Annales UMCS Informatica AI 3 (2005) 65-91

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

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

Algorithms for multi-secret hierarchical sharing schemes of shamir type

Ryszard Smarzewski*, Joanna Kapusta

Department of Mathematics and Computer Science, Catholic University of Lublin, John Paul II, Konstantynów 1H, 20-708 Lublin, Poland

Abstract

In this paper there are presented algorithms for multilevel hierarchical threshold secret sharing schemes based on the interpolation of Hermite type, which use either traditional Shamir's keys or polynomial and orthogonal polynomial keys. These algorithms enable to compute the probability of authenticity of shares during the process of recovering the keys. In addition, two models of secret sharing are considered, which enlarge their security against attacks and decrease their rate of computations by applying FFT-algorithm.

1. Introduction

The first secret sharing schemes were introduced by Shamir [1] and Balkley [2]. Several other models of secret sharing and their characteristic properties were studied in recent papers e.g. [3-6], and monographs [7-9]. In Shamir's scheme the shares of the secret key

$$K = w(0) = a_m$$

have the form

$$(x_0, y_0), (x_1, y_1), ..., (x_n, y_n),$$

where knots $x_0, x_1, ..., x_n$ are pairwise disjoint nonzero points of a field F, $y_i = w(x_i)$ and

$$w(x) = a_0 x^m + a_1 x^{m-1} + ... + a_m, \quad m \le n$$

is a polynomial, which is randomly chosen by a dealer. In order to recover the key K, the Lagrange interpolation formula based on any $r+1$, $m \le r \le n$ shares has been always proposed [1,7-9].

In this paper, we present hierarchical multilevel models of secret sharing based on the Hermite type interpolating polynomials p of degree less or equal to

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

r. In the very particular case $r = n$, these polynomials are uniquely determined by the following conditions

$$\frac{p^{(k_i)}(x_i)}{k_i!} = y_i, \quad i = 0,1,...,n,$$

where values $y_i = w^{(k_i)}(y_i)/k_i!$ can be easily computed by the extended Horner algorithm, and

$$k_i = card\{\ k : x_{i-k} = x_i, k \geq 1\}.$$

Recovering of the key *K* requires now using either the Newton interpolating formula or the Neville recurrent interpolating formula, which have several additional features in comparison with the Lagrange type interpolating formula, even in the case of pairwise disjoint knots considered by A. Shamir. The corresponding theory and algorithms, written in the ISO/ANSI standard C++ [10], are given in Sections 1, 2, 3 and 4. It should be noted that all our algorithms are modeled in such a way that they are as simple, short and concise as possible.

In particular, in Section 3 we establish a new model of multilevel secret sharing, which admits receiving *c* shares by the combiner and remaining $r - c + 1$ shares by the participants of secret sharing. This enables to keep the number $r - c + 1$ of participants being small in comparison to degree *m* of the polynomial $w(x)$ and to take a sufficiently large number *c* in order to increase simultaneously the degree *m* and the probability

$$P_{aut} = 1 - \frac{1}{\left[card(F)\right]^{r-m} - 1}$$

of the authorization of $r+1$ shares, where $card(F)$ denotes the number of elements of a finite field *F*.

A further generalization of secret sharing models is established in Section 5. More precisely, instead of Shamir's key $K = a_m$ there is used the polynomial key of the form

$$K = a_{i_0} a_{i_1} ... a_{i_s}, \ \ 0 \leq i_0 \leq i_1 \leq ... \leq i_s \leq m$$

where $a_{i_0}, a_{i_1}, ..., a_{i_s}$ include all nonzero coefficients of polynomial $w(x)$. This suggests that a general model of secret sharing of Shamir type can be described as follows:

– A dealer chooses polynomial key $K = a_{i_0} a_{i_1} ... a_{i_s}$, which consists of all nonzero coefficients of the expansion

$$w(x) = \sum_{k=0}^{m} a_k p_k(x),$$

with respect to a base $p_0(x), p_1(x), .., p_m(x)$ of all polynomials P_m of degree not greater than *m*.

– The dealer uses Algorithm A to compute shares $b_0, b_1, ..., b_m$, which are identical to the coefficients of the expansion

$$w(x) = \sum_{k=0}^{m} b_k g_k(x)$$

with respect to another base $g_0(x), g_1(x), .., g_m(x)$ of P_m.

– A combiner gathers these shares and uses Algorithm B, inverse to Algorithm A, in order to recover the polynomial key $K = a_{i_0} a_{i_1} ... a_{i_s}$.

In Section 6, this general threshold secret sharing model is used in the particular case when $p_0(x), p_1(x), .., p_m(x)$ is a base of arbitrary monic orthogonal polynomials and $g_0(x), g_1(x), .., g_m(x)$ is the Newton base which occurs in the extended Newton interpolation formula. In this case, the corresponding Algorithms A and B are extended Clenshaw and inverse Clenshaw algorithms. Finally, in Section 7 we discuss the usefulness of the famous FFT algorithm in the threshold secret sharing schemes for finite rings with primitive roots of unity.

2. Hierarchy in secret sharing schemes

Let $F = (F, +, \cdot)$ be a finite, or an infinite field, and let *n* be a positive integer such that $n < card(F) - 1$. Additionally, let the knots of interpolation $x_i \in F \setminus \{0\}$, $i = 0,1,..,n$, satisfy the following condition

$$\underset{0 \le i < j \le n}{\forall} \left(x_i = x_j \Rightarrow x_i = x_{i+1} = ... = x_j \right) \tag{1}$$

and let the values $y_i \in F$ $(i = 0,1,...,n)$ be prescribed. Then a modified interpolating Newton formula of the form

$$p(x) = \langle y_0 \rangle + \langle y_0, y_1 \rangle (x - x_0) + ... + \langle y_0, y_1, ..., y_n \rangle (x - x_0)(x - x_1)...(x - x_{n-1}) \tag{2}$$

defines a unique interpolating polynomial of Hermite type, i.e. a unique polynomial of degree less or equal to *n*, which satisfies the interpolating conditions

$$\frac{p^{(k_i)}(x_i)}{k_i!} = y_i, \quad i = 0,1,...,n, \tag{3}$$

where

$$k_i = \max \{ \; k : x_{i-k} = x_i \}, \tag{4}$$

is a left-hand side multiplicity of the knot x_i. In this Newton formula, the symbols

$$\langle y_0, y_1, ..., y_j \rangle, \quad j = 0,1,...,n,$$

denote modified divided differences with respect to the knots $x_0, x_1, ...x_n$, which are defined by the following recurrent formulae $\langle y_i, y_{i+1}, ..., y_{i+k} \rangle = \begin{cases} y_{i+k-k_i}, \\ \dfrac{\langle y_{i+1}, y_{i+2}, ..., y_{i+k} \rangle - \langle y_i, y_{i+1}, ...}{x_{i+k} - x_i} \end{cases}$

where $0 \le i, \quad i+k \le n$. Let us note that the number of occurrences of a knot x_i among $x_0, x_1, ...x_n$, i.e. the multiplicity of the knot x_i is equal to

$$n_i = \max\left\{k + p + 1 : x_{i-k} = x_{i+p}\right\}.$$

Clearly, we always have $n_i > k_i$ and $n_i = k_s + 1$ for some $s \ge i$.

