Annales UMCS Informatica AI 5 (2006) 37-45

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Fast interpolating algorithms in cryptography

Joanna Kapusta[1*], Ryszard Smarzewski[1,2*]

[1]*Department of Mathematics and Computer Sciences, The John Paul II Catholic University of Lublin, Konstantynów 1H, 20-708 Lublin, Poland*
[2]*The University of Arts and Sciences, Wesoła 52, 25-353 Kielce, Poland*

Abstract

We present two fast polynomial interpolating algorithms with knots generated in a field K by the recurrent formula of the form $x_i = \alpha x_{i-1} + \beta \quad (i = 1,2,..,n-1;\ x_0 = \gamma)$. The running time of them is $C(n) + O(n)$ base operations from K, where $C(n) = O(n \log_2 n)$ denotes the time needed to compute the wrapped convolution in K^n. Moreover, we give an application of these algorithms to threshold secret sharing schemes in cryptography.

1. Introduction and preliminaries

Let $K^n = (K^n, +, \cdot)$ be an n-dimensional linear space of vectors

$$a = (a_0, a_1, ..., a_{n-1}), \quad a_i \in K$$

over a field $K = (K, +, \cdot)$ with a primitive root ψ from the unity of degree $2n$. Additionally, let the vector addition and scalar multiplication be defined in the usual coordinatewise way. In the same coordinatewise way we also define the vector subtraction, multiplication and division. For example, if $a = (a_0, a_1, ..., a_{n-1})$ and $b = (b_0, b_1, ..., b_{n-1})$ are two vectors in K^n, then we set

$$a/b = (a_0/b_0, a_1/b_1, ..., a_{n-1}/b_{n-1}), \quad a_i \in K,$$

where $a_i / b_i = a_i \cdot b_i^{-1}$ and $b_i \neq 0 \quad (i = 0,1,...,n-1)$. Moreover, we define the wrapped convolution

$$a \otimes b = (c_0, c_1, .., c_{n-1}),$$

where

$$a = (a_0, a_1, ..., a_{n-1}),\ b = (b_0, b_1, ..., b_{n-1})$$

*Corresponding authors: *e-mail addresses*: rsmax@kul.lublin.pl (R.Smarzewski), jkapusta@kul.lublin.pl (J.Kapusta)

and

$$c_i = \sum_{k=0}^{i} a_k b_{i-k} \quad (i = 0,1,...,n-1). \tag{1}$$

It is well known that wrapped convolutions can be computed by an algorithm, which has a running time of $O(n \log_2 n)$ base field operations in K. Such an algorithm is based on the following fundamental identity

$$a \otimes b = \left\{ F^{-1}\left[F(a) \cdot F(b) \right] + F^{-1}\left[F(\Psi \cdot a) \cdot F(\Psi \cdot b) \right] / \Psi \right\} / 2, \tag{2}$$

which is implicitly presented and used in [1]. In this formula we have $\Psi = (1, \psi, ..., \psi^{n-1})$. Moreover, discrete Fourier transformations $F = F_\omega : \mathrm{K}^n \to \mathrm{K}^n$ and $F^{-1} = F_\omega^{-1} : \mathrm{K}^n \to \mathrm{K}^n$ $(\omega = \psi^2)$ are defined by

$$\begin{gathered} b = F(a), \quad a = (a_0, a_1, .., a_{n-1}), \quad b = (b_0, b_1, .., b_{n-1}), \\ b_i = \sum_{k=0}^{n-1} a_k \omega^{ik} \quad (i = 0,1,..n-1) \end{gathered} \tag{3}$$

and

$$\begin{gathered} a_i = \frac{1}{n} \sum_{k=0}^{n-1} b_k \omega^{-ik} \quad (i = 0,1,...,n-1), \\ \frac{1}{n} = \left(\underbrace{1+1+...+1}_{n-items} \right)^{-1}, \quad 1 - \text{the unity in K}. \end{gathered}$$

Now the claim follows directly from the well known fact that Fourier transformations F and F^{-1} can be evaluated by the famous FFT-algorithm [1], which has a running time of $O(n \log_2 n)$ base operations from the field K.

