The Multiplication of Very Large Integers Using the Discrete Fast Fourier Transform

From Art to Necessity in Cryptosystems

by **Dr. Daniel GUINIER**

ACM, IEEE, IACR, EDPAA, IIA member

President OSIA, Inc.

and

Computer Security and Quality Department Chairman
Regional Institute for Promotion of Applied Research (IREPA)

Abstract

Year after year, there is an incredible increase in computation power delivered by parallelism and new kinds of processors. The length of integers involved in the computations in public-key cryptosystems will probably be more than **one thousand bits within the next ten years** to warranty a good security level.

The proposal is to present a possible software solution for 1024, 1279, 2048 and 4096-bit numbers which combine **software** and **general purpose hardware**. The algorithm implemented makes use of the Discrete fast Fourier Transform *(DFT)* and produces first results which can be compared to others like Karatsuba's or modular arithmetic.

Considering these results for cryptographic applications using asymmetric systems for key exchanges one can take advantage of such a method for modular exponentiation but only for operations involving more than 1024 bits. While using a quad-transputer board equipped with the new T9000 family transputers, software modular exponentiation involving 1024-bit numbers should take under 0.5 s, 2048 bits under 2 s and 4096 bits less than 8 s which takes more time but insures the future of the *just necessary* security ! Expected results for this method show **possible software applications for highly secure key exchange protocols using general purpose new chips** *(INMOS T9000 transputer or Intel i860 when used in piped mode).*

Keywords : *Arithmetic, Cryptography, DFT, Exponentiation, Key exchange, Large integer, Multiplication, Protocol, Transputer*

Mailing address : BP 86
F 67034 STRASBOURG Cedex, (France)

1. Introduction

Multiplication of large integers is at the **heart of cryptographic public-key systems** like those described by Diffie-Hellman (1976), Rivest-Shamir-Adleman (1978) or El Gamal schemes based on exponentiation. Because computations are done in large finite field, a reduction is then applied modulo N, a 512-bit number in most cases or a 521, 607-bit Mersenne prime (D. Guinier (1988)).

Such a technique for the multiplication of large numbers is quite important as a tool for modern **Cryptography**. Because there is a large increase of computing power delivered by parallelism and new kinds of processors each year, length of such integers will be **probably more than 512 bits in the next ten years** to furnish the same security level as now. This is why efficient software techniques are rewarded for their generation.

Also, we need to be able to do such **computations for very long integers** efficiently and to find an easy method of software development as well as a flexible hardware support at reasonable cost. Several methods using Data Signal Processor chips *(DSP)* and transputers *(TRAM)* for flexible and efficient software implementations have been described by D. Guinier (1989) as well as algorithms for the multiplication of large integers by the use of modular arithmetic (D.Guinier (1990)). Our proposal is to find possible **solutions for 1024, 1279, 2048-bit numbers** *(and more)* which combine software and general purpose hardware.