Note that such an interpolating polynomial exists and is unique. This follows from the fact that interpolating conditions (3) yield a linear system of $n+1$ equations with $n+1$ unknowns, which has the matrix of extended Vandermonde type. One can prove that its determinant [11] is equal either to 1, or to

$$\prod_{(i,j)\in\Lambda} (x_i - x_j)$$

in the case when

$$\Lambda := \left\{(i, j) : 0 \le j < i \le n,\ x_i \ne x_j\right\} \ne \varnothing .$$

Therefore, it is distinct from zero in the field *F*. This finishes the proof of existence and uniqueness of the polynomial of Hermite type. It should be noticed that the choice of interpolating conditions (3), instead of the usual interpolating Hermite conditions of the form

$$p^{(k_i)}(x_i) = y_i, \quad i = 0,1,...,n,$$

enables to diminish the computational complexity of the first two algorithms presented below. More precisely, it enables to omit the multiplications and divisions by $j!(j \le k_i)$ in these algorithms.

Now we can introduce a hierarchicalsecret sharing scheme with a random key $K \in F$. In this model the dealer *D* chooses randomly knots $x_0, x_1, ..., x_n$ from the set $F \setminus \{0\}$, and uses the extended Horner algorithm to compute the values

$$y_i = \frac{w^{(k_i)}(x_i)}{k_i!}, \quad i = 0,1,..,n, \tag{6}$$

where

$$w(x) = \sum_{k=0}^{m} a_k x^{m-k}, \quad m \le n \tag{7}$$

is a polynomial of degree *m* with randomly chosen coefficients $a_0, a_1, ..., a_{m-1}$, $a_m \in F$ such that $a_m = K$. Next, the dealer *D* distributes the key*K* among $n+1$ participants of the secret sharing. More precisely, he gives them shares of the form

$$U_i = (k_i, x_i, y_i), \quad i = 0,1,..,n,$$

where it is assumed that the multiplicities n_i of knots x_i satisfy the following restrictions

$$n_i \le m+1,$$

which guarantee that all shares are essential In order to do it, the dealer computes values

$$y[i] := y_i \quad (\mathrm{i} = 0,1,\ldots,\mathrm{n})$$

by applying Algorithm 1.

```
void Horner(FType w[], int m, FType x[], int n, FType
y[])
{
   FType * b = new FType [m+1];
   int i = 0, mi;
   FType xi;
   while(i<=n)
   {
       for(int j=0; j<=m; j++)
           b[j]= w[j];
       mi=m; xi=x[i];
       while(i<=n && x[i]==xi)
       {
           for(int j=1; j<=mi; j++)
              b[j]= b[j-1]*xi+b[j];
           y[i]= b[mi];
           i++; mi--;
       }
   }
}
```

Algorithm 1. The function Horner

We note that the class type `FType` should contain not only elements of the field F but also operations of addition, multiplication and inversion in F,which have to be denoted in the traditional way. In particular, all numerical examples presented in our paper use the library NTL [12], which contains classes connected with the field Z_p of integers modulo a prime p.

One can also consider the model of secret sharing with priorities, which require the following shares

$$U_i = \left(x_i, y_i^{(0)}, y_i^{(1)}, ..., y_i^{(n_i-1)}\right), \quad i = 0,1,...,\mathrm{s} \tag{8}$$

where it is assumed additionally that $x_i \neq x_j$ for $i \neq j$ and

$$n := \sum_{j=0}^{s} n_j \geq m, \quad y_i^{(j)} = \frac{w^{(j)}(x_i)}{j!}.$$

It is clear that Algorithm 1 can be also applied to compute the coordinates

$$y_i^{(0)}, y_i^{(1)}, ..., y_i^{(n_i-1)}$$

in this new case. Moreover, we note that the last model is not democratic, except the case when

$$n_0 = n_1 = ... = n_s .$$

2. Newton method of recovering the secret

In order to recover the key $K = w(0)$, the combiner C must receive $r+1$ shares

$$U_{i_j} = \left(k_{i_j}, x_{i_j}, y_{i_j}\right), \quad j = 0,1,...,\mathrm{r}, \tag{9}$$

where $m \leq r \leq n$ and $0 \leq i_0 < i_1 < ... < i_r \leq n$. Moreover, these shares must satisfy the conditions typical of the Hermite interpolation, which were presented in the previous section. More precisely, we should $k_{i_0} = 0$, and

$$i_{j-1} = i_j - 1, \quad i_{j-2} = i_j - 2, ..., \quad i_{j-k_{i_j}} = i_j - k_{i_j}, \tag{10}$$

whenever $k_{i_j} > 0$. Thus our model of secret sharing is hierarchical. In other words, this means that each holder of the share $U_{i_j} = \left(k_{i_j}, x_{i_j}, y_{i_j}\right)$, $j > 0$, can take part in recovering the secret only, if his superiors, which have the shares

$$U_{i_j-1}, \ U_{i_j-2}, ..., \ U_{i_j-k_{i_j}},$$

do the same thing.

Without the loss of generality, one can suppose that indices $i_0, i_1, ..., i_r$ are equal to those $0,1,...,r$, respectively. Then, in order to recover the key $K = w(0)$, it is necessary to compute first the modified divided differences

$$z[k] = \langle y_0, \ y_1, ..., y_k \rangle, \quad k = 0,1,...,r,$$

defined by recurrent formula (5). It is obvious that

$$z[k] = 0, \quad k = m+1, \quad m+2, .., r.$$

These identities can be applied to verify, whether shares $U_0, U_1, ..., U_r$ are not falsified, i.e. whether there is not any share with the third coordinate distinct from the values or the derivatives of the polynomial $w(x)$.

Let us note, that the probability P_{aut} of such an authorization of the authenticity of $r+1$ shares from a finite field with $card(F)$ elements, received by the combiner, is given by the formula

$$P_{aut} = 1 - \frac{1}{\left[card\left(F\right)\right]^{r-m} - 1}.$$

On the other hand, the probability P_{rec} of random recovering of the key K is equal to

$$P_{rec} = \frac{1}{\left[card\left(F\right)\right]^{m}}.$$

Now we present Algorithm 2, which computes the modified divided differences $z[k]$ $(k = 0,1,...,r)$. For this purpose, we assume that the shares

$$k_i = k[i], \quad x_i = x[i], \quad y_i = y[i] \quad (i = 0,1,...,r)$$

are given, and use an additional one dimensional array b to remember the divided differences

$$b[k] = \langle y_k, y_{k+1}, ..., y_j \rangle, \quad k = 0,1,...,j,$$

which are necessary in the loop with respect to $j = 0,1,..,r$.

```
for(j = 0; j <= r; j++)
{
   for(i = j; i >= j - k[j]; i--)
         b[i] = y[2*j - k[j] - i];
   for(i = j - k[j] - 1; i >= 0; i--)
        b[i] = (b[i+1]-b[i])/(x[j]-x[i]);
   z[j] = b[0];
}
```

Algorithm 2

Next, in order to derive the key

$$K = w(0) = z[0] - z[1]x_0 + ... + (-1)^r z[r]x_0x_1...x_{r-1}$$

one can use Horner algorithm, presented in Algorithm 3.

```
K = z[r];
for(i = r-1; i >= 0; i--)
        K = z[i]-K*x[i];
```

Algorithm 3

Example 1. Suppose that $F = Z_{37}$ is a field of integers modulo 37, and that the dealer D took the key $K = 23$, the polynomial

$$w(x) = x^3 + 2x + 23$$

knots of interpolation

$$x_0 = x_1 = 11, \quad x_2 = x_3 = x_4 = 36,$$

and the integer $r = 4$. Next, suppose that he has applied Algorithm 1 to compute the coordinates

$$y_0 = 7, \quad y_1 = 32, \quad y_2 = 20, \quad y_3 = 5, \quad y_4 = 34,$$

of the shares, and distributed among members of secret sharing the following shares

$U_0 = (0,11,7), \quad U_1 = (1,11,32), \quad U_2 = (0,36,20), \quad U_3 = (1,36,5), \quad U_4 = (2,36,34)$.

