ATKIN'S TEST: NEWS FROM THE FRONT

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

Département de Mathématiques
Université Claude Bernard
69622 Villeurbanne CEDEX (France)

Abstract

We make an attempt to compare the speed of some primality testing algorithms for certifying 100-digit prime numbers.

1. Introduction. The implementation of several public-key cryptosystems requires the ability to build *large* primes as fast as possible [13, 2]. Several authors [29, 12, 15, 22] have studied this problem and given some good algorithms, which give primes having special forms. Our purpose is to explain how to test a *random* integer for primality.

One possible solution to this problem is to use *probable* primes, recognized by a probabilistic primality testing algorithm, such as Miller-Rabin's [19]. This test is very fast and almost surely yields prime numbers. (For a philosophical interpretation of this test, see [4].)

Another way is to use *deterministic* primality testing algorithms which yield a proof for a number to be a prime. The first general purpose deterministic algorithm was introduced by Adleman, Rumely and Pomerance [1] and refined by H. Cohen, H. W. Lenstra (Jr.) and A. K. Lenstra [10, 11] (and more recently by Bosma and van der Hulst [6]). It gives good running times (on a huge computer). However, the proof given by their program is *yes* or *no* and the only way for someone else to verify the results is to rewrite and rerun the entire program. One of the most recent primality testing algorithm, due to Atkin [3], uses elliptic curves and generalizes the old theorems of Fermat on primality. The author used his own implementation of this algorithm to prove the primality of about fifty large numbers from Cunningham's tables [8] (thus finishing the list of probable primes that were waiting to be certified), the largest one being the 564-digit cofactor of F_{11} [24], and two other large primes, namely S_{1493} (572 digits) and S_{1901} (728 digits), where

$$S_p = \frac{(1+\sqrt{2})^p + (1-\sqrt{2})^p}{2}.$$

For this algorithm, the work needed to check the results is far less than that of establishing proofs.

The purpose of this paper, after a brief description of Atkin's test, is to attempt to compare these algorithms with respect to the following questions:

1. How long does it take to test a 100-digit number for primality?

2. How fast are these algorithms compared to the algorithm of Miller?

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

3. What kind of proof do we get? How long does it take to verify it?

It should be noted that we only describe the implementation of the algorithm that is needed to test 100-digit numbers. Many other strategies are used when dealing with larger numbers (see [24] or the forthcoming papers [23, 26]).

Notations. In the sequel, N will denote an odd integer to be tested for primality, **MR** Miller-Rabin's algorithm, $\mathbf{CL_2}$ Cohen-Lenstra's, and **ATK** Atkin's.

2. A brief description of Atkin's test. *(2.1) Elliptic curves.* 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 [9, 33] that E is isomorphic to a curve with equation:

$$y^2z = x^3 + axz^2 + bz^3, \qquad (1)$$

with a and b in $\mathbf{K}$. The *discriminant* of E is $\Delta = -16(4a^3 + 27b^2)$ and the *invariant* is

$$j = 2^8 3^3 \frac{a^3}{4a^3 + 27b^2}.$$

We write $E(\mathbf{K})$ for the set of points with coordinates $(x : y : z)$ which satisfy (1) with $z = 1$, together with the point at infinity: $O_E = (0 : 1 : 0)$. We will use the well-known *tangent-and-chord* addition law on a cubic [18] over a finite field $\mathbf{Z}/N\mathbf{Z}$ (see [16] for a justification).

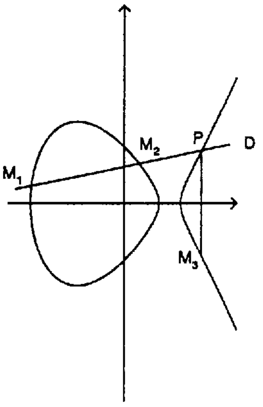


Figure 1: An elliptic curve over $\mathbf{R}$

In order to add two points $M_1 = (x_1, y_1)$ and $M_2 = (x_2, y_2)$ on E resulting in $M_3 = (x_3, y_3)$, the equations are

$$\begin{cases} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{cases}$$

where

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

We can compute kP using the binary method (see also [11]) or addition-subtraction chains [28].

(2.2) Primality testing. Let us recall one of the converses of Fermat's theorem.

