SPEEDING UP THE COMPUTATIONS ON AN ELLIPTIC CURVE USING ADDITION-SUBTRACTION CHAINS

François Morain [*†]
Jorge Olivos [‡]

September 12, 1990

Abstract

We show how to compute x^k using multiplications and divisions. We use this method in the context of elliptic curves for which a law exists with the property that division has the same cost as multiplication. Our best algorithm is 11.11% faster than the ordinary binary algorithm and speeds up accordingly the factorization and primality testing algorithms using elliptic curves.

1. Introduction. Recent algorithms used in primality testing and integer factorization make use of elliptic curves defined over finite fields or Artinian rings (cf. Section 2). One can define over these sets an abelian law. As a consequence, one can transpose over the corresponding groups all the classical algorithms that were designed over $\mathbf{Z}/N\mathbf{Z}$. In particular, one has the analogue of the $p-1$ factorization algorithm of Pollard [29, 5, 20, 22], the Fermat-like primality testing algorithms [1, 14, 21, 26] and the public key cryptosystems based on RSA [30, 17, 19]. The basic operation performed on an elliptic curve is the computation of the analogue of x^k mod N in the case where we work over $\mathbf{Z}/N\mathbf{Z}$. Whatever the model of computation may be, the number of elementary operations (understood as being the product of two large integers modulo an integer N) is very high. Thus, reducing the number of such operations is a primary goal when implementing these algorithms.

One can look first for expressions of the law that minimizes the number of operations [9]. Another idea is to reduce the number of multiplications needed to reach x^k. One way of handling this problem is to introduce the concept of *addition chain* [18] for exponents. An addition-chain for the exponent k is given as an $(r+1)$-tuple $(k_0, \ldots, k_r)$ of positive integers such that

$$k_0 = 1,\ k_r = k, \tag{1}$$

$$\text{and } \forall\ i \in 1..r,\ \exists\ a(i), b(i) < i,\ k_i = k_{a(i)} + k_{b(i)}. \tag{2}$$

Thus, if we have an addition chain for k with length r, we can compute x^k with r multiplications.

[*]Institut National de Recherche en Informatique et en Automatique (INRIA), Domaine de Voluceau, B. P. 105 78153 LE CHESNAY CEDEX (France).

[†]On leave from the French Department of Defense, Délégation Générale pour l'Armement.

[‡]Departamento de Ciencias de la Computación, Universidad de Chile, Casilla 2777. Santiago. Chile. Proyecto Fondecyt 89-1096.

Many results are known [4, 11, 18, 28, 31] and some good algorithms exist, among which the *binary algorithm*, recalled in Section 3, and some of its variations (see [18] and the implementation of the 2^m method in [10]). The authors of these papers have also studied briefly the so–called *addition-subtraction chains*, defined as in (1) but with

$$\forall\, i \in 1..r,\ \exists\, a(i), b(i) \leq i,\ k_i = \pm k_{a(i)} \pm k_{b(i)}. \tag{3}$$

This idea corresponds to the evaluation of x^k by multiplications and divisions. In the case of integers, division is a costly operation and this idea does not seem to have been implemented. The situation dramatically changes when we try to compute on an elliptic curve, because in this case, division is replaced by multiplication by the inverse and that inverse is available at no cost (cf. Section 2). We are going to describe two algorithms that use addition-subtraction chains to compute x^k, the second one being an optimized version of the first one discovered by aid of a computer program. The expected gain is about 8.33% for the first one, and 11.11% for the second one.

In Section 2, we list some results about elliptic curves. Section 3 describes the binary algorithm. The new algorithms are presented in Section 4 and 5. We analyze the cost of these algorithms in Section 6. In Section 7, we compare the implementation of our algorithms to that of the 2^m method.

2. The law on an elliptic curve. Let $\mathbf{K}$ be a field of characteristic prime to 6. An elliptic curve E over $\mathbf{K}$ is a non singular algebraic projective curve of genus 1. It can be shown [7, 32] that E is isomorphic to a curve of equation

$$y^2z = x^3 + axz^2 + bz^3, \tag{4}$$

with a and b in $\mathbf{K}$. We note $E(\mathbf{K})$ the set of points of coordinates $(x : y : z)$ which satisfy (4) with $z = 1$, together with the point at infinity $O_E = (0 : 1 : 0)$.