In order to recover the key, we assume that the combiner *C* receives all these shares. Then he can use Algorithm 2 to compute the modified divided differences $z = [7,32,21,1,0]$. Hence he can notice that the last divided difference is equal to zero, which implies that the probability of authenticity of obtained shares is equal to

$$P_{aut} = 1 - \frac{1}{36} = 0.97233\ldots.$$

Finally, he observes that

$$w(x) = 7 + 32(x-11) + 21(x-11)^2 + (x-11)^2(x-36),$$

and uses Algorithm 3 to recover the key $K = w(0) = 23$ with the same probability as above.

3. The Neville method of recovering the secret

If knots of interpolation $x_0, x_1, \ldots, x_r$ are pairwise disjoint, then the key $K = w(0)$ can be derived from the well-known algorithm due to Neville [13]. It is interesting, that the same is true without any additional assumption. For this purpose, we define partial interpolating polynomials

$$p_{j,s}(x), \quad j,s \geq 0, \quad j+s \leq r,$$

of degree less or equal to *s*, which satisfy the following interpolating conditions

$$\frac{p_{j,s}^{(j_i)}(x_i)}{j_i!} = y_{i-k_i}, \quad i = j, j+1, \ldots, j+s \tag{11}$$

where the order of the derivative is defined by the formula

$$j_i = \min\{k_i, i-j\}$$

Hence it follows directly that

$$p_{j,0} = y_{j-k_j}, \quad w(x) = p_{0,r}(x) = p_{0,m}(x)$$

and

$$p_{j,s}(x) = \sum_{l=0}^{s} y_{j-k_j+l} \left(x - x_{j-k_j}\right)^l,$$

whenever $x_j = x_{j+s}$, i.e. if $s \le k_{j+s}$. On the other hand, if $x_j \neq x_{j+s}$, then the following recurrent formula of the Neville type

$$p_{j,s}(x) = p_{j+1,s-1}(x) + \frac{p_{j+1,s-1}(x) - p_{j,s-1}(x)}{x_{j+s} - x_j}\left(x - x_{j+s}\right),$$

is satisfied. Since the proof of this formula is similar to the proof of the classical interpolating formula of Neville [13], we omit the proof. Instead, we only note that, in view of the unicity of Hermite interpolation, it is sufficient to use induction to show that the right-hand side of the last formula satisfies the interpolating conditions (11).

Now, if we have $0 \le j \le r$, then we denote

$$b[i] = p_{i,j-i}(0), \quad i = 0,1,...,j,$$

and conclude that

$$b[i] = b[i+1] + y[2j - k_j - i]\left(-x_{i-k_j}\right)^{j-i}, \quad i = j-1, j-2, ..., j-k_j,$$

and

$$b[i] = b[i+1] - \frac{b[i+1] - b[i]}{x_j - x_i}x_j, \quad \mathrm{i} = \mathrm{j} - \mathrm{k_j} - 1, \mathrm{j} - \mathrm{k_j} - 1, ..., 0,$$

where

$$b[j] = y[j - k_j]$$

Hence it follows directly that the combiner can apply Algorithm 4 to the data

$$k_i = k[i], \quad x_i = x[i], \quad y_i = y[i] \quad (i = 0,1,...,r),$$

in order to get the key $K = w(0)$, which is identical to the last value of $b[0]$.

```
for(j = 0; j <= r; j++)
 {
     b[j] = y[j - k[j]];  d[j]=1;
     for(i = j-1; i >= j - k[j]; i--)
        {
           d[i] = -d[i]*x[i];
           b[i] = b[i] + y[2*j-i-k[j]]*d[i];
        }
        for(i = j - k[j] - 1; i >= 0; i--)
           b[i]   =   b[i+1]-x[j]*(b[i+1]-b[i])/(x[j]-
x[i]);
 }
K = b[0];
```

Algorithm 4

Clearly, one can finish calculations in Algorithm 4 in the case, when $j = m+1$. This follows from the fact that value of $b[0]$, which is computed for $j = m$, remains unchanged in the next cycles of the loop for

$$j = m+1, m+2, ..., r,$$

which is a simple consequence of the uniqueness of interpolation of Hermite type. However, this property is useful in verification of authenticity of received shares by the combiner. The probability of this verification is given by the

$$P_{aut} = 1 - \frac{1}{\left[card(F)\right]^{r-m} - 1}.$$

In order to illustrate the recovering of the key by Neville method, we can apply Algorithm 4 to the data from Example 1 and get the following vector

$$b = [23, 23, 22, 25, 20],$$

which implies again that $K = 23$. In this case, the probability of authenticity of received shares and the key is equality of the first two coordinates $p_{0,4}(0)$ and $p_{1,3}(0)$ of vector *b* defined by

$$b[i] = p_{i,4-i}(0), \quad i = 0,1,...,4.$$

This probability is equal to

$$p_{aut} = 0.97233...$$

4. Increasing of the security of secret sharing

The assumption $n \geq m$ about the number $n+1$ of participants of secret sharing and about the order *m* of polynomial $w(x)$, can decrease drastically the degree of the polynomial, which simplifies the complexity of the problem of secret sharing and diminish its security. It seems to be a serious drawback. For example, if we admit only two participants $(n=1)$, then the polynomial $w(x)$ has to be a linear function, which is not secure against attacks [9]. This problem can be overc0me if we assume, in the case when $n < m$, that the combiner *C* can receive at least $m-n$ additional shares from the dealer *D*, which together with remaining shares should satisfy conditions (10). Let us note, that this assumption makes the secret sharing democratic, since every $r+1$ share, obtained from the participants of secret sharing, can be extended in an obvious way to $\tilde{r}+1$ share, which satisfies conditions (10) and inequality $\tilde{r} \geq m$.

