Proc. Indian Acad. Sci. (Math. Sci.), Vol. 89, Number 2, May 1980, pp. 75–93.

A cryptographic system based on finite field transforms

E V KRISHNAMURTHY and VIJAYA RAMACHANDRAN
School of Automation, Indian Institute of Science, Bangalore 560 012, India

MS received 24 March 1979; revised 21 August 1979

Abstract. In this paper, we develop a cipher system based on finite field transforms. In this system, blocks of the input character-string are enciphered using congruence or modular transformations with respect to either primes or irreducible polynomials over a finite field. The polynomial system is shown to be clearly superior to the prime system for conventional cryptographic work.

Keywords. Ciphers; computer data protection; cryptography; finite field transforms; irreducible polynomials; military communication; polynomial congruences; prime congruences; public crypto-systems.

1. Introduction

A cryptographic system [7], [11] consists of a set of transformations T_j, each of which can act on an input message M to produce a corresponding enciphered message E_j, i.e.,

$$E_j = T_j(M).$$

Each transformation T_j is specified by an associated key k_j. The enciphered message is transmitted to the receiver over an interceptable medium. At the receiving end, the original message M is recovered by applying the inverse transformation T_j^{-1} on the received (enciphered) message E_j. It is clear that the existence of T_j^{-1} is a necessary condition for T_j to be a valid encoding transformation.

Example :

Substitution cipher—In this cipher, each input character is transformed into another character. The transformed character set may or may not be the same as the input character set. In the former case, the transformation is a simple permutation of the input character set. Let the (ordered) input characters be a, b, c, d. If the transformed character set is $\{0, 1, 2, 3\}$, any permutation of these 4 characters represents a key to a particular transformation. If 2, 1, 3, 0 is the specified key k_j, T_j is given by $a \to 2$, $b \to 1$, $c \to 3$, $d \to 0$, pna T_j^{-1} is given by $0 \to d$, $1 \to b$, $2 \to a$, and $3 \to c$. There are $4! = 24$ different transformations possible.

It is assumed that the enemy (i.e., the persons from whom the message M is protected) knows the set of transformations T_j being used. He also has available,

the *a priori* probabilities of the input characters. However, the enemy does not know which particular transformation is currently being used. This transformation T_j is characterised completely by the corresponding key k_j, which should be communicated to the receiving end through a protected channel.

If the enemy intercepts a sufficient number N of transmitted characters, he will be able to break the code, since he knows the *a priori* probabilities of the input characters. To illustrate this point, let $P(a) = 0{\cdot}8$, $P(b) = 0{\cdot}04$, $P(c) = 0{\cdot}15$, and $P(d) = 0{\cdot}01$ be the *a priori* probabilities in the example given above. Let the enemy intercept the 20 transmitted characters 3 2 2 2 1 2 0 2 2 2 2 3 3 2 2 2 2 2 2 2. As mentioned earlier, he knows that the cipher is one from the set of simple substitutions. By applying his knowledge of the *a priori* probabilities, he can immediately conclude that a is being coded as 2 and c as 3. At this stage he cannot assert anything about the encoding of b and d, but for a larger N he will be in a position to do so.

A good code (i.e., a set of encoding transformations) should satisfy two main criteria:

(i) N should become very large before the enemy is in a position to decipher the code using the *a priori* probabilities.

(ii) The number of elements in the key for the transformation should be relatively small.

Block coding is a very simple method of achieving (i). Here, the encoding transformation is applied to a block of k characters of the message at a time. If the number of different input characters is n, then block coding using k characters effectively increases the number of elements in the input character set to n^k. This is because each different sequence of k characters can be treated as an input character. In the example given above, if $k = 3$, the number of input characters for the block coding increases from 4 to $4^3 = 64$. The corresponding characters are 000, 001, 002, $\cdots$, 333. The probabilities for the occurrence of these characters can be calculated (with some difficulty); however, the value of N at which decoding by the enemy becomes possible is much larger than in the case of simple substitution. (It should be noted that the number of different keys in this case has increased to 64!). The price paid for this is the increase in the size of the key; while the key formerly contained 4 elements, it now contains 64 elements.

One attractive feature of the scheme presented in this paper is that it achieves block coding with a relatively small number of elements in the key. For instance, block coding with $k = 4$ and $n = 29$ can be achieved using just 5 elements in the key (as opposed to 29^4 elements for a block substitution code). Obviously, the number of different keys is much less than 29^4 !. However, the trade-off between key size and the number of transformations appears very attractive.

2. Finite structures from the number system

In this section, we shall outline some basic results from number theory and algebra. These results can be found from [9].

If p is a prime number, then the set of integers $0, 1, 2, \cdots, p-1$ together with addition and multiplication modulo p forms a field containing p elements. This is known as the *finite (Galois) field GF* (p). (It is easily verified that if p is not a

prime, then the resulting algebraic structure is not a field; it is a commutative ring).

Let

$$M = \prod_{i=1}^{k} p_i^{r_i}.$$

Then, given any positive integer a less than M, we can find its residues with respect to each $p_i^{r_i}$.

Let $a_i = a \bmod p_i^{r_i}$*, $0 \leqslant a_i < p_i^{r_i}$, $1 \leqslant i \leqslant k$.

To reconstruct a from the a_i's is not very straightforward. The formula for this reconstruction is given in the following theorem.

