J. Cryptology (2005) 18: 63–76
DOI: 10.1007/s00145-004-0218-8

Journal of
CRYPTOLOGY

Analysis of the Sliding Window Powering Algorithm

Henri Cohen
Laboratoire A2X,
U.M.R. 9936 du C.N.R.S., Université Bordeaux I,
351 Cours de la Libération,
33405 Talence Cedex, France

Communicated by Arjen Lenstra

Received 8 July 2002 and revised 9 June 2003
Online publication 16 March 2004

Abstract. We analyze precisely the parameters entering the sliding window powering method, and apply this to the case of an elliptic curve over a large finite prime field.

Key words. Binary powering, Addition chain, Window method.

1. The Sliding Window Powering Method

Let G be a group, let $g \in G$, and let N be a positive integer. We want to compute g^N efficiently. Several issues enter into this problem, in particular the representation used for the elements of G and the operations between them, and the addition or addition–subtraction chain (if the inverse in G can be computed inexpensively) used for N. In this paper we are mainly concerned with the latter problem.

If N is fixed and not too large, we can try to find the optimal addition or addition–subtraction chain for N. However, if N is large or even if N is not too large but is allowed to vary, this is not practical anymore and we must use reasonable approximations to the optimal chain.

A first quite reasonable idea is the use of the very simple but efficient binary algorithm (see [Kn] or [C]). Denote by D the cost of a squaring in G, by A the cost of a multiplication in G, and let $n = 1 + \lfloor \log_2 N \rfloor$ be the number of bits of N. The total cost of computing g^N by the binary algorithm is approximately equal to $nD + (n/2)A$. Note that we use the letters D (for doubling) and A (for addition) as if the group was written additively, because we reserve the use of the letters S (for squaring) and M (for multiplication) for more basic operations such as base field operations if, for example, G is an elliptic curve over a field.

It is well known that the binary algorithm can be improved by using base 2^e for a suitable e (see [P], [Y], or [C, Algorithm 1.2.4]). The idea is as follows: we write N in

base 2^e and we precompute $g^1, g^3, \ldots, g^{2^e-1}$. If

$$N = (((x_v 2^e + x_{v-1})2^e + x_{v-2})2^e + \cdots + x_1)2^e + x_0,$$

with $0 \le x_i < 2^e$, we compute g^N by iterating operations of the form $a \leftarrow a^{2^e} g^{x_i}$. If $x_i = 2^{k_i} y_i$ with y_i odd, this is done by writing $a \leftarrow (a^{2^{e-k_i}} g^{y_i})^{2^{k_i}}$, and since the g^{y_i} have been precomputed, this requires e squarings and one multiplication in G. Thus the total number of squarings in the computation of g^N is still approximately equal to n (it is in fact slightly less), but the total number of multiplications is now equal to $2^{e-1} + v$. Since v is approximately equal to n/e, we obtain a total computing cost of approximately $nD + (n/e + 2^{e-1})A$.

It is clear that this becomes rapidly better than the binary algorithm. In fact, choosing simply $e = 3$, we have $n/3 + 4 < n/2$ as soon as $n \ge 25$. It is also not difficult to find the optimal value of e for given n by minimizing the function $n/e + 2^{e-1}$. A similar analysis is valid if we allow additions and subtractions.

The above method, consisting essentially in looking at the bits of N through a window of width e (some authors call $w = e-1$ the width of the window, but we stick to e here), is naturally called the window method. A perhaps surprising fact (which will be included in a future reprinting of [C]), is that there is a very easy modification of the window method (which may be called the sliding window method) which leads to a better performance with essentially no extra work. This method is based on the following lemmas.

Lemma 1.1. *Let $e \ge 1$. There exist a unique representation of N as*

$$N = ((((x_v 2^{k_v} + x_{v-1})2^{k_{v-1}} + x_{v-2})2^{k_{v-2}} + \cdots + x_1)2^{k_1} + x_0)2^{k_0},$$

satisfying $1 \le x_i \le 2^e - 1$, x_i odd, and $k_i \ge e$ for $i > 0$.

When subtractions are also allowed, this lemma is modified as follows:

Lemma 1.2. *Let $e \ge 1$. There exist a unique representation of N as*

$$N = ((((x_v 2^{k_v} + x_{v-1})2^{k_{v-1}} + x_{v-2})2^{k_{v-2}} + \cdots + x_1)2^{k_1} + x_0)2^{k_0},$$

satisfying $-2^{e-1} + 1 \le x_i \le 2^{e-1} - 1$, x_i odd, $x_v > 0$, and $k_i \ge e$ for $i > 0$.

Both lemmas are easy to prove, and it is also easy to give algorithms for finding the x_i and k_i.

The point of Lemmas 1.1 and 1.2 is that the number of group squarings and the number of group multiplications during the precomputations will be the same, but on the other hand a cursory analysis (see, for example, [CMO1]) shows that the average order v of group multiplications during the main computation will be approximately equal to $n/(e+1)$ instead of n/e. The purpose of this paper is to analyze precisely the behavior of this sliding window algorithm (see also [Ko] and [MO]). The analysis is useful also to analyze further more marginal improvements, such as the computation of the initial quantity $g^{N_{v-1}}$ with $N_{v-1} = x_v 2^{k_v} + x_{v-1}$. Since the context in which this problem appeared was the case where G is the group of points of an elliptic curve over a

finite field, for which the cost of subtraction is identical to the cost of addition, we give the detailed analysis for the addition–subtraction algorithm based on Lemma 1.2. The analysis for the addition-only algorithm based on Lemma 1.1 is essentially identical, and we only give the results.

