c, we get some garbage. But, if that file is deciphered, we get our plaintext i.e. this file without any problem.

Analysis

Total number of characters in plaintext, 1007; cardinality of character set used (ASCII), 128; value of chi-square statistic $V = 253$. The value of V in this case shows that our keys were not very random. Part of the reason for this was that only few characters were used in the plaintext out of the character set. We should try to choose a character set such that most of the characters would be used all the time.

7. CONCLUSIONS

The proposed scheme is an efficient and crypto secure method for cruptography. The chi-square test was used to study the behavior of the keys used to cipher–decipher each character of the text. The language used to implement the transformation algorithm was the **C** programming language. The scheme proposed in this paper provides a sufficient amount of keys from one polynomial and a constant that need to be exchanged only once between the communicating parties.

Open Questions for Further Study

The repetition cycle for certain polynomials is very small, which makes our keys less random. So, to better conceal our message, such polynomials should be modified. Also, if the text has some repeating pattern, it could induce some pattern in our keys, and we should modify our algorithm for this kind of text.

APPENDIX. ALGORITHMS

Algorithm Upper Bound (*P*, *n*)

To compute the exact upper bound of the root of the polynomial using Corollary 4.1. Let P be the array of coefficients of the polynomial and n be the degree of the polynomial. (In these algorithms the polynomial has one sign variation in the sequence of its coefficients.)

```
Set max to 0
Set i to n
  While (P_i <= 0)
      if (Abs(P_i) > max)
          max ← Abs(P_i)
          i ← i − 1
      End-if
  End-while
Set v to i
for i = 0 to i <= v by increment of 1 do
Sum ← Sum + P_i
End-for
upbound ← (max/Sum + 1)
End Algorithm.
```

Algorithm Lower Bound (*P*, *n*, upbound)

To compute the exact lower bound of the root of the polynomial using the hybrid false position method. Let P be the array of coefficients of the polynomial, n the degree of the polynomial, and upbound the upper bound of the root of the polynomial.

```
Set Xold ← 0
Set Lx ← 0
Set Rx ← upbound
Set done ← no
    LVAL ← P_0(LX)^n + P_1(LX)^(n−1) + ··· + P_n
    RVAL ← P_0(RX)^n + P_1(RX)^(n−1) + ··· + P_n
While (done equals no) do the following:
    ILX ← [LX] (floor function)
    ILX ← [RX]
```

```
        If (Abs(ILX − IRX) <= 1) then
            if (LVAL and RVAL are both + ve or both − ve)
                XNEW = LX
            Else NXEW = RX
                Set done ← Yes
        Else slope ← (LVAL − RVAL)/(LX − RX)
            NXEW ← (LX − LVAL)/Slope;
            NEWVAL = P0(XNEW)^n + P1(XNEW)^(n−1) + ··· + Pn
            If (NEWVAL and LVAL are both + ve or both − ve)
                LX ← XNEW
                LVAL ← NEWVAL
                RVAL ← RVAL/2.0
            Else
                RX ← XNEW
                LX ← LVAL/2.0
            End-if
            XOLD ← XNEW
            End-if
    End-while
    IXNEW ← XNEW
    Lower bound ← IXNEW
    End of Algorithm.
```

Algorithm Translate (P, n, A)

To translate a polynomial $P(x)$ to $P(A + 1/x)$. Let P be the array of the coefficients of the polynomial and n be the degree of polynomial.

```
    For i ← 0 to i <= n by increment of 1 do
        For j ← 0 to j <= n by increment of 1 do
            M(i,j) ← 0
        End-for
    End-for
    For i ← 1 to i <= n by increment of 1 do
        M(0,i) ← M(0,i−1) * A + P(i)
    End-for
    For i ← 1 to i <= n − 1 by increment of 1 do
        M(i,0) ← P(0)
        k ← n − 1
        For j ← 1 to j <= k by increment of 1 do
            M(i,j) ← A * M(i,j−1) + M(i−1,j)
        End-for
```

End-for
$M_{n,0} \leftarrow P_0$
For $i \leftarrow 0$ to $i <= n$ by increment of 1 do
    $k \leftarrow n - 1$
    If $M_{0,n} < 0$
        $P_i = -M_{i,k}$
    Else
        $P_i = M_{i,k}$
End-for
End of Algorithm.

Algorithm Update (P, Q, A)

To update the expression of the root x of the polynomial P by the partial quotient A.

    Set $N \leftarrow P_0 * A + P_i$
    Set $D \leftarrow Q_0 * A + Q_i$
    Set $X \leftarrow N/D$
    Set $P_0 \leftarrow P_1$
    Set $Q_0 \leftarrow Q_1$
    Set $P_i \leftarrow N$
    Set $Q_i \leftarrow D$
End of Algorithm.