Theorem 1

Chinese remainder theorem. Let $p_1, p_2, \ldots, p_k$ be a set of k distinct primes and let $r_1, r_2, \ldots, r_k$ be positive integers. Let $a_1, a_2, \ldots, a_k$ be an arbitrary set of integers. Then the simultaneous congruences

$$a_i = a \bmod p_i^{r_i}, \quad i = 1, 2, \ldots, k$$

have a unique solution for a mod M, where

$$M = \prod_{i=1}^{k} p_i^{r_i}.$$

The solution is given by

$$a = \left(\sum_{i=1}^{k} d_i \left((a_i d_i^{-1}) \bmod p_i^{r_i}\right)\right) \bmod M,$$

where $d_i = M/p_i^{r_i}$, and $d_i^{-1} = (d_i \bmod p_i^{r_i})^{-1} \bmod p_i^{r_i}$.

(The inverse notation is defined in Definition 3 below).

Let a be a non-zero element of $GF(p)$. Then a^k is also a non-zero element of $GF(p)$ by the closure property. Since $GF(p)$ contains only p elements, there exist positive integers m and n such that

$$a^m = a^n \bmod p,$$

i.e., $a^k = 1 \bmod p$ for some integer $k > 0$.

Theorem 2. *Fermat's Theorem.* For every non-zero element a belonging to $GF(p)$,

$$a^{p-1} = 1 \bmod p.$$

Corollary 2.1. For every element a belonging to $GF(p)$, and any positive integer s,

$$a^{s(p-1)+1} = a \bmod p.$$

---

* For convenience, we use the equality sign for denoting congruence; this will not lead to any confusion as the statement modulo w.r.t. some integer will always follow the equality sign in such a case.

The extension of the result in Corollary 2.1 to the case when the modulus is the product of distinct primes is given in theorem 3. Before presenting theorem 3, however, we require the following two definitions.

Definition 1. *Euler's totient function* $\Phi(m)$. Let m be the positive integer. The number $\Phi(m)$ is defined to be the number of positive integers less than or equal to m that are relatively prime to m.

Let the prime decomposition of m be

$$p_1^{r_1} \cdot p_2^{r_2} \cdots p_k^{r_k}.$$

Then, $\Phi(m) = p_1^{(r_1-1)} \cdot p_2^{(r_2-1)} \cdots p_k^{(r_k-1)} (p_1 - 1)(p_2 - 1) \cdots (p_k - 1)$.

In the case when m is the product of distinct primes (i.e., $m = p_1 p_2 \cdots p_k$), $\Phi(m) = (p_1 - 1)(p_2 - 1) \cdots (p_k - 1)$.

Definition 2. Given two integers a and b, the notation (a, b) defines the greatest common divisor (gcd) of a and b. We define $(O, b) = b$.

Theorem 3. Let m be the product of distinct primes $p_1, p_2, \cdots, p_k$. Then, for every integer a, and any positive integer s,

$$a^{s\Phi(m)+1} = a \bmod m.$$

Proof. Let $a_i = a \bmod p_i$, $i = 1, 2, \cdots, k$.

Case 1. $(a,m) = 1$

Then $(a_i, p_i) = 1, i = 1, 2, \cdots, k$.

Hence $a_i^{p_i-1} = 1 \bmod p_i$ by Theorem 2

i.e., $a^{r(p_i-1)} = 1 \bmod p$

Hence $a^{p_i-1} = 1 \bmod p_i$.

Thus $a^{s\Phi(m)} = 1 \bmod m$, since $\Phi(m) = \prod_{i=1}^{k} (p_i - 1)$

or $a^{s\Phi(m)+1} = a \bmod m$.

Case 2. $(a, m) > 1$

Without loss of generality, we can assume

$$a = c \cdot p_1 \cdot p_2 \cdots p_j, \text{ and } (a, p_i) = 1, \quad j < i \leqslant k.$$

Then $a_i = 0$ for $1 \leqslant i \leqslant j$ and $a_i^{p_i-1} = 1 \bmod p_i$, $j < i \leqslant k$. (1)

Thus $a^{r(p_i-1)+1} = a \bmod p_i$, $j < i \leqslant k$ (2)

Also $a^r = 0 \bmod p_i$ for any integer r, $1 \leqslant i \leqslant j$. (3)

Now consider $a^{s\Phi(m)+1}$.

By (1), (2), and (3) we have

$$a^{s\Phi(m)+1} = a_i \bmod p_i, \quad 1 \leqslant i \leqslant k,$$

or $a^{s\Phi(m)+1} = a \mod m$.

We now present a few standard theorems from the theory of congruences.

Theorem 4. If $(a, m) = 1$, then the congruence $ax = b \mod m$ has a unique solution $x = x_1$ with $0 \leqslant x_1 < m$.

Definition 3. Let $(a, m) = 1$. Then, b is the inverse of a mod m (denoted a^{-1}) if $ab = 1 \mod m$.

Theorem 5. Let p be a prime and let $(n, p-1) = 1$. Then the congruence

$$x^n = a \mod p$$

has exactly one solution given by

$$x = a^{m_0} \mod p,$$

where $m_0 = n^{-1} \mod p - 1$.

Theorem 6. Let m be the product of distinct primes $p_1, p_2, \cdots, p_k$ and let $(n, \Phi(m)) = 1$. Then the congruence

$$x^n = a \mod m$$

has exactly one solution

$$x = a^{m_0} \mod m$$

where $m_0 = n^{-1} \mod \Phi(m)$.

Proof

$$a^{m_0} = x^{nm_0} \mod m.$$

But $nm_0 = 1 \mod \Phi(m)$

or $nm_0 = s \cdot \Phi(m) + 1$ for some positive integer s.

Thus, $a^{m_0} = x^{s\Phi(m)+1} \mod m$

$= x \mod m$ by theorem 3.