On the other hand, it seems to be hard to justify the possibility of communicating between the combiner and the dealer during recovering the secret. The reason is that such a communication can diminish significantly the security of the whole process. Therefore, the most interesting solution of the problem seems to be the model of secret sharing which admits the possibility of

simultaneous distributing shares of the key $K = w(0)$, computed by using of Algorithm 1, between the combiner and the participants. More precisely, the combiner should receive *c* shares of the form

$$U_i = (k_i, x_i, y_i), \quad i = 0,1,..,c-1,$$

and participants should get $n-c+1$ shares

$$U_i = (k_i, x_i, y_i), \quad i = c, c+1,...,n,$$

where $0 \le c \le m \le n$. Clearly, interpolating knots x_i of multiplicity $n_i \le m+1$ ought to satisfy the assumption

$$\{x_0, x_1, ..., x_{c-1}\} \cap \{x_c, x_{c+1}, ..., x_n\} = \varnothing .$$

The recovering of the key $K = w(0)$ in this model is possible, if the combiner *C* will receive $r-c+1$ shares

$$U_{i_j} = (k_{i_j}, x_{i_j}, y_{i_j}), \quad j = 0,1,...,r-c \tag{12}$$

from $n-c+1$ participants, where

$$m \le r \le n, \ c \le i_0 < i_1 < ... < i_{r-c} \le n \tag{13}$$

and

$$i_{j-1} = i_j - 1, \quad i_{j-2} = i_j - 2, ..., \quad i_{j-k_{i_j}} = i_j - k_{i_j} . \tag{14}$$

Without loss of generality, the combiner can simplify notations by setting

$$c, c+1, ..., r \text{ instead of } i_0, i_1, ..., i_{r-c},$$

and next can recover the key $K = w(0)$ by using Algorithms 2 and 3.

```
void DivDif(int k[], FType x[],FType y[],FType z[],
 FType b[], int c, int r)
{
    for(int j = c; j <= r; j++)
    {
        b[j] = y[j-k[j]];
        for(int i = j - 1; i >= j - k[j]; i--)
               b[i] = y[2*j - k[j] - i];
        for(int i = j - k[j] - 1; i >= 0; i--)
              b[i] = (b[i+1]-b[i])/(x[j]-x[i]);
       z[j] = b[0];
    }
}
```

Algorithm 5. The function DivDif

It should be noted that the combiner *C* can use Algorithm 2 to compute the following modified divided differences

$$z[k] = \langle y_0, y_1, ..., y_k \rangle \text{ and } b[k] = \langle y_k, y_{k+1}, ..., y_{c-1} \rangle, \quad k = 0,1,...,c-1,$$

as soon as he receives his part of shares. Next he can use them during the recovering of several $K = w(0)$. In order to do this, it is convenient to rewrite Algorithm 2 as a function `DivDif`, where the type `FType` is defined in the same way as in Algorithm 1 from Section 2.

Now, we can call twice the functions

```
DivDif(x,y,z,b,0,c-1);
DivDif(x,y,z,b,c,r);
```

and apply Algorithm 3 to get the key $K = w(0)$.

In our model of secret sharing, it is convenient to use Neville algorithm, written as a function called `Neville`.

```
void Neville(int k[],FType x[],FType y[],FType d[],
FType b[], int c, int r, FType K)
{
   FType d* = new FType[r+1];
   for(int j = c; j <= r; j++)
   {
        b[j] = y[j-k[j]];d[j]=1;
        for (int i = j-1; i >= j - k[j]; i--)
   {
            d[i] = -d[i]*x[i] ;
            b[i] = b[i] + y[j+j-i-k[j]]*d[i];
        }
        for (int i = j - k[j] - 1; i >= 0; i--)
            b[i]   =   b[i+1]-(b[i+1]-b[i])*x[j]/(x[j]-
x[i]);
    }
    K = b[0];
}
```

Algorithm 6. The function Neville

In this case, we can call twice the functions

```
Nevil(k,x,y,d,b,0,c-1,K);
Nevil(k,x,y,d,b,c,r,K);
```

to get the key $K = b[0]$.

Note that the last model of secret sharing can not be programmed effectively, whenever one wants to use the Lagrange interpolating formula instead of the formulae of Newton or Neville. This is true even in the case, when

$$k_i = 0, \quad i = 0,1,...,n,$$

which was considered by Shamir [1]. It follows directly from the fact that the Lagrange interpolating formula is not useful to apply the previous results in order to prolong computations for additional shares.

Example 2. Let us suppose that the dealer D chooses the key $K = 4083 \in Z_{8761}$, and next takes $n = r = 13$ and the polynomial

$$w(x) = 205x^{11} + 89x^9 + 503x^6 + 1223x^3 + 341x + 4803 .$$

Moreover, let us suppose that the knots of interpolation are equal to

$$x_0 = x_1 = ... = x_6 = 523, \quad x_7 = x_8 = ... = x_{10} = 2365, \quad x_{11} = x_{12} = x_{13} = 6543 .$$

After computing via Algorithm 1 of the following values

$$y_i = \frac{w^{(k_i)}(x_i)}{k_i!} \quad (0 \le i \le 13),$$

the dealer distributes $c = 7$ and $n - c + 1 = 7$ shares between the combiner and participants of secret sharing, respectively. Then the combiner can use his shares to compute by Algorithm 5 the following initial divided diference

$$z = [7993,5082,3667,2417,420,721, 468]$$

and

$$b = [7993,5082,3667,2417,420,721,468],$$

in the case of the Newton method. In the case of the Neville method, he can apply Algorithm 6 to get the values

$$b = [5185,1391,2579,1193,6165,4690,7993]$$

of partial interpolating polynomials. Consequently, he gathers the remaining shares distributed among the participants and calls the function

```
DivDif(x,y,z,b,7,13);
```

or

```
Neville(k,x,y,d,b,7,13,K);
```

in order to get the polynomial

$$\begin{aligned} w(x) = {} & 6173 + 4438(x-523) + 7317(x-523)^2 + 4191(x-523)^3 \\ & + 228(x-523)^4 + 1439(x-523)^5 + 4701(x-523)^5(x-2365) \\ & + 2639(x-523)^5(x-2365)^4 + 1688(x-523)^5(x-2365)^2 \\ & + 3461(x-523)^5(x-2365)^3 + 5336(x-523)^5(x-2365)^4(x-6543) \\ & + 205(x-523)^5(x-2365)^4(x-6543)^2 , \end{aligned}$$

or the vector

$$b = [4803,4803,4803,4748,4258,1290,3006,4507,2439,848,5257,5160,2293,6173]$$

of values of partial interpolating polynomials. Hence the recovered key is equal to

$$K = w(0) = 4803,$$

or

$$K = b[0] = b[1] = b[2] = 4803.$$

In this case, the probability of authenticity of shares is identical to

$$P_{aut} = 1 - \frac{1}{8761^2 - 1} = 0.999999987....$$

5. Secret sharing with polynomial keys

In the previous sections, we encountered the situation that the dealer *D* chose randomly coefficient of the polynomial

$$w(x) = \sum_{k=0}^{m} a_k x^{m-k}, \quad \text{m} \le \text{n}$$

distributed *c* shares

$$U_i = (k_i, x_i, y_i), \quad i = 0,1,..,c-1,$$

to the combiner and $n - c + 1$ shares

$$U_i = (k_i, x_i, y_i), \quad i = c, c+1,...,n,$$

to the remaining participants where $0 \le c \le m \le n$ and

$$\{x_0, x_1, ..., x_{c-1}\} \cap \{x_c, x_{c+1}, ..., x_n\} = \varnothing.$$

In each of these models, the role of the dealer started with the choice of the key $K = a_m$ and ended with the distribution of.