For the completeness, we now present a simple proof of the formula (2). In order to do this, we recall that the coordinates c_k $(0 \le k < n)$ of $c = a \otimes b$ are identical with the corresponding coefficients of the polynomial product

$$c(x) = a(x) b(x), \quad x \in \mathrm{K}$$

with

$$a(x) = \sum_{k=0}^{n-1} a_k x^k, \quad b(x) = \sum_{k=0}^{n-1} b_k x^k$$

and

$$c(x) = \sum_{k=0}^{n-1} c_k x^k + x^n \sum_{k=0}^{n-1} c_{n+k} x^k, \quad c_{2n-1} = 0.$$

Since we have

$$c_i = \sum_{k=0}^{i} a_k b_{i-k} \quad \text{and} \quad c_{n+i} = \sum_{k=i+1}^{n-1} a_k b_{n+i-k} \tag{4}$$

we can use the auxiliary vectors $d, h \in \mathrm{K}^n$ with the coordinates defined by

$$d_i = c_i + c_{n+i} \quad \text{and} \quad h_i = c_i - c_{n+i} \quad (i = 0,1,...,n-1)$$

to get

$$d(\omega^i) = \sum_{k=0}^{n-1} d_k \omega^{ik} = \sum_{k=0}^{2n-1} c_k \omega^{ik} = \sum_{k=0}^{2n-1} \left(\sum_j a_j b_{k-j} \right) \omega^{ik} = a(\omega^i) \cdot b(\omega^i)$$

for $i = 0,1,...,n-1$. Clearly, this is equivalent to

$$d = F^{-1}\left[F(a) \cdot F(b) \right], \quad F = F_\omega .$$

On the other hand, one can apply the formulae $\psi^n = -1$ and (4) to obtain

$$\sum_{k=0}^{n-1} \psi^k h_k \omega^{ik} = \sum_{k=0}^{2n-1} \psi^k c_k \omega^{ik} = \sum_{k=0}^{2n-1} \left(\sum_j \psi^j a_j \psi^{k-j} b_{k-j} \right) \omega^{ik} ,$$

whenever $0 \le i < n$. Hence the division by $\Psi = (1, \psi, ..., \psi^{n-1})$ yields

$$h = F^{-1}\left[F(\Psi \cdot a) \cdot F(\Psi \cdot b) \right] / \Psi .$$

Finally, it remains to compute $(d + h)/2$ to finish the proof of the formula (2).

2. Fast interpolation with special knots

Let us suppose that the points $x_i \in \mathrm{K}$ $(i = 0,1,..,n-1)$ are pairwise distinct and that $y_i \in \mathrm{K}$ $(i = 0,1,...,n-1)$ are arbitrary. Then the unique interpolating polynomial $p(x)$ in the space $\mathrm{K}_{n-1}[x]$ of all polynomials of degree less than *n* is defined under the interpolating conditions

$$p(x_i) = y_i \quad (i = 0,1,...,n-1).$$

Moreover, it is given by the Newton interpolatingformula

$$p(x) = c_0 + c_1(x - x_0) + ... + c_{n-1}(x - x_0)(x - x_1)...(x - x_{n-2}) , \tag{5}$$

where the divided differences

$$c_i = [y_0, y_1, ..., y_i] \quad (i = 0,1,..,n-1)$$

can be computed by the usual recurrent formulae, which require $O(n^2)$ base operations from the field K. However, if the knots form an arithmetic or geometric progression, then the running time of the algorithm based on the recurrent formulae can be reduced to $O(n \log_2 n)$ [2].

More generally, let us suppose that pairwise distinct knots $x_0, x_1, ..., x_{n-1} \in \mathrm{K}$ are generated by the following recurrent formula

$$x_0 = \gamma,$$
$$x_i = \alpha x_{i-1} + \beta \quad (i = 1,2,..,n-1),$$

where $\alpha \neq 0, \beta, \gamma$ are fixed in K. Then one can insert

$$x_i = \alpha^i \gamma + \beta\left(\alpha^{i-1} + \alpha^{i-2} + \ldots + 1\right)$$

into the formula

$$c_i = \left[y_0, y_1, \ldots, y_{i-1}\right] = \sum_{j=0}^{i} \frac{y_j}{\prod\limits_{\substack{k=0 \\ k \neq j}}^{i} \left(x_j - x_k\right)} \qquad \left(i = 0,1,..,n-1\right)$$