2. Basic Mathematical Analysis

We fix a positive integer e (the window size), a positive integer n, and we let N range uniformly among the integers having exactly n bits, in other words $2^{n-1} \le N < 2^n$. Let

$$N = ((((x_v 2^{k_v} + x_{v-1})2^{k_{v-1}} + x_{v-2})2^{k_{v-2}} + \cdots + x_1)2^{k_1} + x_0)2^{k_0}$$

be the unique representation of N given by Lemma 1.2, let $d(N) = \sum_{0 \le i \le v} k_i$ be the number of group squarings, and let $a(N) = v$ be the number of group multiplications, so that the cost of the sliding window algorithm is D (for the computation of g^2) plus $(2^{e-2} - 1)A$ (for the computation of $g^3, g^5, \ldots, g^{2^{e-1}-1}$) plus $d(N)D + a(N)A$ (for the main computation), hence in total a cost of

$$C(N) = (d(N) + 1)D + (a(N) + 2^{e-2} - 1)A.$$

We analyze the quantities $d(N)$ and $a(N)$ (and similar quantities) in turn. We first need preliminary results on a special type of linear recurrence.

2.1. *Analysis of the Recurrence* $2u(n + e) - u(n + e - 1) - u(n) = 0$

The purpose of this section is to prove the following proposition:

Proposition 2.1. *Let $u(n)$ be a sequence satisfying the linear recursion*

$$2u(n + e) - u(n + e - 1) - u(n) = 0 \qquad \text{for} \quad n \ge 1.$$

Then there exists $\rho > 1$ such that as $n \to \infty$,

$$u(n) = \frac{1}{e+1}\left(2u(e) + \sum_{i=1}^{e-1} u(i)\right) + O(\rho^{-n}).$$

Proof. Let $P_e(X) = 2X^e - X^{e-1} - 1$ be the characteristic polynomial of the recursion. Note that $P_e(1) = 0$. Denote by $\alpha_1 = 1, \alpha_2, \ldots, \alpha_e$ the roots of $P_e(X)$. We have the following lemma.

Lemma 2.2. *The α_i are distinct, and for $2 \le i \le e$ we have $|\alpha_i| < 1$.*

Proof. The roots of $P_e'(X)$ are $X = 0$ and $X = (e-1)/(2e)$. Clearly, $P_e(0) \ne 0$ so 0 is not a root of $P_e(X)$, and $P_e((e-1)/(2e)) = -(1/e)((e-1)/(2e))^{e-1} - 1 < 0$, so $(e-1)/(2e)$ is also not a root of $P_e(X)$, hence all the roots of $P_e(X)$ are simple and the α_i are thus distinct. Let us show that $|\alpha_i| < 1$ for $i \ge 2$. Indeed, if $|\alpha_i| > 1$, then $|(2\alpha_i - 1)\alpha_i^{e-1}| > (2|\alpha_i| - 1) > 1$. However, even if $|\alpha_i| = 1$, then $|(2\alpha_i - 1)\alpha_i^{e-1}| = |2\alpha_i - 1|$ and if we set $\alpha_i = e^{i\theta}$ this is equal to $\sqrt{5 - 4\cos(\theta)}$ which can be equal to 1 only if $\theta = 0$, i.e., $\alpha_i = 1$, proving the lemma. □

Resuming the proof of the proposition, we deduce that there exist constants A_i such that $u(n) = A_1 + \sum_{2 \le i \le e} A_i \alpha_i^n$, hence by the lemma $u(n) = A_1 + O(\rho^{-n})$ where ρ can be taken to be equal to the inverse of the largest modulus of the α_i for $i > 1$ (numerically we have $\rho = \sqrt{2} = 1.414\ldots$ for $e = 3$, $\rho = 1.2157\ldots$ for $e = 4$, and $\rho = 1.1296\ldots$ for $e = 5$).

It remains to compute the value of A_1. To do this, we must solve the system of equations $\sum_{1 \le j \le e} A_j \alpha_j^i = u(i)$ for $0 \le i \le e-1$, where even though the value of $u(0)$ may not be defined, we define it so as to satisfy the recursion, in other words we set $u(0) = 2u(e) - u(e-1)$. The matrix of this system is the Vandermonde matrix $M = (m_{i,j})$ with $m_{i,j} = \alpha_j^{i-1}$. The inverse of the Vandermonde matrix is well known (see [Kn]) and is equal to $M^{-1} = (n_{i,j})$ with

$$n_{i,j} = \frac{[(\alpha_1 - X) \cdots \widehat{(\alpha_i - X)} \cdots (\alpha_e - X)]_{j-1}}{(\alpha_1 - \alpha_i) \cdots \widehat{(\alpha_i - \alpha_i)} \cdots (\alpha_e - \alpha_i)},$$

where $\widehat{\ }$ means that the corresponding term is omitted and $[Q(X)]_{j-1}$ is the coefficient of X^{j-1} in the polynomial $Q(X)$.