This choice of the key can be modified in such a way that it ought to reflect the structure of the problem to a greater extent and to be more complicated. More precisely, we now consider the keys of the form

$$K = a_{i_0} a_{i_1} ... a_{i_s},$$

where $0 = i_0 < i_1 < ... < i_s \le m$ and $a_{i_0}, a_{i_1}, ..., a_{i_s}$ are the coefficient of the polynomial $w(x)$, which are different from zero. The keys of this form are said to be *polynomial keys*. An example of such a, which corresponds to the polynomial

$$w(x) = 93x^{12345} + 769x^{12243} + 20x^2 + 9071,$$

has the form

$$K = 93769209071.$$

It is clear that this definition could be modified in such a way that the key includes also the coefficients equal to zero. In other words, we could set

$$K = a_0 a_1 ... a_m,$$

at least in the case, when the degree m of the polynomial $w(x)$ is not too large. Furthermore, it is also possible to take as a key an arbitrary subset of different from zero coefficients of $w(x)$ In the following, for the simplicity of our consideration, we will usually assume that polynomial keys do not include a coefficient equal to zero.

In the same way as in the previous section, the recovering of a polynomial key requires from the combiner C computing the modified divided differences

$$z_k = \langle y_0, y_1, ..., y_k \rangle, \quad k = 0,1,...,c-1,c,c+1,...,r,$$

in two steps. He should begin the first step after receiving $r-c+1$ shares of the form

$$U_{i_j} = \left(k_{i_j}, x_{i_j}, y_{i_j}\right), \quad j = 0,1,...,r-c$$

which should satisfy conditions (13) and (14). Let us recall that these conditions are characteristic of the interpolation of Hermite type. At the beginning of the second step, the combiner should simplify the notation by setting

$$U_j = \left(k_j, x_j, y_j\right), \quad j = c, c+1, ..., r.$$

Next, he should verify the authenticity of received shares by checking, whether it is true that

$$z_{m+1} = z_{m+2} ... = z_r = 0.$$

Since the unicity of interpolation implies that the identity

$$w(x) \equiv z_0 + z_1(x - x_0) + ... + z_m(x - x_0)(x - x_1)...(x - x_{m-1}),$$

holds, it follows that the combiner C can recover the polynomial key by setting

$$K = a_{i_0} a_{i_1} ... a_{i_s}, \quad i_0 < i_1 < ... < i_s,$$

where $a_{i_0}, a_{i_1}, ..., a_{i_s}$ include all nonzero coefficients defined by formulae

$$a_j = \frac{w^{(m-j)}(0)}{(m-j)!}, \quad j = 0,1,...,m.$$

Further, the key

$$K = a_0 a_1 ... a_m$$

includes all such coefficients. In order to get an algorithm of computing of coefficients a_j of the polynomial $w(x)$ represented in the Newton form, we need only to derive formulae for remainder c_0 and coefficients $c_1, c_2, ..., c_m$ of the quotient

$$q(x) = c_1 + c_2(x - x_0) + ... + c_m(x - x_0)(x - x_1)...(x - x_{m-2}),$$

which is obtained during the division

$$w(x) = c_0 + (x - \alpha) q(x)$$

of the polynomial $w(x)$ by the binomial $x-\alpha$. By setting $c_{m+1}=0$, one can rewrite the right-hand side of this formula in the form

$$w(x) \equiv c_0 + \left[(x-x_0)+(x_0-\alpha)\right] \sum_{k=1}^{m} c_k (x-x_0)(x-x_1)...(x-x_{k-2})$$

$$\equiv \sum_{k=0}^{m} \left[c_{k+1}(x_k-\alpha)+c_k\right](x-x_0)(x-x_1)...(x-x_{k-1}).$$

If we compare coefficients on both sides of this identity, we derive the following recurrent formulae

$$c_k = z_k - c_{k+1}(x_k-\alpha), \quad k=m,m-1,...,0,$$

where $c_{m+1}=0$. Now one can follow the proof of getting the extended Horner algorithm in order to establish Algorithm 7 of recovering the polynomial key *K*.

```
void   InverseHorner(float   FType   x[],float   FType
z[],float FType a[], int m)
{
   for(int j = 0; j<=m; j++)
   {
      for(int i = m-1; i>=j; i--)
          z[i] = z[i]-x[i-j] *z[i+1];
      a[m-j] = z[j];
   }
}
```

Algorithm 7. The function InverseHorner

Finally, let us note that the process of generating one of the keys

$$K = a_{i_0} a_{i_1} ... a_{i_s}$$

or

$$K = a_0 a_1 ... a_m$$

in the field $Z_b = \{0,1,...,b-1\}$ of integers modulo a prime *b*, can be started from a random choice of a large positive integer k such that the number of digits

$$a_j \in Z_b, \quad j=0,1,...,m,$$

of its representation in the system with base *b*, or equivalently the number of coefficients of the polynomial expansion

$k = \sum_{j=0}^{m} a_j b^{m-j}$,is less than $card(Z_b)-1$. Clearly, the digits of k are printed by the function `generator`, which is stated below.

```
void generator(unsigned int n, unsigned int b)
{
if(n>0){
generator(n/b, b);
     cout<<n%b;
}
}
```

Example 3. Suppose that $F = Z_{75437}$, $n = 4943$, $r = 4943$, $c = 4938$, and

$$w(x) = 205x^{4940} + 623x^{4939} + 603x + 6097 .$$

Then the polynomial key is equal to

$$K = 2056236036097 .$$

If we suppose that the values

$$y_i = w(x_i) \quad (0 \le i \le 4937)$$

of polynomial $w(x)$ at the points $x_i = i + 5$ were received by the combiner, and if the shares

$$U_{4938} = (0,5634,64704),\ U_{4939} = (1,5634,57195),\ U_{4940} = (2,5634,28201),$$
$$U_{4941} = (0,6569,42554),\ U_{4942} = (1,6569,10102),\ U_{4943} = (2,6569,25712),$$

were obtained by the participants, then in order to recover the key it is necessary to compute first the following divided differences

$$z[i] = \langle y_0,...,y_i \rangle, \quad i = 0,1,...,4943 ,$$

by using Algorithm 5. Consequently, one can apply Algorithm 7 to derive the nonzero coefficients

$$a[0] = 205, \quad a[1] = 623, \quad a[4939] = 603, \quad a[4940] = 6097$$

of expansion with respect to the polynomial basis of the modified Newton interpolating polynomial defined by the divided differences $z[i]$. The probability that the shares are authentic is now equal to

$$P_{aut} = 1 - \frac{1}{75437^3 - 1} = 0.99999999999999767... .$$

6. Orthogonal polynomial keys

In the last model of secret sharing with a polynomial key, one can replace the following two bases

$$1, x, ..., x^m$$

and

$$1, x - x_0, ..., (x - x_0)(x - x_1)...(x - x_{m-1})$$

with the arbitrary two bases

$$p_0(x), p_1(x), ..., p_m(x) \text{ and } g_0(x), g_1(x), ..., g_m(x)$$

in the space of polynomials of degree less or equal to *m*. Next, one can choose randomly the key

$$K = a_0 a_1 ... a_m$$

and set

$$w(x) = \sum_{k=0}^{m} a_k p_k(x),$$

where we may additionally assume that the coefficients equal to zero are omitted in the definition of the key *K*. For the simplicity of notation, we will ignore this assumption in the following. If two polynomial bases are fixed, then the dealer has to distribute the shares of the form

$$U_0 = b_0, \quad U_1 = b_1, ..., U_{c-1} = b_{c-1} \text{ and } U_c = b_c, \quad U_{c+1} = b_{c+1}, ..., U_m = b_m$$

between the combiner and the remaining participants, where b_0, b_1,..., b_m are equal to the coefficients of the expansion of the polynomial $w(x)$ with respect to the basis $g_0(x), g_1(x), ..., g_m(x)$, i.e.