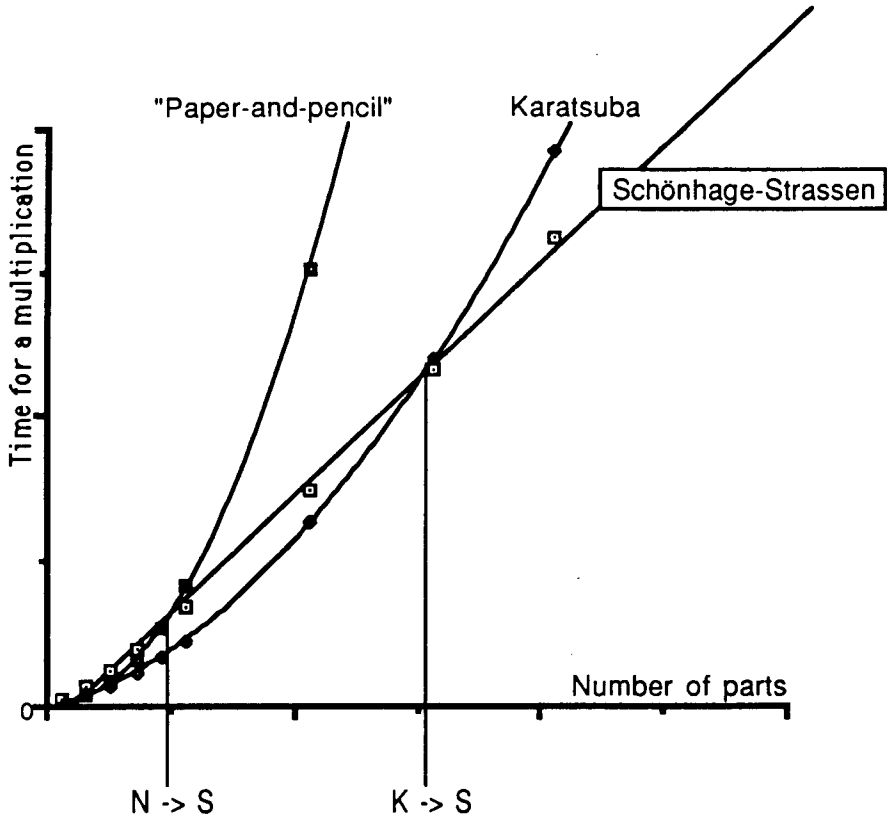
2. Computations and complexity

Computations over large finite fields **involve an arithmetic process** where large integers are split into several parts of length up to the machine word that is 16 or 32 bits for unsigned integers. Compared to the traditional (naïve) *"paper and pencil"* method, the Karatsuba algorithm (Karatsuba A., Ofman Y. (1962)) offers a very time saving way to do the multiplication of two 2.nb-bit numbers.

Karatsuba's method	*(3 multiplications)*
C=A.B= a0 **x** b0*(Mask+1) + (a1 - a0) **x** (b0 + b1)* Mask2 + a1 **x** b1*(Mask+Mask2)	
"Paper and pencil" method	*(4 multiplications)*
C=A.B= a0 **x** b0 + ((a1 **x** b0) + (a0 **x** b1))* Mask2 + a1 **x** b1*(Mask)	

In this case, the two-part long multiplication **A.B = a1a0.b1b0** can be performed in three multiplications, some adds, some shifts and an eventual carry compared to four multiplications in the *"paper and pencil"* method and can handle double length integers with the same number of multiplications. This technique will be used to perform intermediate **quad-multiplications** when necessary.

The principle proposed uses **a variation of the Schönhage-Strassen method** (A. Schönhage and V. Strassen (1971)) which is based on the Discrete fast Fourier Transform (DFT) which simplifies the multiplication of two polynomials. In fact, after transformation, the multiplication can be done term by term which is O(n.log(n).log(log(n))) and dramatically increase the velocity in comparison to the two other methods *(naïve paper and pencil method* $O(n^2)$ *and Karatsuba's method* $O(n^{\log_2 3})$). **The art is *just* to find a good way to transform and inverse transform in a minimal time !**



Comparative performances of the three methods are shown on the graph. These curves are confirmed by F. Morain and P.Zimmermann *(INRIA, personal communication).* **The benefit of such an algorithm will be found only for very large numbers.**

3. Discrete Fourier Transform and Convolution Theorem

An implementation of the algorithm for the DFT was presented by in a finite field by J.M. Pollard (1971) and the algorithm for the multiplication by A. Schönhage and V. Strassen (1971) and D. E. Knuth (1981) using a DFT involving only integer computations. A excellent presentation and a discussion on the implementation can be found in A.V. Aho, J.E. Hopcroft and J.D. Ullman (1975) and G. Brassard, S. Monet, D. Zuffelato (1986). Some variations can be found in A.Borodin, I. Munro (1975), and D.H. Bailey (1988).

There is an n^{th} root of unit ω_n and the Fourier transform of an element of vector A = $\{a_{n-1}, a_{n-2}, ..., a_2, a_1, a_0\}$ from finite field G is :

$$\alpha_i = \sum_{k=0, n-1} a_k \omega_n^{ik} \qquad \text{for } i=0 \text{ to } n-1$$

while the inverse transform is :

$$\alpha^{-1}{}_i = 1/n \sum_{k=0, n-1} a_k \omega_n^{-ik} \qquad \text{for } i=0 \text{ to } n-1$$

If A and B are elements of field G and their convolution product C = A O B then the **Convolution Theorem** is :

$$C = F^{-1}(F(A) . F(B))$$

with $F(A) = \{\alpha_{n-1}, \alpha_{n-2}, ..., \alpha_2, \alpha_1, \alpha_0\}$
and $F(B) = \{\beta_{n-1}, \beta_{n-2}, ..., \beta_2, \beta_1, \beta_0\}$

that is just the inverse transform of the **term-to-term product** of the transforms of the two vectors A and B.