and use the identity

$$\alpha^{j-k} - 1 = \left(\alpha - 1\right)\left(\alpha^{j-k-1} + \alpha^{j-k-2} + \ldots + 1\right)$$

in order to get

$$c_i = \left(\sum_{j=0}^{i} p_j q_{i-j}\right) \Big/ r_i \qquad \left(i = 0,1,..,n-1\right)$$

or equivalently

$$c = \left(p \otimes q\right) / r \,,$$

where

$$p_j = \frac{y_j}{\prod\limits_{k=0}^{j-1} \sum\limits_{m=0}^{k} \alpha^k}, \qquad q_j = \frac{\left(-1\right)^j \prod\limits_{k=0}^{j-1} \alpha^k}{\prod\limits_{k=0}^{j-1} \sum\limits_{m=0}^{k} \alpha^m}$$

and

$$r_j = \left[\left(\alpha - 1\right)\gamma + \beta\right]^j \prod_{k=0}^{j-1} \alpha^k \qquad \left(j = 0,1,\ldots,n-1\right).$$

Here and in the following it is assumed that products and sums are equal to 1, whenever their upper indices are smaller than the lower. Furthermore, note that the coordinates

$$s_k = \sum_{m=0}^{k-1} \alpha^m \qquad \left(k = 1,2,\ldots,n-1\right)$$

of vector $s = \left(s_0, s_1, \ldots, s_{n-1}\right)$ satisfy the following recurrent formula

$$s_k = s_{k-1} \cdot \alpha + 1 \qquad \left(k = 1,2,\ldots,n-1; \quad s_0 = 0\right).$$

Hence we get Algorithm 1 to compute the required divided differences in the Newton interpolating formula. This algorithm uses two classes `KType` and `KTypeVector`, which enable to perform operations in K and K^n.

It is clear that Algorithm 1 has a running time of $C\left(n\right) + O\left(n\right)$ base operations from the field K, where $C(n)$ denotes the time needed to compute the wrapped convolution

$$conv\left(p, q\right) = p \otimes q \,.$$

```
INPUT: KTypeVector y; KType a, b, g;
OUTPUT: KTypeVector c;
   KTypeVector p, q, r;
   KType d=(a–1)*g+b, s=0,
         v=1/a, u=1, z=1;
   p[0]=y[0]; q[0]=1; r[0]=1;
   for(int k=1; k<c.length(); k++){
      s=s*a+1; u=u*s; p[k]=y[k]/u;
      v=v*a; z=-z*v; q[k]=z/u;
      r[k]=r[k-1]*v*d;
   }
   c=conv(p,q)/r;
```

Algorithm 1. The divided differences with knots $x_i = ax_{i-1} + b \quad (i = 1,2,...,n-1; \quad x_0 = g)$

3. Fast evaluation of Newton polynomial at special knots

In this section we present the inverse algorithm to Algorithm 1, which is useful in the threshold secret sharing scheme [3,4]. More precisely, we consider the problem of fast computation of the polynomial

$$p(x) = c_0 + c_1(x - x_0) + ... + c_{n-1}(x - x_0)(x - x_1)...(x - x_{n-2})$$

at the knots

$$x_i = \alpha x_{i-1} + \beta \quad (i = 1,2,..,n-1; x_0 = \gamma).$$

For this purpose, we insert

$$x_i = \alpha^i \gamma + \beta\left(\alpha^{i-1} + \alpha^{i-2} + ... + 1\right)$$

into the formula (5) to derive

$$y_i = \sum_{j=0}^{i} c_j (x_i - x_0)(x_i - x_1)...(x_i - x_{j-1})$$

$$= \left(\sum_{j=0}^{i} p_j q_{i-j}\right) \cdot r_i \quad (i = 0,1,...,n-1)$$

or equivalently

$$y = (y_0, y_1, ..., y_{n-1}) = (p \otimes q) \cdot r,$$

where

$$p_j = c_j \left[(\alpha - 1)\gamma + \beta\right]^j \prod_{k=0}^{j-1} \alpha^k$$