$$w(x) = \sum_{k=0}^{m} b_k g_k(x). \tag{15}$$

Computation of these coefficients requires knowledge of an efficient algorithm to pass from the basis

$$p_0(x), p_1(x), ..., p_m(x)$$

to the basis

$$g_0(x), g_1(x), ..., g_m(x)$$

Furthermore, if the combiner receives the shares *K*

$$U_c = b_c, \quad U_{c+1} = b_{c+1}, ..., U_m = b_m,$$

then he needs an inverse algorithm to recover the key *K*.

In particular, one can obtain an interesting class of models, when we set

$$K = a_0 a_1 ... a_m,$$

$$g_0(x) = 1, \quad g_1(x) = x - x_0, ..., g_m(x) = (x - x_0)(x - x_1)...(x - x_{m-1})$$

and

$$w(x) = \sum_{k=0}^{m} a_k p_k(x), \tag{16}$$

where it is assumed that $0 \le c \le m \le r \le n$, that the knots $x_i \in F \setminus \{0\}$ $(i = 0,1,..,n)$ satisfy the same conditions as in Section 2, and that the polynomials $p_k(x)$ are defined by the following recurrent formulae

$$p_0(x) = 1, \quad p_1(x) = x - u_1$$

$$p_k(x) = (x - u_k)\, p_{k-1}(x) - v_k p_{k-2}(x), \quad k = 2,3,\ldots, m,$$

with some fixed parameters $u_k, v_k \in F$. Let us note that in the particular case, when F is the field of real numbers and if all coefficients v_k are positive, the monic polynomials

$$p_0(x), \ p_1(x), \ldots, p_m(x)$$

are orthogonal with respect to some weight function. In view of this fact, the last key K is said to be an *orthogonal key*.

Since the basis $g_0(x), g_1(x), \ldots, g_m(x)$ is the same as in the Newton formula for interpolation of Hermite type, it follows that, without loss of generality, we can restrict our attention to the case, when the combiner C owns c shares and receives $r - c + 1$ additional shares from the participants of secret sharing. Moreover, we assume that all this shares have the same form as in previous sections, i.e. that

$$U_j = (k_j, x_j, y_j), \quad j = 0,1,\ldots, c-1, c, \ldots, r.$$

In order to establish efficient algorithms to change such bases, we put

$$w_j(x) = \sum_{k=j}^{m} a_k^{(j)} p_{k-j}(x) \tag{17}$$

and

$$w_{j+1}(x) = \sum_{k=j+1}^{m} a_k^{(j+1)} p_{k-j-1}(x) \tag{18}$$

into the recurrent formula

$$w_j(x) = a_j^{(j+1)} + (x - x_i) w_{j+1}(x), \quad 0 \le j \le m,$$

for the division of the polynomial $w_j(x)$ by the binomial $x - x_i$, where $a_j^{(j+1)}$ denotes the remainder and $w_{j+1}(x)$ the quotient of this division. Consequently, one can use the recurrent formulae for the polynomials $p_{k-j}(x)$ to get

$$\begin{aligned}
\sum_{k=j}^{m} a_k^{(j)} p_{k-j}(x) &= a_j^{(j+1)} + \sum_{k=j+1}^{m} a_k^{(j+1)} \left[(x - u_{k-j}) + (u_{k-j} - x_i) \right] p_{k-j-1}(x) \\
&= a_j^{(j+1)} + \sum_{k=j+1}^{m} a_k^{(j+1)} \left[p_{k-j}(x) + (u_{k-j} - x_i) p_{k-j-1}(x) + v_{k-j} p_{k-j-2}(x) \right] \\
&= \sum_{k=j}^{m} \left[a_k^{(j+1)} + (u_{k+1-j} - x_i) a_{k+1}^{(j+1)} + v_{k+2-j} a_{k+2}^{(j+1)} \right] p_{k-j}(x),
\end{aligned}$$

where $0 \le j \le m$ and $a_{m+1}^{(j+1)} = a_{m+2}^{(j+1)} = 0$.

Now we compare the coefficients at these polynomials to get the following recurrent formulae

$$a_m^{(j+1)} = a_m^{(j)}, \quad a_{m-1}^{(j+1)} = a_{m-1}^{(j)} - (u_m - x_i)a_m^{(j+1)},$$
$$a_k^{(j+1)} = a_k^{(j)} - (u_{k+1} - x_i)a_{k+1}^{(j+1)} - v_{k+2}a_{k+2}^{(j+1)}, \quad k = m-2, m-3, \ldots, j,$$

which can be used to compute shares

$$\frac{w_0^{(j)}(x_i)}{j!} = a_j^{(j+1)}, \quad j = 0,1,\ldots,m-1,$$

for the polynomial

$$w_0(x) = \sum_{k=0}^{m} a_k^{(0)} p_k(x) := w(x).$$

It should be noticed, that these recurrent formulae generalize the well-known Clenshaw recurrent formulae [13] for computation of values $w_0(x_i)$ of the polynomial $w_0(x) = w(x)$ at the point $x = x_i$. In the function `Clenshaw`, they are applied to derive

```
void Clenshaw(FType a[], int m, FType x[], int n,
FType u[],
         FType v[],FType y[])
{
FType * b = new FType [m+1];
     int i = 0, ni;
     FType xi;
     while(i<=n)
     {
       for(int j = 0; j<=m; j++)
          b[j] = a[j];
       ni = 0; xi = x[i];
       while(i<=n && x[i]==xi)
       {
           b[m-1] = b[m-1]-(u[m-ni] -x[i])*b[m];
           for(int k=m-2;k>=ni;k--)
               b[k]=b[k]-(u[k+1-ni]-x[i])*b[k+1]-
v[k+2-ni]*b[k+2];
       }
       y[i]= b[ni];
       ni++; i++;
    }
}
```

Algorithm 8. The function Clenshaw

the following coordinates

$$y_i = \frac{w^{(k_i)}(x_i)}{k_i!}$$

of shares

$$U_i = (k_i, x_i, y_i), \quad i = 0,1,..,n$$

where

$$w(x) = \sum_{k=0}^{m} a_k p_k(x), \quad \mathrm{m} \le \mathrm{n},$$

is the polynomial, which corresponds to the orthogonal polynomial key

$$K = a_0 a_1 ... a_m .$$

As in Section 1, we assume that knots of interpolation satisfy conditions (1) and that k_i denotes the left-hand side multiplicity of the knot x_i.