The Schönhage-Strassen algorithm works only with a base which is a power of two. The present algorithm could be qualified as a variation in O (n(log2n)).

4. Choice of an appropriate field

We just **need a ring** which has an n^{th} **root of unit** w_n with an **inverse**. Also, for velocity it is judicious to have w_n as **a power of 2**. Possible rings are in the form with **n/(P-1)** is **an integer** and **2** is an n^{th} **root of unit**. Primes in the form $P = k.2^q + 1$ can work and **Fermat numbers** $Ft=2^{2^t}+1$ can help to find easily the Fourier transforms of length of 2^{t+1} and root = 2. **Mersenne numbers** $Mp=2^p - 1$ could also work because the root of the number theoric transform is also 2.

Inverse of 2 modulo a Fermat number Ft = $2^{2^{t+1}} - 1$

A are **n-part** integers $\{a_{n-1}, a_{n-2}, ..., a_2, a_1, a_0\}$ each < base b
B are **n-part** integers $\{b_{n-1}, b_{n-2}, ..., b_2, b_1, b_0\}$ each < base b
$n=2^m$
$n.(b-1)^2 <$ P that is **$2^m.(b-1)^2 < k.2^q + 1$**

Efficiency of the method increases in proportion of the ratio log (b) /log (P) but also requires more memory location.

5. Algorithms for fast Discrete Fourier Transforms (DFT)

The size n of the DFT will be chosen as a power of 2 to facilitate the operations (J.M.Cooley, J.W.Tukey (1965)). To avoid the rearrangement of the coefficients, two different algorithms will be applied : the **decimation in frequency for the Fourier transforms** F(A) and F(B) to get into the scrambled Fourier domain, and the **decimation in time for the inverse Fourier transform** to realize the convolution product : $C = F^{-1}$ (F(A) . F(B)) and to get back in the time domain. This double transformation trick avoids rearrangement of the data.

5.1. Decimation in frequency

$$\alpha_i = \sum_{k=0,\, n-1} a_k \omega_n^{ik} = \sum_{k=0,\, n/2-1} \omega_n^{ik} . [a_k + a_{k+n/2} . \omega_n^{in/2}] \quad \text{for } i=0 \text{ to } n-1$$

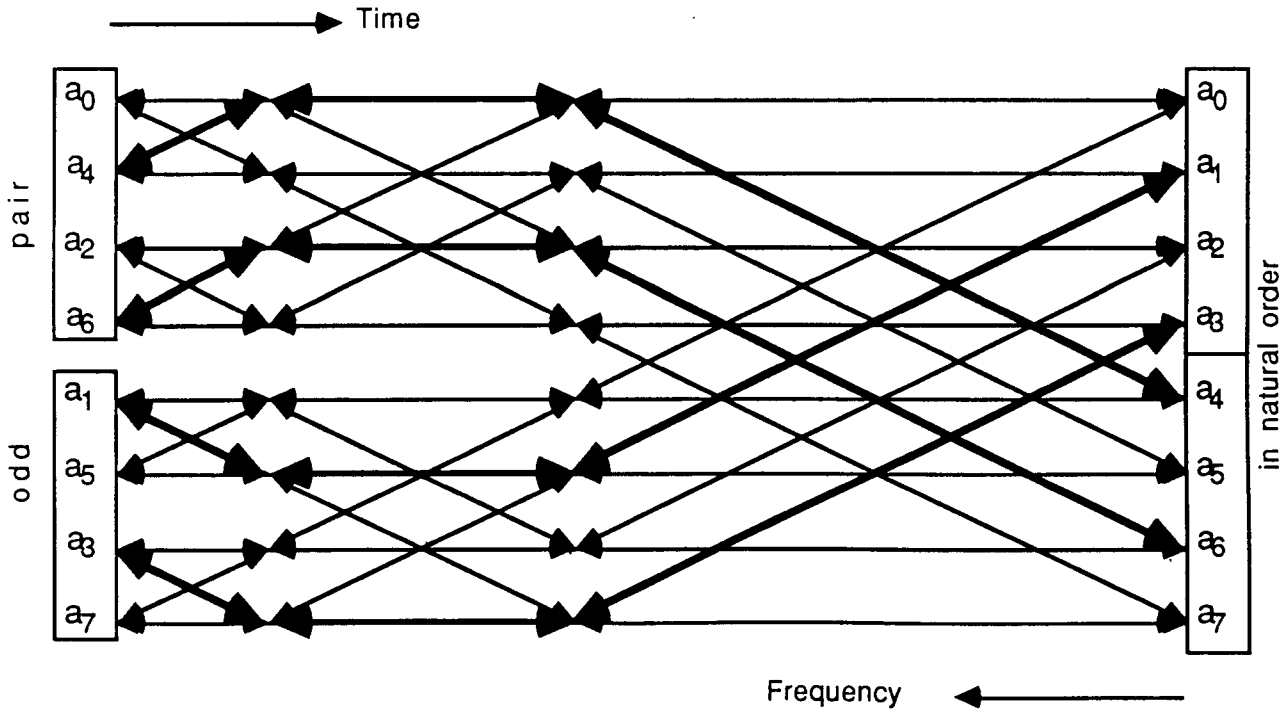
Computations can be done separating **odds and pairs**, each of length n/2 :