We may define on any $E(\mathbf{K})$ an abelian law, but we shall introduce it only in the case where $\mathbf{K} = \mathbf{R}$. This law, noted additively, plays the role of multiplication in our earlier discussion of addition chains. Addition chains are thus used to compute kM on a curve, where M is a point on E.

Over $\mathbf{R}$, let us consider the curve of equation

$$y^2 = x^3 + ax + b. \tag{5}$$

An example of such a curve is represented in Figure 1 (assuming the *discriminant* $\Delta = 4a^3 + 27b^2$ to be negative).

Let M_1 and M_2 be two points on the curve. We want to associate with them a third point M_3 lying on the curve that we call the *sum* of M_1 and M_2. Let $\mathcal{D}$ be the line M_1M_2 (if $M_1 = M_2$, $\mathcal{D}$ is the tangent line). One can see that $\mathcal{D}$ intersects E at only one other point P. The point $M_3 = M_1 + M_2$ is then taken to be the reflexion of P along the x-axis. The neutral element of this law is O_E. The opposite of a point M of coordinates $(x : y : 1)$ is $-M$ whose coordinates are $(x : -y : 1)$. Thus taking the inverse of a point is essentially free.

It is easy to make this process effective. We only list the results. First, $M + O_E = O_E + M = M$. When $M_2 = -M_1$ (i.e. $x_1 = x_2$ and $y_1 = -y_2$), then $M_3 = O_E$. Otherwise, the coordinates of $M_3 = (x_3 : y_3 : 1)$ are

$$x_3 = \lambda^2 - x_1 - x_2, \tag{6}$$

$$y_3 = \lambda(x_3 - x_1) - y_1, \tag{7}$$

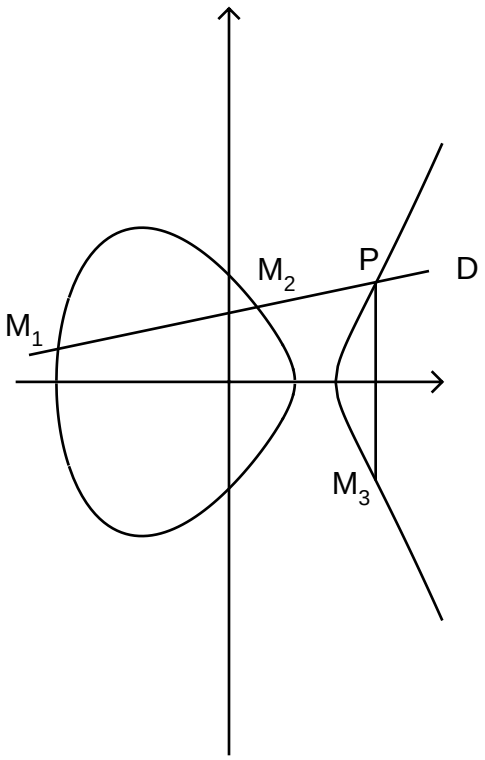

Figure 1: An elliptic curve over $\mathbf{R}$.

where

$$\lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} & \text{if } x_1 \neq x_2 \\ (3x_1^2 + a)(y_1 + y_2)^{-1} & \text{otherwise.} \end{cases}$$

One shows that the preceding equations are valid over any field $\mathbf{K}$ and that they properly define the addition on $E(\mathbf{K})$.

If $\mathbf{K}$ is not a field, but just an Artinian ring, one can also define a law on $E(\mathbf{K})$ (cf. [3]). However, when working over $\mathbf{Z}/N\mathbf{Z}$, N not prime, we do as if we were working over a field and use the preceding equations. The rationale behind this is that if we cannot invert an element, then we have a factor of N, which is the goal we usually want to reach.

The reason why elliptic curves are so attractive is that two different curves provide two different laws, contrary to the case where we work in $\mathbf{Z}/N\mathbf{Z}$, where we only have multiplication. In the case of integer factorization, different curves define distinct factorization algorithms.

It should be noted that adding two different points on a curve (over $\mathbf{Z}/N\mathbf{Z}$) requires three modular multiplications (i.e. multiplication of two integers followed by a reduction modulo N) and one gcd, and that doubling a point on the curve costs one more modular multiplication. We will come back to this problem later.