Recovering of the orthogonal polynomial key K can be organized as follows. At the beginning, it is necessary to use Algorithm 2 in order to compute the modified divided differences

$$b_k = \langle y_0, \ y_1,...,y_k \rangle, \quad 0 \le k \le m .$$

Next, by applying Algorithm 9, which is based on formula (15), it remains to compute the vector *a* starting from the following values

$$a_0 = b_0, \quad a_1 = b_1,...,a_m = b_m .$$

```
void InverseClenshaw(FType a[],int n, FType x[],int m,
                     Ftype u[],Ftype v[])
{
for(int j = m-1; j>=0; j--)
{
     for(int k = j; k <= m - 2; k++)
            a[k]=a[k]+(u[k+1-j]-x[j])*a[k+1]+v[k+2-
j]*a[k+2];
          a[m-1] = a[m-1]+(u[m-j] -x[j])*a[m];
     }
}
```

Algorithm 9. The function InverseClenshaw

Example 4. Suppose that $F = Z_{2341}$, $n = 11$, $m = 10$ and these polynomials

$$p_0(x) = 1, \quad p_1(x) = x,$$
$$p_i(x) = x p_{i-1}(x) - 4 p_{i-2}(x), \quad i = 2,3,...,10,$$

are the monic Chebyshev polynomials of the first kind. If the dealer selects the polynomial

$$w(x)=123p_0(x)+205p_1(x)+10p_4(x)+132p_7(x)+456p_{10}(x)$$

then the orthogonal key can be equal to

$$K=12320500100013200456,$$

or equivalently to

$$K=12320510132456.$$

Further, suppose that the dealer used Algorithm 8 to compute the shares

$$U_0=(0,17,234),\ U_1=(1,18,2224),\ U_2=(2,18,199),\ U_3=(0,23,1252),$$
$$U_4=(1,23,1039),\ U_5=(2,23,389),\ U_6=(0,46,1472),\ U_7=(1,46,1103),$$
$$U_8=(2,46,865),\ U_9=(0,111,181),\ U_{10}=(1,111,1295),\ U_{11}=(0,144,1668),$$

which were given to participants of secret sharing, and perhaps to the combiner. Then the recovering of the key K requires of using Algorithm 9 to at least 10 shares, e.g. to shares $U_0,U_1,...,U_9$. This algorithm yields the following vector

$$a=[123,205,0,0,10,0,0,132,0,0,456],$$

with nonzero coordinates identical to the orthogonal key K.

Finally, let us note that the execution of Algorithm 2 to compute starting values of vector a for all eleven shares, and the calling of the function

```
InverseClenshaw(a,11,x,11,u,v);
```

yield vector a of the form

$$a=[123,205,0,0,10,0,0,132,0,0,456,0].$$

The last coordinate of this vector is equal to zero, which proves the authenticity of received shares and recovered key with the probability

$$P_{aut}=1-\frac{1}{2340}=0.9996...$$

7. Fast Fourier transform in secret sharing

In this section, we consider the case when the dealer fixed the polynomial

$$w(x)=\sum_{j=0}^{n-1}a_jx^j$$

and chose the key

$$K=a_0a_1...a_{n-1}$$

or

$$K=w(0)=a_0$$

in a commutable ring $F=(F,+,\cdot)$, which has a primitive ω of degree $n=2^k$ $(k\geq 1)$ from the unity. Let us recall that ω should satisfy [1] conditions: $\omega\neq 1$, $\omega^n=1$ and

$$\sum_{j=0}^{n-1}\omega^{pj}=0,\quad 1\leq p<n.$$

Moreover, it will be assumed that the combiner recovers the key K from shares of the form $U_i = (i, y_i), \quad i = 0,1,..,n-1,$

where y_i denote the values of polynomial $w(x)$ at the roots ω^i of degree n rom the unity, i.e.

$$y_i = \sum_{j=0}^{n-1} a_j \omega^{ij}, \quad 0 \le i < n. \tag{19}$$

The last equations define the discrete Fourier transform of the cartesian product F^n into itself. If we additionally assume that the element n has an inverse in the ring F [15], then the unique solution of the system (19) is given by the formulae

$$a_i = \frac{1}{n}\sum_{j=0}^{n-1} y_j \omega^{-ij} = \frac{1}{n}\sum_{j=0}^{n-1} y_j \omega^{(n-i)j}, \quad 0 \le i < n, \tag{20}$$

which define the inverse discrete Fourier transform. It is clear that to the computation of the Fourier and inverse Fourier transforms one can apply n times Horner algorithm for the points

$$1, \ \omega, \ \omega^2, ... \ \omega^{n-1},$$

which requires $O(n^2)$ algebraic operations. Let us note, that the order of the number of algebraic operations for the algorithms of Newton and Neville is the same.

One can essentially decrease this order to $O(n \log_2 n)$, whenever we apply the FFT-algorithm (the fast Fourier transform algorithm) for computations of both the Fourier and inverse Fourier transforms. In the proceeding, we present the simplest derivation of this algorithm. More precisely, its recursive version will be given, which is modeled on the famous algorithm of quick sorting presented in the monograph [16]. For this purpose, we need to compute the remainders

$$r_{l,l+m-1}(x) = \sum_{j=0}^{m-1} c_j x^j \quad \text{and} \quad r_{l+m,p}(x) = \sum_{j=0}^{m-1} d_j x^j$$

obtained from the division of the previous remainder

$$r_{l,p}(x) = \sum_{j=0}^{2m-1} b_j x^j, \quad m = \frac{p-l+1}{2}$$

by polynomials

$$g_{l,l+m-1}(x) = \prod_{j=l}^{l+m-1} \left(x - \omega^{r[j]}\right) = x^m - \omega^{r[l/m]},$$

and

$$g_{l+m,p}(x) = \prod_{j=l+m}^{p} \left(x - \omega^{r[j]}\right) = x^m + \omega^{r[l/m]},$$

where $0 \le l \le p \le n-1$, coefficients b_j of the starting remainder $r_{0,n-1}(x)$ are equal to the coefficients a_j of the Fourier transform to be computed, the integers l, m and l/m are either nonnegative powers of 2 or are equal to zero, and the coordinates of sequence $r[0], r[1], .., r[n-1]$ have k-digit binary representations of the form

$$r[j] = \langle j_{k-1}, j_{k-2}, ,..., j_0 \rangle_2 ,$$

which are derived from the reflection of the following k-digit binary representations

$$j = \langle j_0, j_1, ,..., j_{k1} \rangle_2 ,$$

In the following inductive proofs of the simplified formulae for divisors,

$$\begin{aligned} g_{l,p}(x) &= g_{l,l+m-1}(x) g_{l+m,p}(x) \\ &= x^{p-l+1} - \left(\omega^{r[l/m]} + \omega^{r[1+l/m]}\right) x^m + \omega^{r[l/m]} \omega^{r[1+l/m]} \\ &= x^{p-l+1} - \omega^{r[l/(p-l+1)]}, \end{aligned}$$