3. Polynomial systems

In this section, we consider finite fields and rings generated by polynomials over $GF(p)$ [3], [4].

Definition 4. A polynomial is said to be *defined* over $GF(p)$ if its coefficients lie in $GF(p)$.

Definition 5. A polynomial $\phi(x)$ is an *irreducible polynomial* if it has no divisors other than scalars and scalar multiples of itself.

Let $\phi(x)$ be an irreducible polynomial of degree d over $GF(p)$. Then the set of all polynomials over $GF(p)$ with degree less than d, together with addition and multiplication modulo $\phi(x)$, forms a field. This field is known as the *finite* (*Galois*)

field $GF(p^d)$ and it contains p^d elements. The additive identity is 0 and the multiplicative identity is 1. If $\phi(x)$ is not irreducible, then the above structure becomes a finite commutative ring with the same identity elements.

Definition 6. Let $a(x)$ belong to $GF(p^d)$ and let $\phi(x)$ be an irreducible polynomial over $GF(p)$ of degree d. Then, the least positive integer k for which the equation

$$a^k(x) = 1 \bmod \phi(x)$$

is satisfied, is called the *order* of $a(x)$.

Definition 7. If $k = p^d - 1$, then $a(x)$ is called a *primitive element* of the above field.

The powers of a primitive element generate all the non-zero elements of the finite field.

Given two irreducible polynomials $\phi(x)$ and $\psi(x)$ of degree d over $GF(p)$, we can generate two different fields contaning p^d elements by taking the field operations modulo $\phi(x)$ and $\psi(x)$ respectively.

Theorem 7. The number of irreducible polynomials of degree d over $GF(p)$ is given by

$$I_{d,p} = \frac{1}{d} \sum_{\substack{k, \\ k \mid d}} \mu(k) \cdot p^{d/k},$$

where $$\mu(k) = \begin{cases} 1 \text{ if } k = 1 \\ (-1)^r \text{ if } k \text{ is the product of } r \text{ distinct primes} \\ 0 \text{ if } k \text{ contains any repeated prime factors.} \end{cases}$$

Thus, we can have $I_{d,p}$ different representations of $GF(p^d)$ by choosing different irreducible polynomials.

We now generalise some of the results of the previous section to the polynomial case.

Theorem 8. *Chinese remainder theorem for polynomials.* Let $\phi_1(x), \phi_2(x), \cdots, \phi_k(x)$, be irreducible polynomials over $GF(p)$ and let $r_1, r_2, \cdots, r_k$ be positive integers. Let $a_1(x), a_2(x), \cdots, a_k(x)$ be arbitrary polynomials over $GF(p)$. Then, the simultaneous congruences

$$a_i(x) = a(x) \bmod \phi_i^{r_i}(x), \quad i = 1, 2, \cdots, k$$

have a unique solution for $a(x)$ modulo $\phi(x)$ where

$$\phi(x) = \prod_{i=1}^{k} \phi_i^{r_i}(x).$$

The solution is given by

$$a(x) = \Big(\sum_{i=1}^{k} d_i(x) \left[(a_i(x)\, d_i^{-1}(x)) \bmod \phi_i^{r_i}(x) \right] \Big) \bmod \phi(x),$$

where $d_i(x) = \phi(x)/\phi_i^{r_i}(x)$

and $d_i^{-1}(x) = (d_i(x) \bmod \phi_i^{r_i}(x))^{-1} \bmod \phi_i^{r_i}(x)$.

Theorem 9. Let $\phi(x)$ be an irreducible polynomial over $GF(p)$ of degree d. Then, for every non-zero $a(x)$ in $GF(p^d)$,

$$a^{p^d-1}(x) = 1 \bmod \phi(x).$$

Corollary 9.1. For every $a(x)$ in $GF(p^d)$,

$$a^{s\cdot(p^d-1)+1}(x) = a(x) \bmod \phi(x),$$

where s is any positive integer.

Before we extend the above result to the case of composite modulus, we introduce the concept of ' generalised totient function '.

Definition 8. *Generalised totient function*

Let $\phi(x)$ be a polynomial over $GF(p)$ which is the product of distinct irreducible polynomials $\phi_1(x), \phi_2(x), \cdots, \phi_k(x)$. Let $\delta_1, \delta_2, \cdots, \delta_j$ be distinct positive integers representing the degrees of the ϕ_i's. Clearly $j \leqslant k$. We define the generalised totient function of $\phi(x)$ as

$$Q(\phi) = \prod_{i=1}^{j} (p^{\delta_i} - 1).$$

Note. $Q(\phi)$ is not the ' natural ' extension of the number-theoretic totient function, in the sense that it does not give the number of polynomials over $GF(p)$ of degree less than $\phi(x)$ and relatively prime to it. However, this is the generalisation which is useful for the extension of Corollary 9.1.

Theorem 10. Let $\phi(x)$ be a polynomial of degree d over $GF(p)$, which is the product of distinct irreducible. Polynomials $\phi_1(x), \phi_2(x), \cdots, \phi_k(x)$. Let $\delta_1, \delta_2, \cdots, \delta_j$ be distinct positive integers, representing the degrees of the ϕ_i's. Then, for every polynomial $a(x)$ over $GF(p)$ of degree less than d,

$$a^{s\cdot Q(\phi)+1}(x) = a(x) \bmod \phi(x),$$

where s is any positive integer.

Proof. Similar to the proof of theorem 3.

Theorem 11. Let $\phi(x)$ be an irreducible polynomial of degree d over $GF(p)$ and let $(n, p^d - 1) = 1$. Then the congruence

$$w^n(x) = a(x) \bmod \phi(x)$$

has exactly one solution, given by

$$w(x) = a^{m_0}(x) \bmod \phi(x),$$

where $m_0 = n^{-1} \bmod (p^d - 1)$.