Theorem 1 *Let a be such that* $\gcd(a, N) = 1$, *q a prime divisor of $N - 1$. If*

$$a^{N-1} \equiv 1 \bmod N \text{ and } \gcd(a^{(N-1)/q} - 1, N) = 1$$

then each prime divisor p of N satisfies: $p \equiv 1 \bmod q$.

Corollary 1 *If $q > \sqrt{N}$ then N is prime.*

A similar theorem can be stated for elliptic curves.

Theorem 2 (**[14, 21]**) *Let N be an integer greater than 1 and prime to 6. Let E be an elliptic curve over $\mathbf{Z}/N\mathbf{Z}$, m and s two integers such that $s \mid m$. Suppose we have found a point P on E that satisfies $m\,P := O_E$, and that for each prime factor q of s, we have verified that $\frac{m}{q}P \neq O_E$. Then if p is a prime divisor of N, $\#E(Z/pZ) \equiv 0 \bmod s$.*

Corollary 2 *If $s > (\sqrt[4]{N}+1)^2$, then N is prime.*

In order to use the preceding theorem, we need to compute the number of points m. This process is far from trivial in general (see [32]). From a practical point of view, it is desirable to use deep properties of elliptic curves over finite fields. This involves the theory of complex multiplication and class fields and requires a lot of theory [24]. We can summarize the principal properties:

Theorem 3 *Every elliptic curve E $\bmod p$ has complex multiplication by the ring of integers of an imaginary quadratic field $K = \mathbf{Q}(\sqrt{-D})$.*

From a very down-to-earth point of view, this comes down to saying:

- p splits in K: $(p) = (\pi)\,(\pi')$ in K;
- $H_D(j(E)) \equiv 0 \bmod p$ for a fixed $H_D(X)$ in $\mathbf{Z}[X]$;
- $m = (\pi - 1)(\pi' - 1) = p + 1 - t$, where $|t| \leq 2\sqrt{p}$ (Hasse).

The computation of the polynomials H_D is dealt with in [24] and [25]: it requires some 1000 lines of MAPLE code. As a result, I have a list of 575 discriminants (those with $h \leq 10$ and some with $h = 12$), thus providing about 1158 potential number of points.

3. Atkin's algorithm. We now explain how the preceding theorems are used in a *factor and conquer* algorithm similar to the DOWNRUN process of [34]. The first phase of the algorithm consists in finding a sequence $N_0 = N > N_1 > \cdots > N_k$ of probable primes such that: N_{i+1} prime $\Longrightarrow N_i$ prime. The second then proves that each number is prime, starting from N_k.

Procedure SEARCHN

1. $i := 0$; $N_0 := N$;
2. find a fundamental discriminant $-D$ such that (N_i) splits as the product of two principal ideals in $\mathbf{Q}(\sqrt{-D})$;
3. for each solution of $(N_i) = (\pi)(\pi')$, find all factors of $m_\pi = (\pi - 1)(\pi' - 1)$ less than a given bound B and let N_π be the corresponding cofactor;
4. **if** one of the N_π is a probable prime **then** set $N_{i+1} := N_\pi$, store $\{N_i, D, \pi, m\}$ set $i := i + 1$, and go to step 2 **else** go to step 3.
5. **end.**

In Step 2, we use lattice reduction (see [24]). In Step 3, we use a sieve to find all factors less than 2^{15}, which is enough for our purpose (that is testing the primality of 100-digit integers). The sieving process is done as follows (this generalizes a trick described in [7, Section 7, Rem. 1] and [11]):

Procedure Sieve

```
1. for $i = 1..k$ do $RES[i] := (N+1) \bmod p_i$;
2. $N \pm 1$ tests:
  for $i = 1..k$
    • if $RES[i] = 0$ then $p_i \mid N+1$
    • if $RES[i] = 2$ then $p_i \mid N-1$
3. $m_\pm = N + 1 \pm t$, $|t| \leq 2\sqrt{N}$:
  for $i = 1..k$
    • $r := t \bmod p_i$
    • if $RES[i] = \ \ r$ then $p_i \mid m_-$
    • if $RES[i] = -r$ then $p_i \mid m_+$
```

If one of the steps of procedure SEARCHN cannot be achieved, this means that either one of our N_i is indeed composite or that this is a difficult number (see [26]).