we apply the identities

$$\omega^{r[1+l/m]} = \omega^{n/2} \omega^{r[l/m]} = -\omega^{r[l/m]} = -\omega^{\frac{1}{2} r[l/(p-l+1))} ,$$

and formulae for the coefficients c_j we derive similarly as in the previous section:

$$\begin{aligned} \sum_{j=0}^{2m-1} b_j x^j &= \left(x^{m-1} - \omega^{r[l/m]}\right) \sum_{j=0}^{m-1} t_j x^j + \sum_{j=0}^{m-1} c_j x^j \\ &= \sum_{j=0}^{m-1} \left(c_j - t_j \omega^{r[l/m]}\right) x^j + \sum_{j=0}^{m-1} t_j x^{j+m-1}, \end{aligned}$$

whence

$$c_j = b_j + t_j \omega^{r[l/m]}, \quad j = 0,1,..,m-1 .$$

Since the formulae for coefficients d_j are similar, it follows eventually that the recursive function of fast computing the Fourier transform for the data

$$a[j] = a_j \quad \text{and} \quad x[j] = \omega^{r[j]}, \quad 0 \le j \le n-1 ,$$

can be stated as in Algorithm 10.

This algorithm yields permuted coordinates a

$$a[j] = w(x[j]), \quad j = 0,1,...,n-1 .$$

of vector a. In other words, the coordinates of shares $U_i = (i, y_i)$, computed by the dealer, are equal to

$$y_j = a[r[j]], \quad j = 0,1,...,n-1 .$$

```
void Fourie(FType a[],int l, int p, FType x[])
{
```

```
    int m;
    FType z, w;
    m = (p-l+1)/2;
    z = x[l/m];
    for (int j=l; j<l+m; j++)
    {
        w = a[j]+z*a[j+m];
        a[j+m] = a[j]-z*a[jam];
        a[j] = w;
    }
    if (m>l)
        Fourier(a,l,m+l-1,x);
        Fourier(a,m+l,p,x);
}
```

Algorithm 10. The function Fourier

Note that the call of this function of the form

```
Fourier(a,0,n-1,x);
```

will force its next $k = \log_2 n$ consequent calls. Hence the cost of the fast Fourier transform is equal to

$$2n\log_2 n = O\left(n\log_2 n\right)$$

of algebraic operations from the ring F. Clearly, the reduction of algebraic operations in the FFT algorithm increases considerably with the increase of n.

Since recovering of the key K can be based on the inverse Fourier transform, it follows from the second formula given in (20) that the following function call

```
Fourier(a,0,n-1,x);
```

for the vectors

$$a[0] = \frac{1}{n}y_0, \quad a[1] = \frac{1}{n}y_1, ..., \quad a[n-1] = \frac{1}{n}y_{n-1}$$

and

$$x[j] = \omega^{r[n-j]}, \quad 0 \le j \le n-1,$$

transforms the vector a in such a way that the final values

$$a[j] = w\left(x[n-j]\right), \quad 0 \le j \le n-1$$

recover the key

$$K = a\left[r[n]\right]a\left[r[n-1]\right]...a\left[r[1]\right].$$

Example 5. Let $n = 2^3 = 8$, $F = Z_{17}$, $\omega = 2$, and

$$w(x) = 1 + 3x^2 + x^6 + 2x^7 .$$

Then $n^{-1} = 15$ and the key K and values x_i $(0 \le i < 7)$ are identical to

$$K = 10300012$$

and

$$x_0 = \omega^0 = 1, \quad x_1 = \omega^4 = 16, \quad x_2 = \omega^2 = 4, \quad x_3 = \omega^6 = 13,$$
$$x_4 = \omega^1 = 2, \quad x_5 = \omega^5 = 15, \quad x_6 = \omega^3 = 8, \quad x_7 = \omega^7 = 9.$$

By using the function Fourier we get

$$w(1) = a_0 = 7, \quad w(16) = a_1 = 3, \quad w(4) = a_2 = 6, \quad w(13) = a_3 = 5,$$
$$w(2) = a_4 = 10, \quad w(15) = a_5 = 8, \quad w(8) = a_6 = 6, \quad w(9) = a_7 = 14,$$

whence

$$U_0 = (0,7), \quad U_1 = (1,10), \quad U_2 = (2,6), \quad U_3 = (3,6),$$
$$U_4 = (4,3), \quad U_5 = (5,8), \quad U_6 = (6,5), \quad U_7 = (7,14).$$

If the combiner C receives these shares from the participants, he uses formulae (20) to computes

$$a[0] = 3,\ a[1] = 14,\ a[2] = 5,\ a[3] = 5,\ a[4] = 11,\ a[5] = 1,\ a[6] = 7,\ a[7] = 6,$$

and performs a function call of the form

```
Fourier(a,0,7,x);
```

in order to get the new values of coordinates of vector a. Finally, he puts

$$K = a[0]a[7]a[3]a[5]a[1]a[6]a[2]a[4] = 10300012 .$$

References

[1] Shamir A., *How to share a secret*, Communications of the ACM, 22 (1979) 612.
[2] Blakley G., *Safeguarding cryptographic keys*, Proceedings of AFIPS 1979 National Computer Conference 48 (1979) 313.
[3] Chang T.Y., Hwang M.S., Yang W.P., *An improvement on the Lin–Wu (t,n) threshold verifiable multi-secret sharing scheme*, Applied Mathematics and Computation, 163 (2005) 169.
[4] Chang C.C., Lin C.H., Lee W., Hwang P.C., Secret Sharing with Access Structures in a Hierarchy, International Conference on Advanced Information Networking and Applications 2 (AINA'04), Fukuoka, (2004).
[5] Feng J.B., Wu H.C., Tsai C.S., Chu Y.P., *A new multi-secret images sharing scheme using Largrange's interpolation*, The Journal of Systems and Software, 76 (2005) 327.
[6] Yang C., Chang T., Hwang M., *A (t,n) multi-secret sharing scheme*, Applied Mathematics and Computation, 151 (2004) 483.
[7] Menezes A.J., van Oorschot P.C., Vanstone S.A., *Handbook of Applied Cryptography*, CRC Press, 1996.
[8] Pieprzyk J., Hardjono T., Seberry J., *Fundamentals of Computer Security*, Springer-Verlag Berlin Heidelberg, (2003).
[9] Stinson D.R., *Cryptography Theory and Practice*, CRC Press, (1995).
[10] Dale N., Weems C., *Headington M., Programming and Problem Solving with C++*, Jones and Bartlett Publ., Boston, (2002).
[11] Mühlbach G., *Computation of Cauchy-Vandermonde determinants*, J. Number Theory, 43 (1993) 74.
[12] Shoup V., http://www.shoup.net/.

[13] Stoer J., Bulirsch R., *Introduction to Numerical Analysis*, Springer-Verlag, New York, (1993).
[14] Pieprzyk J., Zhang X.M., *Cheating Prevention in Linear Secret Sharing*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2384 (2002) 121.
[15] Aho A.V., Hopcroft J.E., Ullman J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, London, (1974).
[16] Wirth N., *Algorithms + Data Structures = Programs*, Prentice-Hall (1975).