Proof. Let $\alpha(x)$ be a primitive element of the finite field defined by $\phi(x)$ and let $\alpha^j(x) = a(x) \bmod \phi(x)$ for some $j, 0 < j \leqslant p^d - 1$.

Now, let $w(x) = a^k(x) \bmod \phi(x)$ satisfy the equation

$$w^n(x) = a(x) \bmod \phi(x),$$

i.e., $$a^{kn}(x) = a^j(x) \bmod \phi(x).$$

Hence, $nk = j \bmod (p^d - 1)$ by Definition 10.

By theorem 5, this equation has exactly one solution for k if $(n, p^d - 1) = 1$, which is true by assumption.
Thus $k = jn^{-1} \bmod (p^d - 1)$,

or $$w(x) = a^{jn^{-1}}(x) \bmod \phi(x) = a^{n^{-1}}(x) \bmod \phi(x)$$

$$= a^{m_0}(x) \bmod \phi(x).$$

Theorem 12. Let $\phi(x)$ be the product of distinct irreducible polynomials and let $(n, Q(\phi)) = 1$. Then the congruence

$w^n(x) = a(x) \bmod \phi(x)$ has exactly one solution, given by

$w(x) = a^{m_0}(x) \bmod \phi(x)$, where $m_0 = n^{-1} \bmod Q(\phi)$.

Proof. Similar to the proof of theorem 6.

The proofs for theorems stated in this section without proof can be found in [3] and [4].

4. Prime congruence codes

In this section we describe the congruence system for primes, using the results of § 2. The input character set consisting of K symbols, is first converted into a set of k-bit codes (substitution codes), where k is the smallest number satisfying the inequality $2^k > K$. The input string is processed m characters at a time. The string $c_1, c_2, \ldots, c_m$ is interpreted as the number

$$M = c_1 + c_2 \cdot 2^k + \ldots + c_m \cdot 2^{k \cdot (m-1)}.$$

Hardware-wise, this represents the simple concatenation of the binary representation of the m characters.

A set of r distinct primes $p_1, p_2, \ldots, p_r$ is chosen such that $P = p_1 p_2 \cdots p_r > 2^{mk}$. Also, a positive integer n is chosen such that

$$(n, p_i - 1) = 1, \; i = 1, 2, \ldots, r, \; 1 < n < P',$$

where $$P' = \prod_{i=1}^{r} (p_i - 1).$$

The set of r primes p_i together with n constitute the key to the cipher. The encoding is done by taking the residue x_i of M with respect to each prime p_i, determining the e_i's where

$$e_i = x_i^{n_i}, \; i = 1, 2, \ldots, r,$$

and $$n_i = n \bmod (p_i - 1),$$

and combining the e_i's by the Chinese Remainder Theorem to obtain E. By Corollary 2.1, $x_i^n = x_i^{n_i}$. Hence,

$$E = M^n \bmod P,$$

and this is the enciphered message which is transmitted.

At the receiving end, the code is deciphered in an analogous manner as follows. The residue e_i of the message E with respect to each prime p_i is determined, viz.,

$$e_i = E \bmod p_i, \quad i = 1, 2, \ldots, r.$$

By theorem 6, the unique solution x_i to the congruence

$$x_i^n = e_i \bmod p_i, \quad i = 1, 2, \ldots, r$$

can be determined. The individual x_i's are then recombined using an efficient implementation of theorem 1 (Chinese Remainder Theorem) to obtain M.

5. Polynomial congruence codes

We now consider the polynomial congruence coding scheme. A prime p is chosen to satisfy the inequality $p > K$, where K is the number of symbols in the input character set. The number p would normally be the smallest prime larger than K. A substitution code is prepared, which maps the input character set into a set of distinct elements in $GF(p)$.

The input string is processed m characters at a time. A set of r irreducible polynomials over $GF(p), \phi_1(x), \phi_2(x), \ldots, \phi_r(x)$ is chosen, with degrees $d_1, d_2, \ldots, d_r$, such that

$$\sum_{i=1}^{r} d_i = m.$$

Let $\phi(x) = \prod_{i=1}^{r} \phi_i(x)$. An integer n is chosen,

satisfying the property

$$(n, p^{d_i} - 1) = 1, \quad i = 1, 2, \ldots, r, \quad 1 < n < Q(\phi).$$

Let $M(x)$ be the polynomial obtained by treating the m characters of the input string as coefficients of successive powers of x. The encoding is achieved by taking the residue $w_i(x)$ of $M(x)$ with respect to each $\phi_i(x)$, determining $e_i(x)$, where

$$e_i(x) = w_i^{n_i}(x), n_i = n \bmod (p^{d_i} - 1), i = 1, 2, \ldots, r,$$

and combining the $e_i(x)$ by the Chinese Remainder Theorem to obtain $E(x)$. It is clear that

$$E(x) = M^n(x) \bmod \phi(x).$$

The key to this cipher consists of n and the set of polynomials, $\phi_i(x)$, $i = 1, 2, \ldots, r$.

The receiver decodes the cipher in an analogous manner as follows. The residues $e_i(x)$ of the transmitted message $E(x)$ with respect to each $\phi_i(x)$ are

obtained. By theorems 11 and 12, the unique solution $w_i(x)$ to the congruence

$$w_i^n(x) = e_i(x) \bmod \phi_i(x), \quad i = 1, 2, \ldots, r$$

can be determined. The individual $w_i(x)$ thus obtained are combined using an efficient implementation of theorem 8 to obtain $M(x)$.