$$\alpha_{2i} = \sum_{k=0,\, n/2-1} \omega_n^{2ik} . [a_k + a_{k+n/2}]$$

$$\alpha_{2i+1} = \sum_{k=0,\, n/2-1} \omega_n^{i} . \omega_n^{2ik} . [a_k - a_{k+n/2}]$$

for i=0 to n/2 - 1 (n/2 is also an integer because n is a power of two)

In the case of the decimation in frequency, input data are first iterated in natural order to give results according mirror transformation. They should be rearranged but, because decimation in frequency and decimation in time use a reciprocal process, it is not necessary to rearrange any data if we use one such decimation after the other.



5.2. Algorithm DFT-decimation in frequency

Val	Table of **values** corresponding to the coordinates in the **base b**
n	Number of parts
ntm1	$= n^t - 1$
Nsize	The binary log of the number n of parts : $Nsize=\log_2(n)$
ω	The binary log of the n^{th} root of unit
rsh (X,Npos)	Right shift (divide X by 2^{Npos} (Npos is the number of positions))
lsh (X,Npos)	Left shift (multiply X by 2^{Npos} (Npos is the number of positions))

DFT_freq

```
Loop = 1
M = rsh (n,1)
L = Nsize
Do while {L > 0}
    Ω = 0
    For j = 0 to M-1
        For i=0 to Loop-1
            I1 = lsh (i,L) + j
            I2 = I1+M
            Wrk = Val(I1) + Val(I2)
            Val(I2) = Val(I1) - Val(I2)
            Val(I2)=lsh(Val(I2),Ω)
            Val(I1) = Wrk
        End for
        Ω = ( Ω + ω.Loop) and (ntm1)
    End for
    Loop = lsh (Loop,1)
    L = L - 1
    M = rsh (M,1)
End do while
```

5.3. Decimation in time

$$\alpha_i = \sum_{k=0,\, n-1} a_k \, \omega_n^{\,ik} \qquad \text{for } i=0 \text{ to } n-1$$

Computations can be done **in two halves (lower and upper)** of length n/2 :

$$\alpha_i = \sum_{k=0,\, n/2-1} a_{2k} \, \omega_{n/2}^{\,ik} + \omega_n^{\,i} \sum_{k=0,\, n/2-1} a_{2k+1} \, \omega_{n/2}^{\,ik}$$

$$\alpha_{i+n/2} = \sum_{k=0,\, n/2-1} a_{2k} \, \omega_{n/2}^{\,ik} - \omega_n^{\,i} \sum_{k=0,\, n/2-1} a_{2k+1} \, \omega_{n/2}^{\,ik}$$

for i=0 to n/2 - 1 (n/2 is also an integer because n is a power of two)

In the case of decimation in time, input data need to be first arranged according to mirror transformation to give an output in natural order. But because decimation in frequency and decimation in time use a reciprocal process, it is not necessary to rearrange any data if we use one such decimation after the other.

5.4. Algorithm DFT-decimation in time

DFT_freq

```
Loop = rsh (n,1)
M = 1
L = 1
Do while { L <= Nsize }
    Ω = 0
    For j = 0 to M-1
        For i=0 to Loop-1
            I1 = lsh (i,L) + j
            I2 = I1+M
            Wrk = Val(I1) + lsh(Val(I2), Ω)
            Val(I2) = Val(I1) - Wrk
            Val(I1) = Wrk
        End
        Ω = ( Ω + ω.Loop) and (ntm1)
    End
    Loop = rsh (Loop,1)
    L = L + 1
    M = lsh (M,1)
End do while
```