All the algorithms that use elliptic curves over finite fields require the computation of the point kM on E, where k is a large integer and M a point on the curve. These computations are to be performed as fast as possible, so this motivates the study that follows.

3. The binary algorithm and some generalizations. We just describe the algorithm. For validity and explanation, see [18].

procedure BINEXP(P, M, k)

(* $P := kM$, $k = \sum_{i=0}^{i=n} k_i\, 2^i$, $2^n \leq k < 2^{n+1}$ *)

- $Q := M$;
- **if** $k_0 = 1$ **then** $P := M$ **else** $P := O_E$;
- **for** $i = 1 \,..\, n$
  $Q := 2Q$;
  if $k_i = 1$ **then** $P := P + Q$;
- **end.**

Following [18], we introduce

$$\lambda(k) = \lfloor \log_2 k \rfloor, \tag{8}$$

$$\nu(k) = \text{Card}\,\{i \mid 0 \leq i \leq \lambda(k),\ k_i = 1\}. \tag{9}$$

Let $\mathcal{C}_{2P}$ be the cost of a "doubling" (i. e. evaluating $P + P$) and $\mathcal{C}_{P+Q}$ the cost of an "addition" (i. e. evaluating $P + Q$ with $P \neq Q$), supposed independent of P and Q. Then the cost of BINEXP is

$$\mathcal{C}_{\text{BINEXP}}(k) = \lambda(k)\,\mathcal{C}_{2P} + (\nu(k) - k_0)\,\mathcal{C}_{P+Q}. \tag{10}$$

The 2^m-ary algorithm (see [18]) consists of working with base 2^m, rather than base 2. The cost of this algorithm is roughly (see [10])

$$\mathcal{C}_{m-\text{ary}}(k) = 2^{m-1} + \lambda(k)\,\mathcal{C}_{2P} + (\nu_m(k) - k_0)\,\mathcal{C}_{P+Q}, \tag{11}$$

where $\nu_m(k)$ denotes the number of non-zero digits of k in base 2^m.

4. The first algorithm. The idea comes from the observation that long chains of 1's in the exponent c are better treated by division. For instance, we have

$$x^{15} = \frac{x^{16}}{x} = \frac{(((x^2)^2)^2)^2}{x},$$

which is more economical than the standard binary algorithm. In other words, *one replaces a block of at least two* 1*'s by a block of* 0*'s and a division.* If we imagine that we compute with exponents whose binary digits are 0, +1 and −1, then in terms of this extended representation, the algorithm corresponds to the transformation

$$1^a \mapsto 1\,0^{a-1}\text{-}1.$$

We now describe the first algorithm used to compute kM using two basic operations + and −. The idea is to construct two integers k_- and k_+ such that $k = k_+ - k_-$, but for which the evaluation of k_+M and k_-M require less operations than that of kM. We represent the algorithm by the automaton of Figure 2. It is easy to deduce from this the following recursive procedure for the computation of kM.

procedure ADDSUBCHAIN-A(P, M, k)

$Q := M$;
$P := O_E$; { the result is contained in P }
TREAT0(k);
end.

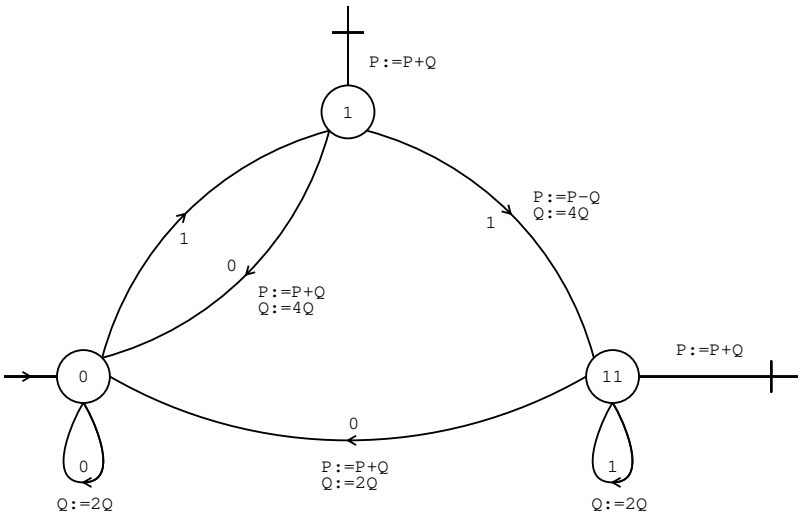

Figure 1: Finite Automaton, version A.