The second phase consists in proving that the numbers N_i are indeed primes. This is done as follows:

Procedure PROOF

```
for $i = k..0$
1. compute a root $j$ of $P_{D_i}(X) \bmod N_i$ using Berlekamp's algorithm if $\deg(P) > 4$, Shanks's if
   $\deg = 2$, Cailler-Williams' if $\deg = 3$, and Skolem's otherwise (see [27]);
2. let $k := j/(1728-j) \bmod N_i$, $a := 3k$, $b := 2k$: $E(a,b)$ has for equation $y^2 \equiv x^3+ax+b \bmod N_i$;
   choose a point $P$ on $E(a,b)$ and compute $Q = m_i P$; if $Q \neq O_E$ then choose a non residue $c$
   and set: $a := ac^2$, $b := bc^3$.
3. verify the condition of theorem (2).
```

The same remarks can be made if we cannot complete our task. It has been observed that as soon as we can complete Phase 1, Phase 2 is no problem (apart from the execution time).

4. Implementation and empirical comparisons. I have implemented **ATK** on a SUN 3/60 (12 Mo) using the BigNum package described in [17]. This package includes about 700 lines of assembly code together with 1500 lines of Le-Lisp (or C). My program is written in Le-Lisp.

(4.1) A brief comparison with $\mathbf{CL}_2$. In [11], the authors describe the implementation of $\mathbf{CL}_2$ on a CDC Cyber 170/750. They used a 47-bit arithmetic and they gave the time for doing elementary operations on *multiples* (i.e. 8 words of 47 bits) and *doubles* (16 words).

We can attempt to compare the speeds of these two arithmetics by measuring the time needed on a SUN to do the same operations on numbers having equal numbers of bits: a multiple consists of 12 words of 32 bits and a double of 24 words of 32 bits. We list below these times in milliseconds.

operation	CDC	SUN 3/60
multiple + multiple	0.014	0.260
multiple $\times$ multiple	0.070	1.270
double mod multiple	0.200	2.850

We can satisfy ourselves with the crude statement that our arithmetic is 15 times slower than that of Cohen and Lenstra.

Following the same line, we can compare the time needed to test 100-digit numbers for primality. We now describe the protocol we used (it is the protocole of [11] without the testing for small factors).

Protocol for measuring the time for algorithm $\mathcal{A}$

```
repeat 20 times

1. select a random odd number $n$;
2.if $n$ passes four times MR
    then record the corresponding time;
    else $n := n + 2$; go to 2.
3.if $n$ passes algorithm $\mathcal{A}$
    then ouput "Prime"; record the time;
    else output "Composite".
```

At the end of the process, we compute statistics. In the following table, we have listed the times for the fastest execution, the slowest, the mean running time and the standard deviation. There are two columns with "4× **MR** " which corresponds to four executions of the two versions of algorithm **MR**, one along with **CL**$_2$, the other with **ATK**. Times are in seconds.

	CL$_2$	4×**MR**	ratio	**ATK**	4×**MR**	ratio
minimum	26.031	0.544		350.4	4.0	
maximum	75.416	0.602		1042.3	5.0	
mean	50.442	0.567	88.92	661.0	4.4	150.23
standard deviation	15.203	0.015		197.0	0.4	

We now see that our **MR** is about 8 times slower, but **ATK** algorithm is less than 14 times slower than **CL**$_2$. These arguments are not very strong, but gives some hints on the relative behaviors of **CL**$_2$ compared to **ATK**.

For the sake of completeness, I list in the following table the corresponding times for the implementation of **CL**$_2$ by A. K. Lenstra on a Cray I [20].

	CL$_2$	4×**MR**
minimum	3.822	0.061
maximum	9.174	0.057
mean	7.047	0.058
standard deviation	1.549	0.001

We can also proceed to give in Table 1 the time needed to test a number of d words of 32 bits with my program, for $d = 2(2)20$. Time are in seconds. The first line is concerned with **ATK**, the second with the number of steps in procedure SEARCHN.