5.5. Algorithm DFT-mutiplication

DFT_longmul (A, B, C)

```
Call DFT_Freq (A ->FA)                          (Get terms of transform)
Call DFT_Freq (B -> FB)
For i=0 to N-1
    Call Modular_PROD (AB(i),FA(i),FB(i))       (Term to term products)
End for
Call DFT_Time (AB -> FAB)                       (Inverse transform of products)
Call Modular_DIVI (C(0),FAB(0),N)
For i=1 to N-1
    Call Modular_DIVI (C(N-i),FAB(i),N)         (Shift of log2 [(P-1)/N])
End for
```

6. Implementation of the very long multiplication

P will be the Fermat number $P=Ft=2^{2^t}+1$ with t=6 and log2(P-1)=64 and $N=2^7=128$ which will be the actual maximum size for the DFT corresponding also to Nsize=7. That means we are working in the ring $G/(2^{64}+1)G$. A and B are composed of 64-part integers :

{an-1, an-2, ..., a2, a1, a0 } and { ban-1, bn-2, ..., b2, b1, b0 } to form a product of 128 parts.

Using the inequation $n.(b-1)^2 < P$, we can choose a value for the **base b** :

nbr. of parts for C=A.B	**k** for $P=2^k+1$	**nb** for **$b=2^{nb}-1$**	max length for A or B	ratio log (b) /log (P)
256	16	4	512 bits	0.25
128	32	12	768	0.37
64	**64**	**29**	**928**	**0.45**
128	64	28	1792	0.43
256	64	27	3456	0.42
512	64	27	6912	0.42

With base $b < 2^{29}$ *(maximum value for base b=536870911)* it is possible to work with a product $< (2^{29})^{64}-1$ that is $2^{1856}-1$ corresponding to 928-bit numbers for A and B. This gives about 279 decimals which is sufficient for cryptographic computations and gives the best efficient ratio *(0.45)*. With such a choice, it is required to use the quad-multiplication for the products of 64-bit integers implemented with Karatsuba's method to increase velocity*(see D.Guinier (1988))*.

7. Reduction modulo Ft of a product

If A or B is equal to zero : result will be 0 ! Otherwise **A and B are non zero integers** (which is the case in Cryptosystems) : let the binary representation of a product C=A.B be a candidate for the reduction modulo Ft with $m=2^t$ to be :

$C=A.B=\{ c_{2m-1}, c_{2m-2}, ..., c_{m+2}, c_{m+1}, c_m , c_{m-1}, c_{m-2}, ..., c_2, c_1, c_0 \}$ bits
The m-least significant bits (lsb) $\{ c_{m-1}, c_{m-2}, ..., c_2, c_1, c_0 \}$ bits
The m-most significant bits (msb) $\{ c_{2m-1}, c_{m-2}, ..., c_{m+2}, c_{m+1}, c_m \}$ bits

The method consists in the subtraction of the m most significant bits from the m least significant bits, which is performed by the add of the *1's complement plus one*. The algorithm is :

Split C in the {m msb}-{m lsb}	
Take the m lsb bits	$\{ c_{m-1}, c_{m-2}, ..., c_2, c_1, c_0 \}$ bits
Add 1's complement of the msb	$\{ \overline{c_{2m-1}}, \overline{c_{2m-2}}, ..., \overline{c_{m+2}}, \overline{c_{m+1}}, \overline{c_m} \}$ bits
Add 1	1
Add carry 1's complement	1/0
Result	C = A.B modulo Ft

This method is efficient and takes advantage of large Fermat numbers, where it is required to use multiprecision arithmetic (eg. $2^{64}+1$). It needs only one split, two long adds (one to find the complement and one sum) and two single word adds.

8. Performance of such a very long multiplication

The actual DFT implementation of the **928-bit multiplication** takes 5.5 less time than the corresponding *naïve schoolboy paper and pencil* long multiplication but is not optimized. Y.Herreros (1990) has shown that it is possible to use a base b different from a power of two *(e.g. decimal base)* and the **multiplication of two 65536 decimal numbers** *(using 32768-parts in base b=100 and* $P= 3.2^{31} + 1$*)* takes **15 s** on a **Sun3** workstation. The application is easily implementable under a Transputer machine and gives efficient results (**0.5 s** is needed when it is performed with 32 transputers for the same calculus).