and

$$\frac{1}{q_j} = r_j = \prod_{k=0}^{j-1} \sum_{v=0}^{k} \alpha^v .$$

These identities immediately yield Algorithm 2, which is an inversion of Algorithm 1.

```
INPUT: KTypeVector c, KType a, b, g;
OUTPUT: KTypeVector y;
   KTypeVector p, r;
   KType d=(a–1)*g+b, s=0,
            v=1/a, w=1;
   p[0]=c[0]; r[0]=1;
   for(int k=1; k<c.length(); k++){
      v=v*a; w=w*v*d; p[k]=c[k]*w;
      s=s*a+1; r[k]=r[k-1]*s;
   }
   y = conv(p,1/r)*r;
```

Algorithm 2. Newton's polynomial evaluation at the knots
$x_i = ax_{i-1} + b \quad (i = 1,2,...,n-1; \quad x_0 = g)$

The running time of this algorithm is again $C(n) + O(n)$ base operations from the field K.

4. An application to secret sharing schemes

Let us consider a secret sharing scheme of Shamir type [3,4] with respect to a Newton interpolating polynomial in $\mathrm{K}_{n-1}[x]$, where $\mathrm{K} = Z_{65537}$ is a field of integers modulo 65537 and $n = 128$ is an artificially small number. More precisely, let us suppose that a dealer does the following:

- in a random way chooses numbers $\alpha = 6$, $\beta = 2257$, $\gamma = 528$, $\psi = 62750$ and a polynomial
$$p(x) = 578 + 3452(x - x_0) + 218(x - x_0)(x - x_1)...(x - x_{122}) + \\ +947(x - x_0)(x - x_1)...(x - x_{123}) + 5321(x - x_0)(x - x_1)...(x - x_{126}),$$
where $x_0 = 528$ and $x_i = 6x_{i-1} + 2257 \quad (i = 1,2,...,127)$;
- defines the polynomial key $\kappa = 57834522189475321$, which guarantees opening an access gate to the secret;
- applies Algorithm 2 with $a = \alpha$, $b = \beta$ and $g = \gamma$ to compute the shares $y_i = p(x_i) \quad (i = 0,1,...,127)$ and distributes a fixed number (say 6) of the following shares

$(x_0, y_0) = (528, 578)$, $(x_1, y_1) = (5425, 62013)$, $(x_2, y_2) = (34807, 37401)$,
$(x_3, y_3) = (14488, 20803)$, $(x_4, y_4) = (23648, 52289)$, $(x_5, y_5) = (13071, 44594)$

among participants of the secret sharing scheme, and the remaining 122 shares (x_i, y_i) $(6 \le i < 128)$ together with n, α, β, γ and ψ gives to the combiner.

Then the combiner can recover the key κ and get the required access after receiving all shares from the participants of the secret sharing. For this purpose, he has only to apply Algorithm 1 in order to show that the nonzero coefficients of the polynomial

$$p(x) = \sum_{i=0}^{127} c_i \prod_{k=0}^{i-1} (x - x_k)$$

are equal to

$$c_0 = 578, \quad c_1 = 3452, \quad c_{123} = 218, \quad c_{124} = 947, \quad c_{127} = 5321.$$

Consequently, he recovers the key

$$\kappa = 57834522189475321.$$

It is important that the secret sharing scheme can be changed in such a way that combiner can check if shares are falsified, and consequently can verify authenticity of recovered key. For this purpose the dealer should be allowed to generate more shares than $n = 128$ with respect to pairwise distinct interpolating knots x_i $(i = 0,1,..,m-1; \quad m > 128)$. Then the combiner ought to verify that

$$c_{128} = c_{129} = ... = c_{m-1} = 0.$$

Finally, we note that the dealer and combiner can save about 99 percent of base operations in the field K, whenever the dealer chooses n of order 2^{14}. Indeed, if $n = 2^{14}$ then Algorithm 1 requires 5799888 base operations in the field K. On the other hand, the classical algorithm based on usual recurrent formulae uses 402628608 such operations. Hence the number of based operations in Algorithm 1 equals 1.44% of the number of operations in the classical algorithm.