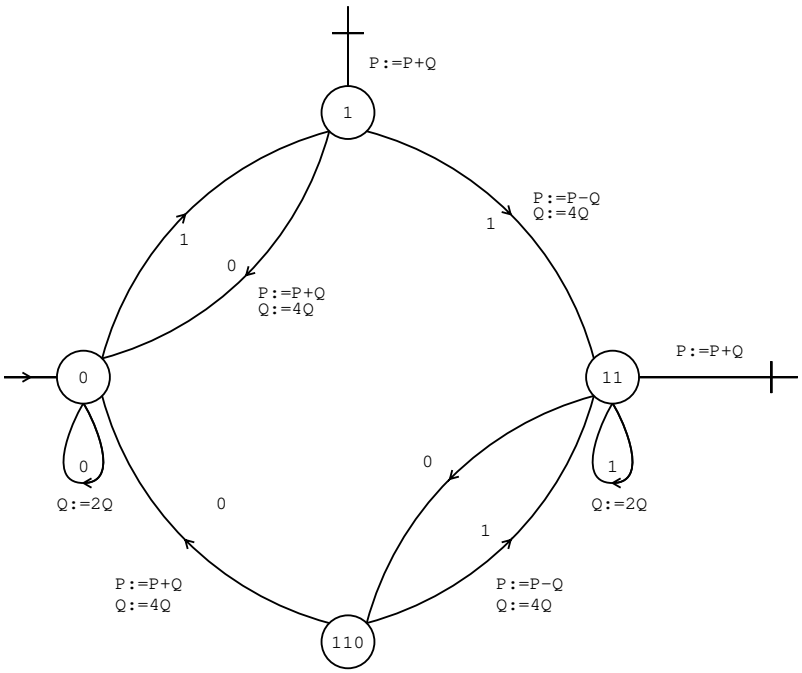

Figure 2: Finite Automaton, version B.

procedure TREAT0(k) { invariant: $R = P + kQ$ }

if $k = 0$ **then return**(P)
  else if k is even
    then $Q := 2Q$;
      TREAT0($\lfloor k/2 \rfloor$);
    else TREAT1($\lfloor k/2 \rfloor$);
end.

procedure TREAT1(k) { invariant: $R = P + (2k+1)Q$ }

if $k = 0$ **then return**($P + Q$)
  else if k is even
    then $P := P + Q$;
      $Q := 4Q$;
      TREAT0($\lfloor k/2 \rfloor$);
    else $P := P - Q$;
      $Q := 4Q$;
      TREAT11($\lfloor k/2 \rfloor$);
end.

procedure TREAT11(k) { invariant: $R = P + (k+1)Q$ }

if $k = 0$ **then return**($P + Q$)
  else if k is even
    then $P := P + Q$;
      $Q := 2Q$;
      TREAT0($\lfloor k/2 \rfloor$);
    else $Q := 2Q$;
      TREAT11($\lfloor k/2 \rfloor$);
end.

Example. Using the binary method, $k = 1101001110111$ is calculated by 12 doublings and 8 additions; in total 20 operations. Using ADDSUBCHAIN-A, we compute

k	1101001110111
k_-	100000010001
k_+	10001010001000

for which there are 13 doublings, 5 additions and 1 subtraction; in total 19 operations.
The following is an optimized iterative form of this algorithm.

procedure ADDSUBCHAIN(M, k);

$\{k = k_n \ldots k_0\}$

- $b := 0$, $P := O_E$, $Q := M$;

- **for** $i := 0..n$
  if $k_i = 0$
    then if $b \neq 0$

```
            then P := P + Q;
                 if b = 1 then Q := 2Q;
                 b := 0;
         endif
         Q := 2Q;
    else if b = 0
            then b := 1;
            else if b = 1
                    then P := P − Q;
                         Q := 2Q;
                         b := 11;
                 endif
                 Q := 2Q;
• P := P + Q;
• end.
```