Algorithm Encipher (M, C)

M represents the plaintext and C the ciphertext. Let Cons be the constant for the encipherment of the initial character of the plaintext, and let size be the integer modulo which all arithmetic is performed.

Initialize:
    $P_0 \leftarrow 0$
    $Q_0 \leftarrow 1$
    $P_1 \leftarrow 1$
    $Q_1 \leftarrow 0$
    $i \leftarrow 1, M_0 \leftarrow 0$
    Key $\leftarrow$ Cons
    Num $\leftarrow 0$
    Den $\leftarrow 0$
Read the polynomial k and degree of polynomial n

While there are characters to encipher do
  Max ← upbound (k, n)
  Min ← lowbound (k, n, max)
  Update (P, Q, min)
  Translate (k, n, min)
  Num2 ← Num + Key1
  Den2 ← Den + Key1
  Key2 ← (Num2 XOR Den2) mod size
  get a character M_i to encipher
  $C_i \leftarrow (M_i + \text{Key2}) \bmod \text{size}$
  Key1 $= M_i$
  $i \leftarrow i + 1$
End-while
End of Algorithm.

Algorithm Decipher (C, M)

Variable names are the same as in the previous algorithm.

Initialize:
  $P_0 \leftarrow 0$
  $Q_0 \leftarrow 1$
  $P_1 \leftarrow 1$
  $Q_1 \leftarrow 0$
  Keyl ← Cons
  Num ← 0
  Den ← 0

Read the polynomial k and the degree of polynomial n

While there are characters to decipher do
  Max ← upbound (k, n)
  Min ← lowbound (k, n, max)
  Update (P, Q, min)
  Translate (k, n, min)
  Num2 ← Num + Key1
  Den2 ← Den + Key1
  Key2 ← (Num2 XOR Den2) mod size
  get a character C_i to decipher
  $M_i \leftarrow (C_i - \text{Key2} + \text{size}) \bmod \text{size}$
  Key1 $\leftarrow K_i$
  $i \leftarrow 1 + 1$
End-while
End of Algorithm.

REFERENCES

1. A. G. Akritas, An implementation of Vincent's theorem, *Numerishche Mathematik* **36**:53–62 (1980).
2. A. G. Akritas, The fastest exact algorithms for the isolation of the real roots of a polynomial equation, *Computing* **24**:299–313 (1980).
3. A. G. Akritas, Exact algorithm for the implementation of Cauchy's rule, *Int. J. Comp. Math.* **9**:323–333 (1981).
4. A. G. Akritas, Application of Vincent's theorem in cryptography or one-time pads made practical, *Cryptologia* **6**(4):312–318 (1982).
5. A. G. Akritas and S. D. Danielopoulos, On the complexity of algorithms for the translation of polynomials, *Computing* **24**:51–60 (1980).
6. A. G. Akritas and S. D. Danielopoulos, An unknown theorem for the isolation of the roots of polynomials, *Ganita Bharati* **2**(3/4):41–49 (1980).
7. W. F. Ehrsam, S. M. Matyas, C. H. Meyer, and W. L. Tuchman, A cryptographic key management scheme for implementing the data encryption standard, *IBM Syst. J.* **17**(2):106–125 (1978).
8. W. Diffie and M. E. Hellman, A critique of the proposed date encryption standard, *Comm. ACM* (March 1976), pp. 164–165.
9. E. Gudes, The design of a cryptography based secure file system, *IEEE Trans. Software Eng.* **SE-6**(5): 411–420 (September 1980).
10. E. Gudes, F. A. Stahl, and H. E. Koth, Applications of cryptographic transformations to data base security, in *Proceedings of National Computer Conference* (1976).
11. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, New Jersey (1978).
12. D. E. Knuth, *The Art of Computer Programming*, Addison Wesley, Reading, Mass., Vol. 2, pp. 1–70 (1969).
13. S. Lang and H. Trotter, Continued franctions for some algebraic numbers, *J. Reine Angew. Math.* **255**:122–134 (1972).
14. R. C. Merkle, Secure communications over insecure channels, *Comm. ACM* **1978**:194–299.
15. K. H. Ng, Polynomial real root approximation using continued fractions, M.S. Research Report, Department of Computer Science, University of Kansas, Lawrence, Kansas (1980).
16. S. M. Pizer, *Numerical Computing and Mathematical Analysis*, Science Research Associates, Chicago, pp. 187–217 (1975).
17. G. B. Purdy, A high security log-in procedure, *CACM* **17** (August 1974).
18. A. Rampuria, A secure method for cryptography, M.S. Research Report, Department of Computer Science, University of Kansas, Lawrence, Kansas (1982).
19. J. V. Uspensky, *Theory of Equations*, McGraw-Hill, New York (1948).
20. A. J. H. Vincent, Sur la résultion des équation numeriques, *J. Math. Pures Appl.* **1**:341–372 (1836).

Printed in Belgium