Thus in particular we have the formula

$$A_1 = \sum_{j=1}^{e} \frac{[(\alpha_2 - X) \cdots (\alpha_e - X)]_{j-1}}{(\alpha_2 - \alpha_1) \cdots (\alpha_e - \alpha_1)} u(j-1).$$

Set $Q_e(X) = P_e(X)/(2(X-1))$. Then $Q_e(X) = \prod_{2 \le j \le e}(X - \alpha_j)$, and since $\alpha_1 = 1$, we have

$$A_1 = \sum_{j=1}^{e} \frac{Q_e(X)_{j-1}}{Q_e(1)} u(j-1).$$

Now we easily check that

$$Q_e(X) = X^{e-1} + \tfrac{1}{2}(X^{e-2} + \cdots + 1)$$

so that $Q_e(X)_{j-1} = \frac{1}{2}$ for $j \le e-1$ and $Q_e(X)_{j-1} = 1$ for $j = e$. In addition, $Q_e(1) = (e+1)/2$. Hence finally we obtain

$$A_1 = \frac{2}{e+1}\left(u(e-1) + \frac{1}{2}\sum_{j=1}^{e-1} u(j-1)\right) = \frac{1}{e+1}\left(u(e-1) + \sum_{j=0}^{e-1} u(j)\right).$$

Replacing the (possibly artificial) $u(0)$ by $2u(e) - u(e-1)$, we obtain the given formula. □

Remark. It is easy to check that the recursion for u implies that the quantity

$$2u(e+k) + \sum_{i=k+1}^{e+k-1} u(i)$$

is independent of k.

2.2. *Analysis of $d(N)$*

Set

$$f_d(n) = \frac{1}{2^{n-2}} \sum_{2^{n-1} \le N < 2^n,\ N \text{ odd}} d(N)$$

and

$$g_d(n) = \frac{1}{2^{n-1}} \sum_{2^{n-1} \le N < 2^n} d(N).$$

These are the averages of $d(N)$ on n-bit odd numbers and all n-bit numbers, respectively. Although we are only interested in the function $g_d(n)$, the auxiliary function $f_d(n)$ will be very useful.

First note the following. If $2^{n-1+e} \le N < 2^{n+e}$ is odd, we can write in a unique way $N = q2^e + r$ with $-2^{e-1} + 1 \le r \le 2^{e-1} - 1$ odd, and we have either $2^{n-1} < q < 2^n$, or $q = 2^n$ and $r < 0$, or $q = 2^{n-1}$ and $r > 0$. In addition, we have $d(N) = d(q) + e$. It follows that

$$\begin{aligned} 2^{n+e-2} f_d(n+e) &= 2^{e-1} \sum_{2^{n-1} < q < 2^n} (e + d(q)) + 2^{e-2}(e + d(2^{n-1})) + 2^{e-2}(e + d(2^n)) \\ &= 2^{e-1} \left(2^{n-1} e + \sum_{2^{n-1} \le q < 2^n} d(q) \right) + 2^{e-2} \\ &= 2^{e+n-2} e + 2^{e-2} + 2^{e+n-2} g_d(n), \end{aligned}$$

so that

$$f_d(n+e) = g_d(n) + e + 2^{-n}.$$

On the other hand, since an integer is either even or odd and $d(2M) = d(M) + 1$, we have

$$\begin{aligned} 2^{n-2} f_d(n) &= \sum_{2^{n-1} \le N < 2^n} d(N) - \sum_{2^{n-2} \le N < 2^{n-1}} (d(N) + 1) \\ &= 2^{n-1} g_d(n) - 2^{n-2} g_d(n-1) - 2^{n-2} \end{aligned}$$

so that

$$f_d(n) = 2g_d(n) - g_d(n-1) - 1.$$

Replacing in the preceding recursion, we obtain the recursion involving only the function $g_d(n)$, which is the one we are interested in:

$$2g_d(n+e) - g_d(n+e-1) - g_d(n) = 1 + e + 2^{-n}.$$

This recurrence relation determines $g_d(n)$ entirely as soon as we know the values of $g_d(1), \ldots, g_d(e)$. If $N \le 2^{e-1} - 1$ is odd, then $d(N) = 0$ hence $f_d(n) = 0$ for $n \le e - 1$. Since clearly $g_d(1) = 0$, the recursion $2g_d(n) - g_d(n-1) - 1 = f_d(n) = 0$ for $2 \le n \le e - 1$ implies that $g_d(n) = 1 - 2^{1-n}$ for $1 \le n \le e - 1$. In addition, if $2^{e-1} < N < 2^e$ is odd, then $N = 1 \cdot 2^e + x_0$ with $-2^{e-1} + 1 \le x_0 \le -1$ is the

representation of Lemma 1.2, so that $d(N) = e$, hence $f_d(e) = e$ and so $2g_d(e) = f_d(e) + g_d(e-1) + 1 = e + 1 + 1 - 2^{2-e}$, hence $g_d(e) = (e+2)/2 - 2^{1-e}$.

We summarize what we have just proved in the following:

Lemma 2.3. *If $g_d(n)$ is the average of the number $d(N)$ of group squarings for n-bit numbers, then $g_d(n)$ satisfies the recursion*

$$2g_d(n+e) - g_d(n+e-1) - g_d(n) = 1 + e + 2^{-n}$$

with the initial conditions $g_d(n) = 1 - 2^{1-n}$ for $1 \le n \le e-1$ and $g_d(e) = (e+2)/2 - 2^{1-e}$.