5. A second algorithm. The idea is now to extend the preceding idea to the case where there are isolated 0's in the binary representation of the exponent. In this case, an isolated 0 inside a block of 1's only contributes one extra division. Using the rule of algorithm A, we have first

$$1^a\,0\,1^b \mapsto 1\,0^{a-1}\text{-}1\,1\,0^{b-1}\,1.$$

But since $-2+1=-1$, we can pile up the transformation -1 1 $\mapsto$ 0-1, whence the rule describing algorithm B

$$1^a\,0\,1^b \mapsto\ 1\,0^{a}\text{-}1\,0^{b-1}\text{-}1.$$

This process is represented by a suitable modification of automaton A to produce automaton B: We introduce state 110 that takes this into account. We will make the analysis on this automaton (see Section 6), but the actual program can be made simpler. This follows from the remark that the arcs that leave state 110 are the same as those that leave state 1. We can now build procedure ADDSUBCHAIN-B, that is the same as ADDSUBCHAIN-A except for procedure TREAT11.

procedure TREAT11(k) { invariant: $R = P + (k+1)Q$ }

if $k = 0$ **then return**$(P+Q)$
    else if k is even
        then TREAT1($\lfloor k/2 \rfloor$);
        else $Q := 2Q$;
            TREAT11($\lfloor k/2 \rfloor$);
end.

Example. With the same value of k, we find

k	1101001110111
k_-	100000001001
k_+	10001010000000

thus requiring only 13 doublings, 4 additions, and 1 subtraction; in total 18 operations.

6. Analysis of the algorithms. We recall that the expected cost of the binary method is $3/2n + O(1)$ and that of the 2^m method is $n(1+(1-2^{-m})/m)+O(1)$ when all the operations have the same cost (see [15]).

In order to analyze the algorithm (both versions), we will count the number of operations required to calculate k, assuming that $\mathcal{C}_{2P} = \mathcal{C}_{P+Q} = 1$ for the sake of simplicity in the first version and the real cost in the second version, i.e. $\mathcal{C}_{2P} = K + 4$ and $\mathcal{C}_{P+Q} = K + 3$ (see Section 2 and below). Our analysis is based on the automata shown in Figures 1 and 2. We will use an approach based on grammatical specification (see e.g. [16]). The associated bit strings belong to the language $L = \{0,1\}^*1$ where we have inverted the binary representation of a number greater than or equal to 1. The idea is to associate with each string in L a commutative polynomial in variables z and u that represents the relevant parameters: the number of bits (z) and the calculation cost of the string (u). For instance, in the binary case

$$011001 \mapsto z^6u^7.$$

The corresponding production rules (see Fig. 1) are the following

$$\begin{array}{lcllllll} A & \rightarrow & T_0 & & & & & \\ T_0 & \rightarrow & 0T_0 & + & 1T_1 & & & \\ T_1 & \rightarrow & 0T_0 & + & 1T_{11} & + & \epsilon \\ T_{11} & \rightarrow & 0T_0 & + & 1T_{11} & + & \epsilon \end{array}$$

Usual techniques can be used to obtain the generating function

$$A(z,u) = \sum_{n,m} a_{n,m} z^n u^m$$

where $a_{n,m}$ $(= [z^n u^m]A(z,u))$ is the number of strings with n bits (only $n-1$ of which are really involved here because the last is always 1), and with cost m (in number of operations). We only need to solve the following system of equations

$$\begin{array}{lcllllll} A & = & T_0 & & & & & \\ T_0 & = & zuT_0 & + & zT_1 & & & \\ T_1 & = & zu^3T_0 & + & zu^3T_{11} & + & u \\ T_{11} & = & zu^2T_0 & + & zuT_{11} & + & u \end{array}$$

where each new bit has an associated z and u whose exponent is equal to the cost given on the corresponding arcs of the automata (Fig. 1). $A(z,u)$ has been obtained by solving this system using MAPLE [8]. Since we are interested in the expected cost, we compute

$$a(z) = \left(\frac{\partial A}{\partial u}\right)\Big|_{u=1} = \frac{z^2(2+z+z^2)}{(1-2z)^2} + \frac{(z+2z^2)}{(1-2z)}$$

from this we deduce

$$\frac{1}{2^{n-1}}[z^n]\ a(z) = [z^n]\ 2a(z/2) = \frac{11}{8}n + \frac{1}{8}.$$

This compares favorably with the cost of the binary algorithm which is $3/2n + O(1)$. The relative saving in the number of additions is 25% ((1/2-3/8)/(1/2)), and there is an overall relative saving of 8.33% if all operations are considered.