Encoding and decoding algorithms

Algorithm 1: *Encoding*

Input

(a) The block length m and the value of the exponent n.

(b) The r distinct irreducible polynomials over $GF(p)$, $\phi_1(x), \phi_2(x), \ldots, \phi_r(x)$ of degrees $d_1, d_2, \ldots, d_r$ respectively. The block length

$$m = \sum_{i=1}^{r} d_i.$$

(c) A substitution code table (τ) for the input character set. Each coded value lies in $GF(p)$.

(d) The input character string.

Output. The coded message string.

Algorithm

```
begin0
    for i ← 1 until r do
      begin1
2.      ni ← n mod (p^di − 1)
      end1
  end0
  begin2
3.  while input string is present do
      begin3
4.      read in the next m characters, s1, s2, ..., sm
5.      form the corresponding substitution code
          c1, c2, ..., cm by table look-up
          (using table τ)
6.      M(x) ← c1 + c2x + ... + cm x^(m−1)
7.      for i ← 1 until r do
          begin4
8.          wi(x) ← M(x) mod φi(x)
9.          ei(x) ← wi^ni(x) mod φi(x)
          end4
```

10. combine $e_1(x), e_2(x), \ldots, e_r(x)$ by Theorem 8 to obtain $E(x)$
11. write the coefficients of $E(x)$, starting with the constant term

end_3
end_2

Algorithm 2. Decoding

Input

(a) The block length m and the value of the exponent n.

(b) The coefficients of the r distinct irreducible polynomials $\varphi_1(x), \varphi_2(x), \ldots, \varphi_r(x)$ over $GF(p)$ of degrees $d_1, d_2, \ldots, d_r$ respectively.

(c) A table (τ') for back conversion from the $GF(p)$ codes to the message character set.

(d) The coded input.

Output. The decoded message string.

Algorithm

$begin_0$

1. *for* $i \leftarrow 1$ *until* r *do*
   $begin_1$
2. $m_i \leftarrow n^{-1} \bmod (p^{d_i} - 1)$
   end_1

end_0
$begin_2$

3. *while* input string is present *do*
   $begin_3$
4. read in the next m characters (i.e., the coefficients of $E(x)$)
5. *for* $i \leftarrow 1$ *until* r *do*
   $begin_4$
6. $e_i(x) \leftarrow E(x) \bmod \varphi_i(x)$
7. $w_i(x) \leftarrow e_i^{m_i}(x) \bmod \varphi_i(x)$
   end_4
8. combine $w_1(x), w_2(x), \ldots, w_r(x)$ using Theorem 8 to obtain $M(x)$
9. decode the coefficients of $M(x)$ by table look-up (using table τ')
10. write the decoded version of the coefficients of $M(x)$, starting with the constant term

end_3
end_2

Example

We use the block size of $m = 4$ and proceed to illustrate Algorithms 1 and 2 using the message

TAT TVAM ASI

We use $GF(29)$ since this will conveniently accommodate the English alphabet, along with full-stop and blank. The substitution code used for this example is given in table 1.

We obtain the block-length of 4 by using two irreducible polynomials of degree 2 over $GF(29)$.

$$\varphi_1(x) = 8 + 10x + x^2,$$

$$\varphi_2(x) = 2 + 7x + x^2.$$

Hence $\varphi(x) = \varphi_1(x) \cdot \varphi_2(x) = 16 + 4x + 17x^2 + x^3$.

The exponent n can take any value between 1 and $Q(\varphi)$ that is relatively prime to $Q(\varphi)$. We shall use $n = 517$. The number of different values for n is $\Phi(Q(\varphi)) - 1 = \Phi(29^2 - 1) - 1 = 191$. Thus, the number of different keys in this system is

$$\binom{I_{29,2}}{2} \cdot 191.$$

$$I_{29,2} = \tfrac{1}{2} \sum_{k,\, k|d} \mu(k) \cdot 29^{d/k}$$

$$= \tfrac{1}{2} \cdot (\mu(1) \cdot 29^2 + \mu(2) \cdot 29)$$

$$= \tfrac{1}{2} \cdot (1 \cdot 841 - 29) = 406,$$

Table 1. Substitution code for example.

Character	Code	Character	Code
A	14	O	15
B	28	P	2
C	27	Q	4
D	25	R	8
E	21	S	0
F	13	T	16
G	26	U	3
H	23	V	6
I	17	W	12
J	5	X	24
K	10	Y	19
L	20	Z	9
M	11	.	18
N	22	b	7

i.e., the number of different keys

$$= 191 \frac{406 \times 405}{2} = 15703065.$$

The encoding procedure is shown in table 2.
The transmitted message string is

4 9 12 4 13 22 10 2 4 23 13 13

At the receiving end, this string is again processed 4 symbols at a time. The decoding steps are shown in table 3. At the end of the procedure, the decoded message is obtained as

TAT TVAM ASI

which is the message originally transmitted.

The key for this scheme consists of the following 5 elements: $n = 517$, constant term of $\varphi_1(x) = 8$, coefficient of x in $\varphi_1(x) = 10$, constant term of $\varphi_2(x) = 2$, and coefficient of x in $\varphi_2(x) = 7$.