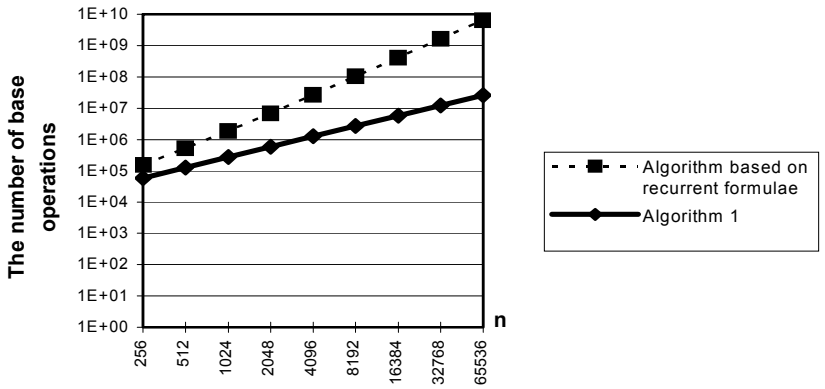

Fig. 1. The numbers of base operations in the classical recurrent algorithm and Algorithm 1

Further, if $n = 2^{16}$ then the dealer saves

$$\frac{6416531501}{6442352640} * 100\% = 99.6\%$$

of base operations in the field K, whenever he uses Algorithm 2 instead of the extended Horner algorithm. Moreover, the graphs of numbers of base operations are presented in Figures 1 and 2 for these four algorithms. In view of the huge difference between the numbers of operations in the corresponding algorithms, we use the logarithmic scale in these graphs.

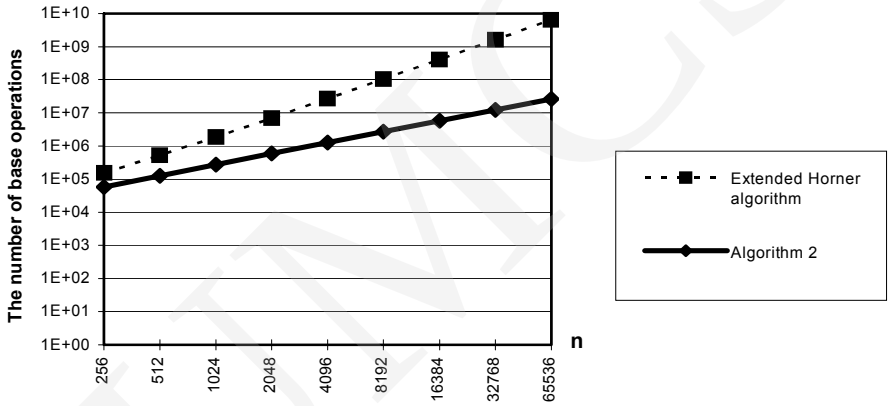

Fig. 2. The numbers of base operations in the extended Horner algorithm and Algorithm 2

Conclusions

In this paper, we have presented fast polynomial interpolating and evaluating algorithms in the case of n knots generated dynamically in a field K by the recurrent formula of the form

$$x_i = \alpha x_{i-1} + \beta \quad (i = 1,2,..,n-1;\ x_0 = \gamma).$$

The first algorithm computes divided differences with the running time of $C(n) + O(n)$ base operations from the field K, where $C(n) = O(n \log_2 n)$ denotes the complexity of computation of the wrapped convolution in K^n. In comparison, the classical algorithm using usual recurrent formulae requires $O(n^2)$ of base operations in K. The second algorithm evaluates Newton polynomial at n knots given by the above recurrent formula in the same time as the first one. Numerical experiments show that these both algorithms can be useful in practice, whenever n is sufficiently large. For example, such a situation occurs during the computation of shares and recovering keys in the secret sharing schemes of Shamir type [4].

References

[1] Aho A.V., Hopcroft J.E., Ullman J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, London, (1974).

[2] Bostan A., Schost E., *Polynomial evaluation and interpolation on special sets of points*, Journal of Complexity, 21(4) (2005) 420.

[3] Menezes A.J., van Oorschot P.C., Vanstone S.A., *Handbook of Applied Cryptography*, CRC Press, New York, (1997).

[4] Smarzewski R., Kapusta J., *Algorithms for multi-secret hierarchical sharing schemes of Shamir type*, Annales UMCS Informatica, AI3 (2005) 65.