In a similar manner for version B (recall that program ADDSUBCHAIN-B does not correspond precisely to Fig. 2), the production rules are

$$\begin{array}{lcllllll} B & \rightarrow & T_0 & & & & & \\ T_0 & \rightarrow & 0T_0 & + & 1T_1 & & & \\ T_1 & \rightarrow & 0T_0 & + & 1T_{11} & + & \epsilon \\ T_{11} & \rightarrow & 0T_{110} & + & 1T_{11} & + & \epsilon \\ T_{110} & \rightarrow & 0T_0 & + & 1T_{11} & & \end{array}$$

Introducing the respective costs for elliptic curves, the corresponding equations are

$$\begin{array}{lcllllll} B & = & T_0 & & & & & \\ T_0 & = & zu^{K+4}\,T_0 & + & zT_1 & & & \\ T_1 & = & zu^{3K+11}\,T_0 & + & zu^{3K+11}\,T_{11} & + & u^{K+3} \\ T_{11} & = & zT_{110} & + & zu^{K+4}\,T_{11} & + & u^{K+3} \\ T_{110} & = & zu^{3K+11}\,T_0 & + & zu^{3K+11}\,T_{11} & & \end{array}$$

From which we found, using MAPLE, the expected cost

$$[z^n]2b(z/2) = \frac{4K+15}{3}n + \frac{4K+9}{9} + O(2^{-n}).$$

With version B we achieve a relative saving of 33% with respect to additions, and 11.11% if all operations are considered.

We must note that version B of the algorithm was discovered with the help of $\Lambda\Upsilon\Omega$ [12], a powerful system designed to perform automatic analysis of a broad class of algorithms, developed at INRIA.

We now give the results for elliptic curves when we have $\mathcal{C}_{2P} = K + 4$ and $\mathcal{C}_{P+Q} = K + 3$, where K is the cost of a gcd over the integers, the unit being the time of a multiplication modulo N. For example, in [5], the author takes $K = 30$. P. Zimmermann kindly computed the costs of the algorithms given this assumption, using $\Lambda\Upsilon\Omega$. Here are the results

Binary algorithm	$(3K + 11)/2\, n + O(1)$
2^m-ary algorithm	$(K + 4 + (K + 3)(1 - 2^{-m})/m)\, n + O(1)$
Algorithm A	$(11K + 41)/8\, n + O(1)$
Algorithm B	$(4K + 15)/3\, n + O(1)$

Our algorithms require no extra-storage, contrary to the 2^m method, which needs to store $2 \times 2^{m-1}$ n-bits integers (if we work over $\mathbf{Z}/N\mathbf{Z}$ with N an n-bit integer).

7. Implementation and conclusions. The first author has used the second algorithm in his implementation of the so called Atkin's test [26] for primality testing. The overall gain is about 3% in time for 100-digit numbers and 2.7% for 300-digit numbers.

It should be possible to combine the idea of addition-subtraction chains with that of the 2^m-ary algorithm. As for now, it is not clear how we could do that, the problem being that our algorithm must keep track of what happened a few bits before.

Acknowledgments. The second author would like to express his gratitude to INRIA for an invited visit during which his work on the subject was done. Many thanks are due to P. Flajolet. This work has also benefitted from financial assistance from the French-Chilean cooperation program whose help is gratefully acknowledged.

Both authors would like to acknowledge the help of P. Zimmermann with the $\Lambda\Upsilon\Omega$ system and that of P. Flajolet who very carefully read the first version of the paper and made valuable remarks.

References

[1] A. O. L. Atkin. Manuscript.

[2] F. Bergeron, J. Berstel, S. Brlek, C. Duboc. Addition chains using continued fractions. *Journal of Algorithms*, **10**, 3, September 1989, pp. 403–412.

[3] W. Bosma. Primality testing using elliptic curves. Report 85-12, Math. Instituut, Universeit van Amsterdam.

[4] A. Brauer. On addition chains. *Bull. Amer. Math. Soc.*, **45**, 1939, pp. 736-739.

[5] R. P. Brent. Some integer factorization algorithms using elliptic curves. Research Report CMA-R32-85, The Australian National University, Canberra, 1985.