9. Comparison with other software methods

Performance obtained with other software implemented methods for the multiplication and the exponentiation must be considered. The following data are from the software RSA toolkit produced by the european firm CRYPTECH and programmed in Microsoft C 5.10.When a 512-bit RSA is considered to be used in the *middle case* (the 512-bit exponent is half with '1'), the computation time is :

8.3 s. on a 80286 machine at 8 MHz (IBM AT)
5.3 s. on a 80286 machine at 12 MHz
3.0 s. on a 80386 machine at 20 MHz

Different n-bit modulus-exponent implementations of the RSA on a 80286 machine at 12 MHz could be estimated in time performance as :

n-bit modulus	k=n-bit/256	Time in s. = $k.(0.25+ 0.6.k^2)$
512-bit	k= 2	**5.3 s**
1024-bit	k= 4	**39 s**
2048-bit	k= 8	309 s **5 mn**
4096-bit	k=16	2461 s **41 mn**

Remark : A 512-bit modulus and exponent RSA takes respectively from 4.0 s to under 6.0 s if the exponent comes with one '1' or 100% of '1'; 5.3 s has been given for the *middle case (50% of '1')* and the time required only for the exponentiation without reduction should be at least reduced by a factor two.

10.Conclusion

Considering these results for **Cryptographic applications** using asymmetric systems **for key exchanges, one can take advantage of such a DFT method** for modular exponentiation but only for operations involving more than 1024 bits. Under this condition, Karatsuba's method is quite preferable. With a quad-transputer board using the new **T9000 Transputer** *(200 Mips)* from INMOS, software exponentiation involving **1024-bit** numbers should take under **0.5 s** and **2048-bit** numbers, under **2 s** and **4096-bit** under **8 s** to enforce security in case of necessity ! Another possibility should be to use the **i860 pipes** because alternate additions and multiplications is a typical problem the i860 can solve using at the same time the adder and the multiplier *(200 Mips at 40 MHz);* the Cray X would crank just 6 times more but not at the same price and not on a portable machine ! Also, these different expected results show **possible software applications of the method for highly secure key exchange protocols using just general purpose new chips**.

10. Bibliography

D.H. Bailey (1988) : The computation of π to 29 3600 000 decimal digits using Borwein' quartically convergent algorithm. Math Comput, Vo.50, No.181, pp.283-296.

Borodin A., Munro I. (1975) : The computational complexity of algebraic and numeric problems. Amer Elsevier, New York.

Brassard G., Monet S., Zuffelato D. (1986) : Algorithme pour l'arithmétique des grands entiers. Technique des Sciences- Informatique. AFCET-Gauthier-Villars Vo.5, no.2, pp.89-102.

Cooley J.M., Tukey J.W. (1965) : An algorithm for the machine calculation of complex Fourier series. Math Comput,Vol.19, pp.297-301.

Diffie W., Hellman M.E. (1976) : New directions in cryptography, IEEE Trans on Information Theory, vol IT-22, Nov, pp.644-654.

Guinier D. (1988) : D.S.P.P. : A Data Security Pipe Protocol for PC's, Large Scale Systems or Networks. ACM - SIGSAC *REVIEW* , (Special Interest Group on Security Audit and Control, Vol.6, No.4, pp.4-9.

Guinier D. (1989) : Parallel exponentiators using Data Signal Processor chips and Transputers for a flexible and efficient software implementation of Public-key cryptosystems to run on PC's or larger systems.ACM - SIGSAC *REVIEW* , Special Interest Group on Security Audit and Control, Vol.7, No.4,

pp.20-33.

Guinier D. (1990) : Multiplication of large integers by the use of modular arithmetic, Application to Cryptography. ACM - SIGSAC *REVIEW* , Special Interest Group on Security Audit and Control, Vol.7, No.4, pp.8-20.

Herreros Y. (1990) : Un algorithme pour la multiplication rapide de grands entiers à l'aide de transformés de Fourier dans des corps finis (in press).

Karatsuba A., Ofman Y. (1962) : Multiplication on multidigit numbers on automata. Dolk Akad Nauk SSSR. Vol.145, pp.293-294.

Knuth D.E. (1981) : The art of computer programming : Semi-numerical algorithms. 2nd. édit., Addison-Wesley Reading, Vol.2.

Pollard J.M. (1971) : The Fast Fourier Transform in a finite field. Math Comput. Vol.25, pp.365-371.

Rivest R.L., Shamir A., Adleman L. (1978) : On digital signatures and public-key cryptosystems. Comm ACM, Vol.21, No.2, pp.120-126.

Schönhage A., Strassen V. (1971) : Schnelle multiplikation grosser zalhen. Computing. Vol.7, pp.281-292.