These data are reported in Figure 2. They suggest the following approximation for the average running time of our program (in seconds if d is the number of 32-byte words):

$$T_{ATK}(d) \approx 0.27 \times d^{3.41}.$$

Using the data in [11], we can compute a similar approximation for the running time of **CL**$_2$ for numbers from 100 to 200 digits. This yields:

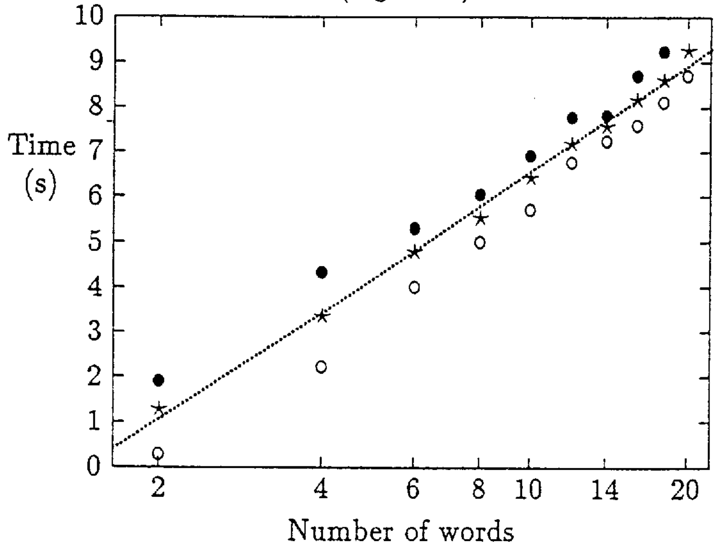
$$T_{CL}(d) \approx 0.024 \times d^{3.28}.$$

If we use the theoretical running time of **CL**$_2$, we find

$$T_{th}(d) \approx 2 \times d^{1.65 \log \log d}.$$

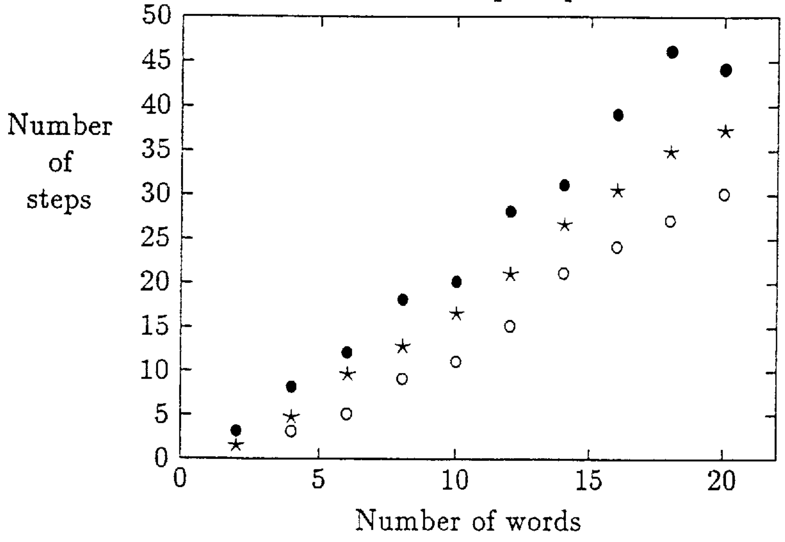
We can draw some conclusions regarding this comparison between **CL**$_2$ and **ATK**. It seems that **CL**$_2$ is slightly faster for this range of numbers (100 to 200 digits). It is worth noting that the implementation of **CL**$_2$ was very optimized for this range, as mine is not (at least for the time

Time needed to test a *d*-word number
(log scale)



○: minimum time
●: maximum
⋆: mean

Number of steps required



d	min	max	mean	st. dev.
2	1.3	6.5	3.6	1.5
	0	3	1.5	0.9
4	9.0	74.8	28.4	15.8
	3	8	4.7	1.4
6	54.1	195.9	119.9	37.6
	5	12	9.6	1.7
8	146.1	413.4	250.6	73.9
	9	18	12.8	2.6
10	294.7	983.9	608.9	160.8
	11	20	16.5	2.7
12	846.6	2334.5	1312.0	384.0
	15	28	21.0	3.6
14	1369.4	2433.5	1937.0	276.5
	21	31	26.6	2.7
16	1945.1	5886.5	3513.1	1077.8
	24	39	30.5	3.9
18	3289.8	10009.5	5454.9	1469.1
	27	46	34.9	5.3
20	5871.0	17845.8	10405.6	2772.2
	30	44	37.3	4.3

Table 1: Time needed to test a d-word number for primality

being). My arithmetic is able to deal with arbitrary large numbers and I was able to certify numbers from 250 to 700 digits. (For these sizes, I use special algorithms, including Karatsuba's multiplication algorithm as implemented by P. Zimmermann at INRIA.) On the contrary, the size of integers used on the CDC must be fixed at compile time [20].