[6] J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, S. S. Wagstaff, Jr. *Factorizations of* $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ *up to high powers.* Contemporary Mathematics, **22**, AMS, 1983.

[7] J. W. S. Cassels. Diophantine equations with special references to elliptic curves. *J. London Math. Soc.*, **41**, 1966, pp. 193-291.

[8] B. W. Char, K. O. Geddes, G. H. Gonnet, S. M. Watt. *MAPLE Reference Manual, Fourth Edition.* Symbolic Computation Group, Department of Computer Science, University of Waterloo, 1985.

[9] D. V. Chudnovsky, G. V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. Research report RC 11262, IBM, Yorktown Heights, 1985.

[10] H. Cohen, A. K. Lenstra. Implementation of a new primality test. *Math. Comp.*, **48**, 177, 1987, pp. 103-121.

[11] P. Erdős. Remarks on number theory III: on addition chains. *Acta Arithmetica*, **6**, 1960, pp. 77-81.

[12] Flajolet, P., Salvy, B., and Zimmermann, P. Lambda–Upsilon–Omega: An assistant algorithms analyzer. In *Applied Algebra, Algebraic Algorithms and Error–Correcting Codes* (1989), T. Mora, Ed., vol. 357 of *Lecture Notes in Computer Science*, pp. 201–212. (Proceedings AAECC'6, Rome, July 1988).

[13] Flajolet, P., Salvy, B., and Zimmermann, P. Lambda–Upsilon–Omega: The 1989 Cookbook. Research Report 1073, Institut National de Recherche en Informatique et en Automatique, August 1989. 116 pages.

[14] S. Goldwasser, J. Kilian. Almost all primes can be quickly certified. *Proc. 18th ACM Symp. on the Theory of Compt.*, Berkeley, 1986, pp. 316-329.

[15] G. H. Gonnet. *Handbook of Algorithms and Data Structures.* Addison-Wesley. 1984.

[16] D. H. Greene. Labelled Formal Languages and Their Uses. Technical Report STAN-CS-83-982, Stanford University, 1983.

[17] B. S. Kaliski, Jr. A pseudo-random bit generator based on elliptic logarithms. *Proc. Crypto 86*, p.13-1,13-21.

[18] D. E. Knuth. *Seminumerical algorithms.* The Art of Computer Programming, T. II, Addison-Wesley.

[19] N. Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, **48**, 177, 1987, p.203-209.

[20] H. W. Lenstra, Jr. Factoring with elliptic curves. Report 86-18, Math. Inst., Univ. Amsterdam, 1986.

[21] H. W. Lenstra, Jr. Elliptic curves and number theoretic algorithms. Report 86-19, Math. Inst., Univ. Amsterdam, 1986.

[22] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Math.*, **126**, 1987, pp. 649-673.

[23] D. P. McCarthy. The optimal algorithm to evaluate x^n using elementary multiplication methods. *Math. Comp.*, **31**, 137, 1977, p.251-256.

[24] D. P. McCarthy. Effect of improved multiplication efficiency on exponentiation algorithms derived from addition chains. *Math. Comp.*, **46**, 174, 1986, p.603-608.

[25] P. L. Montgomery. Modular multiplication without trial division. *Math. Comp.*, **44**, 170, 1985, p.519-521.

[26] F. Morain. Implementation of the Atkin-Goldwasser-Kilian test. INRIA Research Report 911, 1988.

[27] F. Morain, J. Olivos. Un algoritmo de Evaluación de Potencia utilizando Cadenas de Suma y Resta. *Proc. XIV Conference Latinoamericana de Informatica (CLEI, Expodata)*, Buenos Aires, September 1988.

[28] J. Olivos. On Vectorial Additions Chains. *J. of Algorithms*, **2**, 1981, p.13-21.

[29] J. M. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Phil. Soc.*, **76**, 1974, p.521-528.

[30] R. L. Rivest, A. Shamir, L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. of the ACM*, **21**, 2, 1978, p.120-126.

[31] A. Schönhage. A lower bound for the length of addition chains. *Theor. Comput. Science*, **1**, 1, 1975, p.1-12.

[32] J. T. Tate. The arithmetic of elliptic curves. *Inventiones Math.*, **23**, 1974, p.179-206.