We now solve this linear recurrence explicitly. If we set $h(n) = g_d(n) - n + 2^{-n}$, then $h(n)$ satisfies the homogeneous recurrence $2h(n+e) - h(n+e-1) - h(n) = 0$ with the initial conditions $h(n) = 1 - n - 2^{-n}$ for $1 \le n \le e-1$ and $h(e) = (2-e)/2 - 2^{-e}$.

It follows from Proposition 2.1 that the solution $h(n)$ to our recurrence relation satisfies $h(n) = A + O(\rho^{-n})$ for some $\rho > 1$, with

$$(e+1)A = 2 - e - 2^{1-e} + \sum_{j=1}^{e-1}(1 - j - 2^{-j}) = \frac{-e(e-1)}{2}$$

so that we have proved:

Proposition 2.4. *Let $g_d(n)$ be the average of the number $d(N)$ of group squarings on n-bit numbers. There exists a real number $\rho > 1$ such that*

$$g_d(n) = n - \frac{e(e-1)}{2(e+1)} + O(\rho^{-n}).$$

2.3. *Analysis of $a(N)$*

The analysis will be very similar to that of $d(N)$, so we can be quite brief. Set

$$f_a(n) = \frac{1}{2^{n-2}} \sum_{2^{n-1} \le N < 2^n,\ N \text{ odd}} a(N)$$

and

$$g_a(n) = \frac{1}{2^{n-1}} \sum_{2^{n-1} \le N < 2^n} a(N).$$

These are the averages of $a(N)$ on n-bit odd numbers and all n-bit numbers, respectively.

Since $a(2M) = a(M)$ we immediately obtain the relation $2^{n-2} f_a(n) = 2^{n-1} g_a(n) - 2^{n-2} g_a(n-1)$ so $f_a(n) = 2g_a(n) - g_a(n-1)$. On the other hand, we have $a(q2^e + r) = a(q) + 1$ and $a(r) = 0$ for $-2^{e-1} + 1 \le r \le 2^{e-1} - 1$, r odd. It follows by writing $N = q2^e + r$ that

$$\begin{aligned} 2^{n+e-2} f_a(n+e) &= 2^{e-1} \sum_{2^{n-1} < q < 2^n} (a(q)+1) + 2^{e-2}(1 + a(2^{n-1})) + 2^{e-2}(1 + a(2^n)) \\ &= 2^{e-1}(2^{n-1} + 2^{n-1} g_a(n)) + 2^{e-2}(a(2^n) - a(2^{n-1})) \\ &= 2^{e+n-2}(1 + g_a(n)), \end{aligned}$$

so that $f_a(n+e) = g_a(n)+1$. Combining with the other formula that we have obtained, this gives the recursion $2g_a(n+e) - g_a(n+e-1) - g_a(n) = 1$.

On the other hand, $g_a(n) = 0$ for $1 \le n \le e-1$, while $f_a(e) = 1$ so $g_a(e) = \frac{1}{2}$. We have thus proved:

Lemma 2.5. *If $g_a(n)$ is the average of the number $a(N)$ of group multiplications for n-bit numbers, then $g_a(n)$ satisfies the recurrence relation $2g_a(n+e) - g_a(n+e-1) - g_a(n) = 1$ with the initial conditions $g_a(n) = 0$ for $1 \le n \le e-1$ and $g_a(e) = \frac{1}{2}$.*

To solve this recursion, we set $h(n) = g_a(n) - n/(e+1)$, so that $h(n)$ satisfies the homogeneous recurrence relation $2h(n+e) - h(n+e-1) - h(n) = 0$ with the initial conditions $h(n) = -n/(e+1)$ for $1 \le n \le e-1$ and $h(e) = \frac{1}{2} - e/(e+1)$. Applying Proposition 2.1, this gives:

Proposition 2.6. *Let $g_a(n)$ be the average of the number $a(N)$ of group multiplications on n-bit numbers. There exists a real number $\rho > 1$ such that*

$$g_a(n) = \frac{n}{e+1} - \frac{(e-1)(e+2)}{2(e+1)^2} + O(\rho^{-n}).$$

Corollary 2.7. *The average cost of computing g^N for an n-bit number N using the addition–subtraction chain coming from Lemma* 1.2 *is approximately equal to*

$$C(N) = \left(n+1-\frac{e(e-1)}{2(e+1)}\right)D + \left(2^{e-2}-1+\frac{n}{e+1}-\frac{(e-1)(e+2)}{2(e+1)^2}\right)A.$$

We will see below a more precise estimate in the case of elliptic curves.

2.4. *Analysis of $x_v(N)$*

Let z be an odd integer such that $1 \le z < 2^{e-1}$. We want to estimate the probability that the leading coefficient x_v is equal to z. Contrary to what intuition could suggest, this probability is not even close to being uniformly distributed among all values of z. In fact, we will find the well-known phenomenon that the initial digit of a number is more often equal to 1 than to any other digit, in an appropriate sense.

Denote by $f_{x,z}(n)$ (resp., $g_{x,z}(n)$) the average of $\delta_z(N)$ on all odd (resp., all) n-bit numbers N, in other words

Proposition 2.8. *Let $f_{x,z}(n)$ be the probability that the leading coefficient $x_v(N)$ be equal to an odd number z for n-bit numbers. There exists a real number $\rho > 1$ such that*

$$f_{x,z}(n) = \begin{cases} \dfrac{3}{e+1} + O(\rho^{-n}) & \text{if } z = 1, \\ \dfrac{1}{2^{m-2}(e+1)} + O(\rho^{-n}) & \text{if } z \text{ has exactly } m \text{ bits with } \quad m \ge 2. \end{cases}$$

In particular, the probability that $x_v(N)$ has one bit is equal to $3/(e+1)+O(\rho^{-n})$ while the probability that $x_v(N)$ has exactly m bits is equal to $1/(e+1)+O(\rho^{-n})$ for $m \geq 2$.