6. System performance

In this section, we compare the performance of the procedures presented in § 4 and § 5. The following points should be noted:

(a) *Number of keys* : In the prime coding scheme, the number of keys corresponding to a given set of r primes is given by $\Phi(P') - 1$, where

$$P' = \prod_{i=1}^{r} (p_i - 1).$$

If k such sets of primes are chosen for the system, the total number of different keys is given by

$$\sum_{j=1}^{k} (\Phi(P'_j) - 1),$$

Table 2. An example of encoding using algorithm 1. Block length $m = 4$; Prime field: $GF(29)$; Number of irreducible polynomials $r = 2$; $\phi'(x) = 8 + 10x + x^2$; $\phi_2(x) = 2 + 7x + x^2$; Exponent $n = 517$; Input message: TAT TVAM ASI.

Block Number	Input Block	$M(x)$	$w_i(x) = M(x) \bmod \phi_i(x)$	$e_i(x) = w_i^n(x) \bmod \phi_i(x)$	$E(x)$
1	TATb	$16 + 14x + 16x^2 + 7x^3$	$w_1(x) = 13 + 5x$ $w_2(x) = 24 + 28x$	$e_1(x) = 25 + 25x$ $e_2(x) = 7 + 26x$	$4 + 9x + 12x^2 + 4x^3$
2	TVAM	$16 + 6x + 14x^2 + 11x^3$	$w_1(x) = 1 + 8x$ $w_2(x) = 26 + 19x$	$e_1(x) = 6 + 19x$ $e_2(x) = 21 + 17x$	$13 + 22x + 10x^2 + 2x^3$
3	bASI	$7 + 14x + 17x^3$	$w_1(x) = 4 + 12x$ $w_2(x) = 13 + x$	$e_1(x) = 12 + 16x$ $e_2(x) = 15 + 21x$	$4 + 23x + 13x^2 + 13x^3$

Table 3. An example of decoding using algorithm 2.
Block length $m = 4$; Prime field: $GF(29)$; Number of irreducible polynomials $r = 2$; $\phi_1(x) = 8 + 10x + x^2$; $\phi_2(x) = 2 + 7x + x^2$; Exponent $n = 517$; $m_0 = n^{-1} \bmod 840 = 13$; Input string. 4 9 12 4 13 22 10 2 4 23 13 13.

Block Number	$E(x)$	$e_i(x) = E(x) \bmod \phi_i(x)$	$w_i(x) = e_i^{m_0}(x) \bmod \phi_i(x)$	$M(x)$	Message string
1	$4 + 9x + 12x^2 + 4x^3$	$e_1(x) = 25 + 25x$ $e_2(x) = 7 + 26x$	$w_1(x) = 13 + 5x$ $w_2(x) = 24 + 28x$	$16 + 14x + 16x^2 + 7x^3$	TATb
2	$13 + 22x + 10x^2 + 2x^3$	$e_1(x) = 6 + 19x$ $e_2(x) = 21 + 17x$	$w_1(x) = 1 + 8x$ $w_2(x) = 26 + 19x$	$16x + 6x + 14x^2 + 11x^3$	TVAM
3	$4 + 23x + 13x^2 + 13x^3$	$e_1(x) = 12 + 16x$ $e_2(x) = 15 + 21x$	$w_1(x) = 4 + 12x$ $w_2(x) = 13 + x$	$7 + 14x + 17x^3$	bASI

where the extension of the earlier notation to cover different sets of primes is obvious.

In the polynomial case, let us assume, as before, that we have r irreducible polynomials over $GF(p)$ of degrees $d_1, d_2, \ldots, d_r$ respectively, whose product is $\varphi(x)$. Let $\delta_1, \delta_2, \ldots, \delta_k$ be distinct positive integers representing the degrees of the φ_i's. Further, let there be r_1 irreducible polynomials of degree δ_1, r_2 irreducible polynomials of degree δ_2, ..., r_k irreducible polynomials of degrees δ_k. Clearly, $\sum_{i=1}^{k} r_i = r$. Then, the number of different keys in the system is immediately determined as

$$(\Phi(Q(\phi)) - 1) \cdot \prod_{j=1}^{k} \binom{I_{\delta_j, p}}{r_j}.$$

We arrive at this number as follows: The number of irreducible polynomials of degree δ_j over $GF(p)$ is $I_{\delta_j, p}$ (by theorem 7). The number of different ways in which we can choose r_j polynomials from this set is

$$\binom{I_{\delta_j, p}}{r_i}.$$

By theorem 12, the number of permitted values of n is $\Phi(Q(\varphi)) - 1$. Hence the above formula gives the number of different keys when irreducible polynomials are used.

(b) *Exponentiation* : A fast algorithm for the computation of $M^n \bmod P$ is available in Knuth (1969, p. 399) [8]. The algorithm is given below.

Algorithm 3. *Exponentiation*

Input. A modulus P, a positive integer $M < P$, and a positive exponent n.

Output. $M^n \bmod P$.

Algorithm

$begin_0$

1. *initialise* $N \leftarrow n$; $Y \leftarrow 1$; $Z \leftarrow M$
2. *while* $N \neq 1$ *do*
   $begin_1$
3. *if* N is odd *then*
   $begin_2$
4. $Y \leftarrow Z \cdot Y \bmod P$
5. $Z \leftarrow Z \cdot Z \bmod P$
   end_2
6. *else* $Z \leftarrow Z \cdot Z \bmod P$
7. $N \leftarrow [N/2]$

   end_1
8. $Y \leftarrow Y \cdot Z \bmod P$
9. *write* Y

   end_0

This algorithm requires $[\log_2 n] + v(n)$ multiplications, where $v(n)$ is the number of ones in the binary representation of n. A similar algorithm can be written for the polynomial exponentiation of Algorithms 1 and 2.

(c) *Back conversion* : Let each prime in the prime coding scheme require at most b bits for representation, and let $M_n(k)$ be the time required to multiply two k-bit numbers. An $O(M_n(br) \log r)$ preconditioned algorithm for back conversion is available in [1]. A similar algorithm for polynomial moduli is given below.