(4.2) Further remarks on my implementation. In a typical run of my program, three quarters of the time are needed to complete the first phase of the algorithm and one quarter for the second (again for a 100-digit number). The most time consuming part of the algorithm is the factorization of the number of points m.

5. Generalized certificate of primality. Certificates of primality have been introduced by Pratt [31]. Recently, Pomerance studied some certificates using elliptic curves [30]. It is very easy to generalize the work of Pratt. A certificate generated by algorithm **ATK** consists in blocks of numbers. Each block has the following structure:

n_i

type

P
R
O
O
F

0

where n_i is the number to be tested, *type* giving the type of theorem used to show the primality of n_i. This is an integer, chosen as follows:

-1	:	use of the factors of $n_i - 1$,
1	:	use of the factors of $n_i + 1$,
D	:	an integer ($D > 2$) used in **ATK**.

To each of the types corresponds a list of numbers used to complete the proof of n_i being prime, whenever the following block is valid. For instance, the format of a proof using elliptic curves is the following:

D	the discriminant used
m	the number of points on the curve
r_0	
$\cdots$	the factors of m
r_k	
0	
a	the curve is E: $y^2 = x^3 + ax + b$
b	E has complex multiplication by $\mathcal{O}(\sqrt{-D})$
x	the coordinates of a point P on the curve
y	
f_1	
$\cdots$	the factorization of the order of P on E
f_l	

For example, here is the first block of proof for a 50-digit prime:

```
n0=35090920174233837395447134480305116522935098213281
D=4
m=35090920174233837395447125326861763110277715724842
r1=937
   853
   13
   2
0
a=6
b=0
x=7778128793230599416235595023534938267181834875
y=18796494177062591514397495874290295720055455015935
f=16886233357214610359515335621502364304351 97
0
```

It is also possible to give a very short label to a certificate, which I call a *primality path*. It consists in some brief informations on the discriminants used and the actual value of m choosen (see [26]). For example, for the above mentioned prime, the path is: $4d1 - 1 + 1 + 3f1 + 1+$.

An independant verifier can check the proof by simply coding the necessary basic operations on elliptic curves.

The size of the whole program is about 230 kbytes (about 6500 lines of Le-Lisp) using 370 kbytes of data (the polynomials H_D and the primes below 2^{15}). The code needed for verifying a certificate is about 74 kbytes. Moreover, the time needed to verify a certificate for a 100-digit prime is about one fifth the time needed to build it.

6. Further directions of research. In the future, I intend to optimize my code a little further, namely using special techniques for gcd's, ... It will also be interesting to use the hardware multiplier built by J. Vuillemin and his team at INRIA and DEC-PRL [5] to speed up the modular operations.

Apart from this technological research, it is possible to give some hints concerning the analysis of the algorithm and also what a bad number is for **ATK**.

7. Conclusions. If we try to answer our original questions in brief, we see that $\mathbf{CL}_2$ gives a succinct proof in a short time on a huge computer, taking 90 times more time than four executions of **MR**; on the other hand, **ATK** gives a lengthy proof that can be verified in a small fraction of the time needed to establish it, and it works well on a small computer, achieving a ratio of 150 with 4×**MR**, which is not too bad compared to $\mathbf{CL}_2$, since the arithmetic is approximately fifteen times slower.

Acknowledgments. I wish to thank A. K. Lenstra for some useful comments on the first version of this article and S. S. Wagstaff, Jr. for his careful reading of this article.

References

[1] L. M. Adleman, C. Pomerance, and R. S. Rumely. On distinguishing prime numbers from composite numbers. *Annals of Math.*, 117:173–206, 1983.