Proof. For simplicity of notation, define $\delta_z(N) = 1$ if $x_v(N) = z$ and $\delta_z(N) = 0$ otherwise. We have therefore

$$f_{x,z}(n) = \frac{1}{2^{n-2}} \sum_{2^{n-1} \leq N < 2^n,\ N \text{ odd}} \delta_z(N) \quad \text{and} \quad g_{x,z}(n) = \frac{1}{2^{n-1}} \sum_{2^{n-1} \leq N < 2^n} \delta_z(N).$$

Since $\delta_z(2k) = \delta_z(k)$ we immediately obtain the relation

$$2^{n-2} f_{x,z}(n) = 2^{n-1} g_{x,z}(n) - 2^{n-2} g_{x,z}(n-1),$$

hence $f_{x,z}(n) = 2g_{x,z}(n) - g_{x,z}(n-1)$. On the other hand for $q > 0$ we have $\delta_z(q2^e + r) = \delta_z(q)$, hence it follows by writing $N = q2^e + r$ with r as above that

$$\begin{aligned} 2^{n+e-2} f_{x,z}(n+e) &= 2^{e-1} \sum_{2^{n-1} < q < 2^n} \delta_z(q) + 2^{e-2} + 2^{e-2} \\ &= 2^{e-1} \sum_{2^{n-1} \leq q < 2^n} \delta_z(q) \\ &= 2^{e+n-2} g_{x,z}(n), \end{aligned}$$

so that $f_{x,z}(n+e) = g_{x+z}(n)$. Combining with the other formula that we have obtained, this gives the recursion $2g_{x,z}(n+e) - g_{x,z}(n+e-1) - g_{x,z}(n) = 0$.

We now consider two cases.

Case 1: $z = 1$. Then we have $\delta_z(r) = 0$ for $3 \leq r < 2^{e-1}$, r odd, $\delta_z(1) = 1$, and $\delta_z(r) = 1$ for $2^{e-1} < r < 2^e$, r odd. Thus, $f_{x,z}(1) = 1$, $f_{x,z}(n) = 0$ for $2 \leq n \leq e-1$, $f_{x,z}(e) = 1$, hence $g_{x,z}(n) = 1/2^{n-1}$ for $1 \leq n \leq e-1$, while $g_{x,z}(e) = 1/2 + 1/2^{e-1}$.

Using Proposition 2.1, we obtain that for $z = 1$ the average of $\delta_z(N)$ on n-bit numbers is equal to $3/(e+1) + O(\rho^{-n})$ for some $\rho > 1$.

Case 2: $z > 1$. Let m be the number of bits of z, so that $2^{m-1} \leq z < 2^m$. We have $2 \leq m \leq e-1$. Then $\delta_z(r) = 0$ for $1 \leq r < 2^e$, r odd, except for $r = z$. Thus, $f_{x,z}(n) = 0$ for $1 \leq n \leq e$, $n \neq m$, and $f_{x,z}(m) = 1/2^{m-2}$, hence $g_{x,z}(n) = 0$ for $n < m-1$, $g_{x,z}(n) = 1/2^{n-1}$ for $m \leq n \leq e$. Using Proposition 2.1, we obtain that for $2^{m-1} \leq z < 2^m$, $2 \leq m \leq e-1$, the average of $\delta_z(N)$ on n-bit numbers is equal to $1/(2^{m-2}(e+1)) + O(\rho^{-n})$ for some $\rho > 1$. □

2.5. *Analysis of $s(N)$*

For a fine analysis of the window method, we must also know the average number of ± 1 among the $x_i(N)$. Thus, we let

$$s(N) = \sum_{0 \leq i \leq v,\ x_i(N) = \pm 1} 1$$

be the number of ± 1 in N.

$$f_s(n) = \frac{1}{2^{n-2}} \sum_{2^{n-1} \le N < 2^n,\ N \text{ odd}} s(N)$$

and

$$g_s(n) = \frac{1}{2^{n-1}} \sum_{2^{n-1} \le N < 2^n} s(N).$$

Since $s(2k) = s(k)$ we immediately obtain the relation $2^{n-2} f_s(n) = 2^{n-1} g_s(n) - 2^{n-2} g_s(n-1)$, so $f_s(n) = 2g_s(n) - g_s(n-1)$.