Algorithm 4. *Back conversion*

Input

(a) Relatively prime polynomial moduli over $GF(p)$, $\phi_1(x), \phi_2(x), \ldots, \phi_r(x)$, where $r = 2^t$ for some t.

(b) A set of inverses $d_1(x), d_2(x), \ldots, d_r(x)$ such that $d_i(x) = [\phi(x)/\phi_i(x)]^{-1} \bmod \phi_i(x)$, where

$$\phi(x) = \prod_{i=1}^{r} \phi_i(x).$$

(c) A sequence of residues $w_1(x), w_2(x), \ldots, w_r(x)$.

Output. The unique polynomial $M(x)$ over $GF(p)$ with degree less than $\phi(x)$, satisfying the congruences $w_i(x) = M(x) \bmod \phi_i(x)$, $i = 1, 2, \ldots, r$.

Algorithm

$begin_0$

1. *for* $i \leftarrow 1$ *until* r *do* $S_{i0}(x) \leftarrow d_i(x) \cdot w_i(x)$

2. *for* $j \leftarrow 1$ *until* t *do*

begin$_1$

3. *for* $i \leftarrow 1$ *step* 2^j *until* r *do*

begin$_2$

4. $S_{ij}(x) \leftarrow S_{i,j-1}(x) \cdot q_{i+2^{j-1},j-1}(x) + S_{i+2^{j-1},j-1}(x) \cdot q_{i,j-1}(x)$

Comment $q_{ij}(x) = \prod_{m=i}^{i+2^j-1} \phi_m(x)$

end$_2$

end$_1$

5. *write* $S_{1t}(x) \bmod q_{1t}(x)$

end$_0$

If each $\phi_i(x)$ is of degree d, then the complexity of this algorithm is $O(dr \log r \log dr)$.

(d) *GCD and congruence inverse* : The standard method of computing the *gcd* of two integers a and b is Euclid's algorithm. Euclid's algorithm can be extended to find integer multipliers x and y such that

$$xa + yb = (a, b).$$

This algorithm is available in [8] and [1]. This algorithm can be used to generate the inverse of a mod b when $(a, b) = 1$. In this case we have

$$xa + yb = 1$$

i.e., $$xa = 1 \bmod b.$$

Thus the premultiplier x is the required inverse of a mod b.

The *gcd* algorithm is required at the key generation stage to ensure that the chosen n satisfies the criterion $(n, p_i - 1) = 1$, $i = 1, 2, \ldots, r$ for primes (or $(n, p^{d_i} - 1) = 1$ for irreducible polynomials). Similarly, the inverse algorithm is required to generate the m_i's for the exponent at the decoding end. These two algorithms are required only when the key is changed, and hence their complexity does not affect the complexity of the encoding-decoding process.

(e) *Choice of moduli.* The choice of primes is determined largely by the criterion

$$P = \prod_{i=1}^{r} p_i > 2^{mk}.$$

As r becomes larger, the decoding operation becomes more efficient, since the b_i's and m_i's become smaller. The effect of increasing r on the number of keys is not very clear, and will probably depend on the structure of the $(p_i - 1)$, $i = 1, 2, \ldots, r$. The number P should be chosen close to the bound to keep the size of the transmitted code as small as possible.

In the polynomial case, the overall set of irreducible polynomials is determined once p, r_i's and d_i's are fixed. A method for generating such polynomials is given in [3], but the procedure is very complicated. Tables of such polynomials have been constructed, and it is advisable to refer to them where feasible. Alanen and Knuth [2] for instance, tabulate all indexing polynomials for fields containing 1024 elements or less. (An indexing polynomial is an irreducible polynomial, all of whose roots lie in the field generated by it).

As r increases, the decoding operation becomes more efficient in the polynomial case also. The effect on the number of keys is again not clear, since $(p^d - 1)$ decreases with increasing r while the behaviour of

$$\binom{I_{d,p}}{r}$$

is not easy to predict ($I_{d,p}$ is expected to decrease with increasing r). However, it is clear that having irreducible polynomials of different degrees increases the number of keys (for a fixed m), since this choice increases $Q(\phi)$ and hence $\Phi(Q(\phi))$.

(f) *Comparison of prime and polynomial systems* : In comparing the prime and polynomial systems, the following point becomes clear. In the prime system, the cryptanalyst can form a fairly good idea of the value of P as he intercepts more of the transmitted message. For example, let $P = 4199$. As mentioned earlier, the cryptanalyst knows the system being used, i.e., he knows the block size. The message intercepted by him will contain no block whose value is greater than $P = 4199$, while the values occurring in the message will give him a lower bound for P which will come closer to P as more of the message is intercepted. Once the cryptanalyst determines P from this method, the values of the p_i can be determined by factoring P.

In the polynomial case, however, each irreducible polynomial generates the *same* finite field. Hence, different sets of moduli will generate elements from the same set, and hence, the cryptanalyst can obtain no further information about the key by studying the transmitted blocks. Decoding by studying the pattern of occurrence of the blocks can be made impractical by making the block-length large. The cryptanalyst's best method of attack will be to run through the possible combinations of n and the $\phi_i(x)$ to determine which combination gives a meaningful decoded message. The key should be changed sufficiently often to prevent deciphering by such a method (that is, the probability of such a deciphering taking place should be made vanishingly small). Since the number of elements in the key is small, the key can be changed without much inconvenience.

In view of this, a cipher system using irreducible polynomials appears more attractive than a similar system using primes, especially when the degrees of the irreducible polynomials are different.