[2] L. M. Adleman, R. L. Rivest, and A. Shamir. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21(2):120–126, 1978.

[3] A. O. L. Atkin. Manuscript. August 1986.

[4] P. Beauchemin, G. Brassard, C. Crépeau, C. Goutier, and C. Pomerance. The generation of random numbers that are probably prime. *J. Cryptology*, 1:53–64, 1988.

[5] P. Bertin, D. Roncin, and J. Vuillemin. Introduction to programmable active memories. In *Proc. of the Internat. Conf. on Systolic Arrays*, 1989.

[6] W. Bosma and M.-P. van der Hulst. Faster primality testing. In *Proc. Eurocrypt '89*, 1989.

[7] J. Brillhart, D. H. Lehmer, and J. L. Selfridge. New primality criteria and factorizations of $2^m \pm 1$. *Math. of Comp.*, 29(130):620–647, 1975.

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

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

[10] H. Cohen and Jr H. W. Lenstra. Primality testing and Jacobi sums. *Math. of Comp.*, 42(165):297–330, 1984.

[11] H. Cohen and A. K. Lenstra. Implementation of a new primality test. *Math. of Comp.*, 48(177):103–121, 1987.

[12] C. Couvreur and J.J. Quisquater. An introduction to fast generation of large prime numbers. *Philips J. Research*, 37:231–264, 1982.

[13] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22-6, nov 1976.

[14] S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. In *Proc. 18th STOC*, pages 316—329, Berkeley, 1986.

[15] D. Gordon. Strong primes are easy to find. In *Proc. Eurocrypt '84*, pages 216–223. Springer, 1984.

[16] Jr. H. W. Lenstra. Factoring integers with elliptic curves. *Annals of Math.*, 126:649–673, 1987.

[17] J.-C. HERVÉ, F. MORAIN, D. SALESIN, B. SERPETTE, J. VUILLEMIN, AND P. ZIMMERMANN. Bignum: A portable and efficient package for arbitrary precision arithmetic. Rapport de Recherche 1016, INRIA, avril 1989.

[18] D. HUSEMXEOLLER. *Elliptic curves*, volume 111 of *Graduate Texts in Mathematics*. Springer, 1987.

[19] D. E. KNUTH. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley, 1981.

[20] A. K. LENSTRA. Data concerning the implementation of the Jacobi sums algorithm on a cray-1. Personnal Communication, April 1989.

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

[22] U. M. MAURER. Fast generation of secure RSA-products with almost maximal diversity. In *Proc. Eurocrypt '89*, 1989.

[23] F. MORAIN. Distributed primality proving. In preparation.

[24] F. MORAIN. Implementation of the Atkin-Goldwasser-Kilian primality testing algorithm. Rapport de Recherche 911, INRIA, Octobre 1988.

[25] F. MORAIN. Construction of Hilbert class fields of imaginary quadratic fields and dihedral equations modulo p. Rapport de Recherche 1087, INRIA, Septembre 1989.

[26] F. MORAIN. Elliptic curves and primality proving. In preparation, 1989.

[27] F. MORAIN. Résolution d'équations de petit degré modulo de grands nombres premiers. Rapport de Recherche 1085, INRIA, Septembre 1989.

[28] F. MORAIN AND J. OLIVOS. Speeding up the computations on an elliptic curve using addition-subtraction chains. Rapport de Recherche INRIA 983, INRIA, Mars 1989.

[29] D. A. PLAISTED. Fast verification, testing and generation of large primes. *Theoretical Computre Science*, 9:1–16, 1979.

[30] C. POMERANCE. Very short primality proofs. *Math. of Comp.*, 48(177):315–322, 1987.

[31] V. R. PRATT. Every prime has a succint certificate. *SIAM J. Comput.*, 4:214–220, 1975.

[32] R. SCHOOF. Elliptic curves over finite fields and the computation of square roots mod p. *Math. of Comp.*, 44:483–494, 1985.

[33] J. T. TATE. The arithmetic of elliptic curves. *Inventiones Math.*, 23:179–206, 1974.

[34] M. C. WUNDERLICH. A performance analysis of a simple prime-testing algorithm. *Math. of Comp.*, 40(162):709–714, 1983.