To simplify notation, set $\delta(r) = 1$ if $r = \pm 1$, $\delta(r) = 0$ otherwise. Then $s(q2^e + r) = s(q) + \delta(r)$ and $s(r) = \delta(r)$ for $1 \le r < 2^{e-1}$, r odd, $s(r) = 1$ for $2^{e-1} < r \le 2^e - 3$, r odd, and $s(2^e - 1) = 2$. It follows by writing $N = q2^e + r$ that

$$\begin{aligned} 2^{n+e-2} f_s(n+e) &= \sum_{-2^{e-1} < r < 2^{e-1},\ \text{odd}} \ \sum_{2^{n-1} < q < 2^n} (s(q) + \delta(r)) \\ &\quad + \sum_{1 \le r < 2^{e-1},\ \text{odd}} (1 + \delta(r)) + \sum_{-2^{e-1} < r \le -1,\ \text{odd}} (1 + \delta(r)) \\ &= \sum_{-2^{e-1} < r < 2^{e-1},\ \text{odd}} \ \sum_{2^{n-1} \le q < 2^n} (s(q) + \delta(r)) \\ &= 2^{e+n-2} g_s(n) + 2^n, \end{aligned}$$

so that $f_s(n+e) = g_s(n) + 1/2^{e-2}$. Combining with the other formula that we have obtained, this gives the recursion $2g_s(n+e) - g_s(n+e-1) - g_s(n) = 1/2^{e-2}$.

On the other hand, $f_s(1) = 2$, $f_s(n) = 0$ for $2 \le n \le e-1$, $f_s(e) = 1 + 1/2^{e-2}$, so $g_s(n) = 1/2^{n-1}$ for $1 \le n \le e-1$ and $g_s(e) = 1/2 + 1/2^{e-2}$. We have thus proved:

Lemma 2.9. *If $g_s(N)$ is the average number of ± 1 for n-bit numbers, then $g_s(N)$ satisfies the recurrence relation $2g_s(n+e) - g_s(n+e-1) - g_s(n) = 1/2^{e-2}$ with the initial conditions $g_s(n) = 1/2^{n-1}$ for $1 \le n \le e-1$ and $g_s(e) = 1/2 + 1/2^{e-2}$.*

To solve this recursion, we set $h(n) = g_s(n) - n/(2^{e-2}(e+1))$, so that $h(n)$ satisfies the homogeneous recurrence relation $2h(n+e) - h(n+e-1) - h(n) = 0$ with the initial conditions $h(n) = 1/2^{n-1} - n/(2^{e-2}(e+1))$ for $1 \le n \le e-1$ and $h(e) = 1/2 + 1/(2^{e-2}(e+1))$. Applying Proposition 2.1, we finally obtain:

Proposition 2.10. *The average number of $x_i(N) = \pm 1$ for n bit numbers is equal to*

$$\frac{n}{2^{e-2}(e+1)} + \frac{3}{e+1} - \frac{(e-1)(e+2)}{2^{e-1}(e+1)^2} + O(\rho^{-n})$$

for some $\rho > 1$.

Since we have proved in the preceding section that the probability that $x_v(N) = 1$ is equal to $3/(e+1)$ ($x_v(N) = -1$ is not possible), we obtain:

Corollary 2.11. *The average number of $x_i(N) = \pm 1$ among the $x_i(N)$ for $0 \le i \le v-1$ for n bit numbers is equal to*

$$\frac{n}{2^{e-2}(e+1)} - \frac{(e-1)(e+2)}{2^{e-1}(e+1)^2} + O(\rho^{-n})$$

for some $\rho > 1$.

Since we may choose two among the 2^{e-1} possible digits, we see that this time intuition does not fool us, since the value given above is (up to $O(\rho^{-n})$) exactly equal to the average number of group multiplications given in Proposition 2.6 divided by 2^{e-2}.

2.6. *The Case of the Addition-Only Window Method*

In this section we give the corresponding results for the addition-only method using the representation of Lemma 1.1 instead of Lemma 1.2. The proofs are essentially identical to the addition/subtraction case so are omitted.

Proposition 2.12. *In the case of the addition-only method, there exists $\rho > 1$ such that*:

(1) *The average number $g_d(n)$ of group squarings is equal to*

$$g_d(n) = n - \frac{e^2+e+2}{2(e+1)} + O(\rho^{-n}).$$

(2) *The average number of group multiplications is equal to*

$$g_a(n) = \frac{n}{e+1} - \frac{e(e+3)}{2(e+1)^2} + O(\rho^{-n}).$$

(3) *The probability $f_{x,z}(n)$ that the leading coefficient $x_v(N)$ is equal to an odd number z for n-bit numbers is equal to*

$$f_{x,z}(n) = \frac{1}{2^{m-2}(e+1)}$$

if z has exactly m bits, with $1 \le m \le e$. In particular, the probability that $x_v(N)$ has one bit is equal to $2/(e+1)$ while the probability that $x_v(N)$ has exactly m bits is equal to $1/(e+1)$ for $2 \le m \le e$.

(4) *The average number of $x_i = 1$ for n bit numbers is equal to*

$$\frac{n}{2^{e-1}(e+1)} + \frac{2}{e+1} - \frac{e(e+3)}{2^e(e+1)^2} + O(\rho^{-n}).$$

3. Application to Elliptic Curve Exponentiation

In [CMO2], we have presented an improved method for elliptic curve exponentiation using the sliding window method together with two new ideas: the use of a new modified

coordinate system $\mathcal{J}^m$, and the use of mixed coordinates for a single computation. In the present section we use the results of the preceding section together with the results of [CMO2] to analyze the method precisely.

Let $\mathbb{F}_p$ be a finite prime field, where p is assumed to be large (for example, at least 96 bits), and let E be an elliptic curve over $\mathbb{F}_p$ whose affine equation is

$$y^2 = x^3 + ax + b.$$

Apart from the choice of affine coordinates, denoted $\mathcal{A}$, we also use the Jacobian coordinates (x, y, z) corresponding to the affine point $(x/z^2, y/z^3)$ when $z \neq 0$, denoted $\mathcal{J}$, and the modified Jacobian coordinates (x, y, z, az^4) with the same meaning but with an extra component, denoted $\mathcal{J}^m$ (in [CMO2] we also considered ordinary projective coordinates and Chudnovsky Jacobian coordinates, but they are not useful in practice).