(g) *Complexity* : Let $M_n(b)$ be the complexity of multiplying two b-bit numbers and let $M_p(k)$ be the complexity of multiplying two kth degree polynomials. Since the encoding and decoding operations are similar, we will consider only the complexity of the decoding operation. We assume that all irreducible polynomials have the same degree d. The decoding operation for polynomials requires

$$O(r \log m_0 \cdot M_p(d) + \log r \cdot M(dr)) \text{ time,}$$

where $m_1 = m_2 = \ldots = m_r = m_0$. The first term represents the time required to execute step 7 of Algorithm 2. The second term gives the time required for combining r terms by Chinese Remaindering. (The calculation of the residues in step 6 of Algorithm 2 requires an equivalent time [1]). The decoding operation for primes requires

$$O\left[\sum_{i=1}^{r} \log m_i \cdot M_n(b_i) + \log r M_n(b)\right],$$

time, where b_i is the number of bits required to represent p_i and $b = \sum_{i=1}^{r} b_i$.

Cryptographic systems involving finite field transforms have been suggested in connection with public-key cryptography [10]. Rivest's system is similar to our prime coding system and its secrecy depends on the complexity of factoring the product of two very large primes. Our system, on the other hand, has been developed for conventional cryptographic use, in which the entire key is assumed to be inaccessible to the cryptanalyst and the key is changed periodically for security. In our system, the primes can be much smaller than those used by Rivest and this obviously reduces the complexity of encoding and decoding processes. In fact, under these circumstances, we have already shown (in the previous point (f)) that the use of irreducible polynomials is superior to the use of primes. Rivest's system cannot be readily extended to the polynomial case since there are polynomial-time algorithms for factoring a polynomial over a finite field [4], [5].

We should also like to mention here that our method of coding and decoding is more efficient than Rivest's method, which directly determines $M^n \bmod P$ for encoding (and $E^{m_0} \bmod P$ for decoding). To compare the complexities of the two methods, let us assume that b bits are required to represent P and b' bits are required to represent each p_i, $i = 1, 2, \ldots, r$. If the individual p_i's are of the same order of magnitude, then $b \simeq rb'$. We further assume that $n = \prod_{i=1}^{r} n_i$ (in practice n can be greater than, equal to or less than $\prod_{i=1}^{r} n_i$). Since the value of n_i can range from 1 to $\Phi(p_i)$, n requires, on the average, $b'/2$ bits for representation, i.e., $O(b')$ bits. Rivest's method for encoding requires $O(M_n(b) \cdot \log n) = t_1$ time while our method for primes requires $O(M_n(b') \cdot \log n_i + M_n(b) \cdot \log r) = t_2$ time, i.e.,

$$t_1 = O(b \cdot M_n(b)),$$

$$t_2 = O(b' \cdot M_n(b') + \log(r) \cdot M_n(b)).$$

Depending on which term dominates in t_2, we obtain the ratio t_1/t_2 as either $b \cdot M_n(b)/M_n(b') \cdot b'$ or $b/\log r$. Assuming $M_n(b)$ to be a linear function of b, we obtain $b \cdot M_n(b)/M_n(b') \cdot b' = r^2$. More reasonable assumptions for $M_n(b)$ such as $b \cdot \log b \cdot \log \log b$ or b^k, $1 < k \leqslant 2$, give a higher value for this ratio. Thus our method is clearly more efficient than Rivest's method. A similar consideration applies for the decoding procedure. The reason for this clear improvement in efficiency is not merely the use of modular arithmetic, but the fact that the exponent for each residue is much smaller than n. Hence, the number of multiplications for each residue is decreased, in addition to the decrease in size.

7. Conclusion

The congruence cipher presented in this paper allows the use of block-coding with an extremely small size for the associated key. This cipher will be very useful in a number of cases where pre-processing to remove the statistics of the message source is inconvenient or undesirable. An 8-character block-coding scheme using the English alphabet set can be constructed using 4 different irreducible polynomials of degree 2 over $GF(29)$. This system will have around 10^{11} different keys and an inexpensive microprocessor-based system can be easily developed to implement this scheme. Each key in this system contains only 9 elements (the value of n, and the coefficients of x and the constant term for the 4 irreducible polynomials). Hence the key can be changed often without inconvenience. The number of different keys can be increased by using polynomials of different degrees. It is expected that similar systems will find wide application in military communications and computer data-protection.

References

[1] Aho A V, Hopcropt J E and Ullman J D 1974 *The Design and analysis of computer algorithms.* (Reading, MA : Addison Wesley)
[2] Alanen J D and Knuth D E 1964 *Sankhya (Calcutta)* **A26** 305
[3] Albert A A 1956 *Fundamental concepts of higher algebra* (Chicago : University Press)
[4] Berlekamp E R 1967 *Bell Syst. Tech. J.* **46** 1853
[5] Berlekamp E R 1968 *Algebraic coding theory* (New York : McGraw-Hill)
[6] Berlekamp E R 1970 *Math. Computation* **24** 713
[7] Diffie W and Hellman M E 1976 *IEEE Trans. Inform. Theory* **22** 644
[8] Knuth D E 1969 *The art of computer programming, Vol.* 2, *Semi-numerical algorithms* (Reading, MA : Addison Wesely)
[9] Niven I and Zuckerman H S 1960 *An introduction to the theory of numbers* (New York : John Wiley)
[10] Rivest R L, Shamir A and Adleman L 1978 *Communications of the Association for Computing Machinery* **21** 120
[11] Shannon C E 1949 *Bell Syst. Tech. J.* **28** 656