3.1. *The Ordinary Horner Scheme*

The strategy given in [CMO2] is the following, assuming reasonable practical hypotheses about the respective times for inversion I, multiplication M, and squaring S in the base field $\mathbb{F}_p$.

Let P be a point on E given in affine coordinates. To simplify the analysis, we assume that we are in a generic situation so that no point encountered during the computation is the point at infinity. This is of course highly likely since the prime p has been assumed large.

To compute $N \cdot P$ for some large integer N using the base 2^e sliding window representation, we first precompute $2P$ in affine coordinates, then the points $x \cdot P$ in affine coordinates for $3 \leq x \leq 2^{e-1} - 1$, x odd by $2^{e-2} - 1$ additions in affine coordinates. Since, with evident notations, we have $t(\mathcal{A} \leftarrow \mathcal{A} + \mathcal{A}) = I + 2M + S$ and $t(\mathcal{A} \leftarrow 2\mathcal{A}) = I + 2M + 2S$, the precomputations require a time T_1 given by

$$T_1 = S + 2^{e-2}(I + 2M + S).$$

For the main computation, we use Horner's scheme on the representation

$$N = ((((x_v 2^{k_v} + x_{v-1})2^{k_{v-1}} + x_{v-2})2^{k_{v-2}} + \cdots + x_1)2^{k_1} + x_0)2^{k_0}$$

given by Lemma 1.2. Thus, if we set

$$N_j = ((x_v 2^{k_v} + x_{v-1})2^{k_{v-1}} + \cdots + x_{j+1})2^{k_{j+1}} + x_j,$$

and $P_j = N_j \cdot P$, we initialize $P_v \leftarrow x_v \cdot P$ and we set for $j = v - 1, \ldots, j = 0$,

$$P_j \leftarrow 2^{k_{j+1}} \cdot P_{j+1} + x_j \cdot P$$

and the final result is

$$N \cdot P = 2^{k_0} \cdot P_0.$$

The computation of $P_j \leftarrow 2^{k_{j+1}} \cdot P_{j+1} + x_j \cdot P$ is performed as follows. By induction P_{j+1} will be in the modified Jacobian coordinates $\mathcal{J}^m$. The $k_{j+1} - 1$ doublings will be done using these coordinates, while the last such doubling will use these coordinates

but give the result in ordinary Jacobian coordinates. The following times are taken from [CMO2]:

$$\begin{aligned} t(\mathcal{J}^m \leftarrow 2\mathcal{J}^m) &= 4M+4S,\\ t(\mathcal{J} \leftarrow 2\mathcal{J}^m) &= 3M+4S,\\ t(\mathcal{J}^m \leftarrow 2\mathcal{A}) &= 3M+4S,\\ t(\mathcal{A} \leftarrow 2\mathcal{J}^m) &= I+6M+5S,\\ t(\mathcal{J}^m \leftarrow \mathcal{J}+\mathcal{A}) &= 9M+5S,\\ t(\mathcal{J}^m \leftarrow \mathcal{A}+\mathcal{A}) &= 5M+4S,\\ t(\mathcal{A} \leftarrow \mathcal{J}+\mathcal{A}) &= I+11M+4S. \end{aligned}$$

Thus the computation of $P_j \leftarrow 2^{k_{j+1}} \cdot P_{j+1} + x_j \cdot P$ for $1 \le j \le v-2$ requires a time $t_{2,j}$ given by

$$t_{2,j} = k_{j+1}(4M+4S) + 8M + 5S.$$

For $j = 0$, the time is given by the same formula of $k_0 > 0$, but we can subtract $M+2S$ if $k_0 = 0$ since we need the result only in Jacobian, but not modified Jacobian, coordinates.

For $j = v-1, 2P$ has already been computed, in affine coordinates, so the computation time is

$$t_{2,v-1} = k_v(4M+4S) + 3M + S.$$

Finally, assuming we want the result in affine coordinates, the final k_0 doublings require the time for conversion from Jacobian to affine if $k_0 = 0$, that is, $I + 3M + S$, and otherwise time

$$t_{2,-1} = k_0(4M+4S) + I + 2M + S.$$

Thus, the time for Horner's scheme is

$$T_2 = \left(\sum_{0 \le j \le v} k_j\right)(4M+4S) + I + v(8M+5S) - 3M - (3+2\delta(k_0)S),$$

where $\delta(x)$ is the Kronecker symbol equal to 0 except when $x = 0$ for which it is equal to 1.

Using the notation used in the preceding section, this can be written

$$T_2(N) = d(N)(4M+4S) + I + a(N)(8M+5S) - 3M - (3+2\delta(k_0)S).$$

Since the probability for $k_0 = 0$ is equal to $\frac{1}{2}$ (no need for a detailed analysis for that), we deduce from Section 2 that the average of $T_2(N)$ over integers N having exactly n bits is approximately equal to

$$\left(n - \frac{e(e-1)}{2(e+1)}\right)(4M+4S) + \left(\frac{n}{e+1} - \frac{(e-1)(e+2)}{2(e+1)^2}\right)(8M+5S) + I - 3M - 4S.$$

3.2. *Horner's Scheme with Initialization*

As already mentioned, it is possible to improve slightly on the first step of Horner's scheme. For example, if $x_v = 1$ and $k_v = e$, instead of computing $2^{k_v} \cdot P$ which requires k_v doublings, we can simply compute $2 \cdot (P + (2^{e-1} - 1)P)$ which requires only one doubling and one addition, since $(2^{e-1} - 1)P$ has been precomputed. This is especially attractive since by the results of Section 2 we know that x_v will be equal to 1 with high probability.

We could very precisely write the optimal addition chain to choose for the initialization step. Since the gain would be minimal (less than M), we will be content with the following approximation to optimality.

If $2^{m-1} \le x_v < 2^m$, in other words if x_v has exactly m bits, with $1 \le m \le e-1$, we compute $P_{v-1} \leftarrow x_v 2^{k_v} \cdot P$ in ordinary Jacobian coordinates by using the identity

$$x_v 2^{k_v} = 2^{k_v-(e-m)}((2^{e-1} - 1) + (x_v 2^{e-m} + 1 - 2^{e-1})).$$

Since $2^{m-1} \le x_v \le 2^m - 1$ and $m \le e - 1$, we have

$$1 \le x_v 2^{e-m} + 1 - 2^{e-1} \le 2^{e-1} - 2^{e-m} + 1 \le 2^{e-1} - 1,$$

so the computation of $x_v 2^{k_v}$ using this method requires only $k_v - (e-m)$ doublings and one addition, giving a gain of $e - m$ doublings minus one addition.

More precisely, the usual method for computing $x_v 2^{k_v} \cdot P$ giving the result in ordinary Jacobian coordinates requires time $(k_v-1)(4M+4S)-2M$ since $2P$ has been computed.

The above method requires time $(k_v - e + m + 1)(4M + 4S)$. The improvement is thus equal to $(e - m - 2)(4M + 4S) - 2M$, so should be used when $m \le e - 3$. In particular, there is no possible gain when $e \le 3$.

It is easy to compute the average gain when $e \ge 4$: by Proposition 2.8, we have $x_v = 1$ with probability $3/(e+1)$ and x_v has m bits with probability $1/(e+1)$ for $2 \le m \le e-1$. Since we use this method only for $m \le e-3$, a small computation shows that the average gain is equal to

$$\frac{2}{e+1}((e^2 - 2e - 5)M + (e^2 - e - 6)S).$$

3.3. *Using Montgomery's Trick*

There is still another trick which can be used to speed up the computation. This is the use of a variant of Montgomery's method for computing simultaneous inverses, applied to the precomputations which are necessary in the window method for elliptic curves. Recall that if k inversions modulo p are to be performed simultaneously, using Montgomery's method we may compute a single inverse modulo p at the cost of $3(k-1)$ multiplications (see, for example, Algorithm 10.3.4 of [C]).

As explained above, we first compute $2P$ in affine coordinates. However, now we compute *simultaneously* $3P = 2P + P$ and $4P = 2P + 2P$ (still in affine coordinates) using Montgomery's idea, then simultaneously $5P = 4P + P$, $7P = 4P + 3P$, $8P = 4P + 4P$, and so on until $(2^{e-2} + 1)P = 2^{e-2}P + P, \ldots, (2^{e-1} - 1)P = 2^{e-2}P + (2^{e-2} - 1)P$ and we do not compute $2^{e-1}P$. Although we compute the apparently useless

quantities $4P, 8P, \ldots, 2^{e-2}P$, a short calculation shows that the time for computing $x \cdot P$ for x odd, $3 \leq x \leq 2^{e-1} - 1$, using this method is equal to

$$(e-1)I + (5 \cdot 2^{e-2} + 2e - 12)M + (2^{e-2} + 2e - 5)S$$

when $e \geq 3$. For example, with $e = 5$ this gives $4I + 38M + 13S$, compared with $T_1 = 8I + 16M + 9S$ using the ordinary method, and this is almost always faster since I is much larger than M.

References

[C] H. Cohen, *A Course in Computational Algebraic Number Theory*, Graduate Texts in Mathematics, vol. 138, Springer-Verlag, Berlin, 1993.

[CMO1] H. Cohen, A. Miyaji, and T. Ono, Efficient elliptic curve exponentiation, *Proceedings ICICS* '97, Lecture Notes in Computer Science, vol. 1334, Springer-Verlag, Berlin, 1997, pp. 282–290.

[CMO2] H. Cohen, A. Miyaji, and T. Ono, Efficient elliptic curve exponentiation using mixed coordinates, *Proceedings ASIACRYPT* '98, Lecture Notes in Computer Science, vol. 1514, Springer-Verlag, Berlin, 1998, pp. 51–65.

[Kn] D. Knuth, *The Art of Computer Programming*, vol. II, third edition, Addison-Wesley, Reading, MA, 1997.

[Ko] C. Koc, Analysis of sliding window techniques for exponentiation.

[MO] F. Morain and J. Olivos, Speeding up the computations on an elliptic curve using addition–subtraction chains, 1990.

[P] N. Pippenger, On the evaluation of powers and monomials, *SIAM J. Comput.*, **9** (1980), 230–250.

[Y] A. Yao, On the evaluation of powers, *SIAM J. Comput.*, **5** (1976), 100–103.