On the complexity of certain multi-exponentiation techniques in cryptography

(To appear in: Journal of Cryptology)

Roberto M. Avanzi*
Institut für Experimentelle Mathematik
Ellernstraße 29, D-45326 Essen
Email: mocenigo@exp-math.uni-essen.de

December 13, 2003

Abstract

We describe, analyze and compare some combinations of multi-exponentiation algorithms with representations of the exponents. We are especially interested in the case where the inversion of group elements is fast: This is true for example for elliptic curves, groups of rational divisor classes of hyperelliptic curves, trace zero varieties and XTR. The methods can also be used for computing *single* exponentiations in groups which admit an appropriate automorphism satisfying a monic equation of small degree over the integers.

Keywords and phrases: Exponentiation, Scalar multiplication, Integer recoding, Elliptic and Hyperelliptic curves, Trace zero varieties, XTR, Groups with automorphisms.

1 Introduction

Some public-key cryptographic protocols such as the verification of digital signatures require the computation of the product of powers of two [1, 6, 28, 24] or three [11] elements of a group. Furthermore, if a commutative group G admits an appropriate automorphism σ satisfying a monic equation over the integers of degree d then the *single exponentiation* g^e can be computed as $g^{e_0} \cdot \sigma(g)^{e_1} \cdots \sigma^{d-1}(g)^{e_{d-1}}$ for suitable integers $e_0, \ldots, e_{d-1}$ which in many practical instances have size $O(e^{1/d})$ (see [14, 29]). In this context, too, the cases $d = 2$ and $d = 3$ are of particular practical relevance because of trace zero varieties and XTR (see Subsection 4.2). Such computations can be performed by computing the various powers separately and then multiplying them together: In some special contexts this approach can lead to very good performance [9].

An idea due to Straus [35] and since ElGamal's paper [11] erroneously attributed to Shamir and called *Shamir's trick*, works by reading the exponents simultaneously: it saves several squarings and also many multiplications if some products of the bases are precomputed. Straus' idea can be extended in a straightforward way by using sliding windows: to our knowledge this was first reported in [38] and applied there only to the binary representation

*The work described in this paper has been supported by the Commission of the European Communities through the Fifth Framework Programme under Contract IST-2001-32613 (see http://www.arehcc.com).

of the exponents. Möller has compared this basic form to other methods such as *interleaved exponentiation* [20]. We combine the idea of [38] with different exponent recodings and thus complement Möller's analysis. We use signed digit representations – first introduced in [5] – for the exponents, in particular the *non-adjacent form* [27, 22] (see Theorem 3.6 and Remark 3.7) and a new representation of pairs of integers due to Solinas [33] (see Theorem 3.14), called the *joint sparse form* (JSF). The combination with the NAF, in the case of two exponents only, has been independently proposed in [25], where a correct performance analysis is missing.

In the next section we will introduce the general form of the algorithm from [38], which will be analyses in detail in Section 3: this forms the main part of this paper. In Section 4 the optimal choice of parameters will be discussed, and some applications will be outlined.

2 The algorithm

We now review the description of the algorithm from [38]. Let G be a commutative group of order $q \approx 2^n$ and d a (small) integer. Suppose we are given elements $g_1, \ldots, g_d \in G$ and integers $e_1, \ldots, e_d$ and want to compute $x := \prod_{i=1}^{d} g_i^{e_i}$. Write

$$e_i = \sum_{j=0}^{n-1} e_{i,j}\, 2^j \tag{1}$$

with $e_{i,j} \in \{0, \pm 1\}$. In an unsigned expansion the value -1 is not allowed. In this paper the commonly accepted notation $\bar{1}$ for -1 is used.

For the moment we assume that the chosen representation is the unsigned binary one.

The most obvious way of performing the desired computation, as already mentioned, consists in computing the powers separately and multiplying them together. The second most natural way is perhaps the following one, which saves some squarings.

Algorithm 2.1 Simple multi-exponentiation

INPUT: Group elements $g_1, \ldots, g_d$ and exponents $e_1, \ldots, e_d$ written as in (1) in base 2 (*i.e.* with $e_{i,j} \in \{0, 1\}$)
OUTPUT: $\prod_{i=1}^{d} g_i^{e_i}$

1. $x \leftarrow 1 \in G$

2. **for** $j = n-1 \ldots 0$ **do** **{**
  $x \leftarrow x^2$ [Skip at first iteration]
  for $i = 1 \ldots d$ **do** **{ if** $e_{i,j} = 1$ **then** $x \leftarrow x \cdot g_i$ **} }**

3. `return` x

Using Straus' trick [35], Algorithm 2.1 is enhanced as follows (note that Straus' trick is more general, since it is 2^w-ary and not simply binary). First, precompute the 2^d values $\prod_{i \in \mathcal{S}} g_i$ for all subsets $\mathcal{S} \subseteq \{1, 2, \ldots, d\}$. Then, put $x = \prod_{i=1}^{d} g_i^{e_{i,n-1}}$ by one table look-up. Then, for $j = n-2, \ldots, 1, 0$, replace x by $x^2 \cdot \prod_{i=1}^{d} g_i^{e_{i,j}}$ by one squaring, one table look-up and one multiplication. This method requires $2^d - d - 1$ multiplications to prepare the table,

$n-1$ squarings and on average $(1-2^{-d})n$ multiplications, 2^{-d} being the probability that for a fixed j, $e_{i,j}$ is 0 for all $i = 1, 2, \ldots, d$. If the exponents are written in a signed binary representation, the table $\mathcal{E}$ can be formed from the products $\prod_{i=1}^{d} g_i^{k_i}$ with $k_i \in \{0, \pm 1\}$. If the cost of an inversion in the group G is negligible, which is usually the main reason for adopting a signed binary representation, one only needs a half of those values, *i.e.* those where the first nonzero k_i equals 1. Then some multiplications are replaced by divisions. This method can be improved by means of a sliding window in the same way as the square-and-multiply method [36, 16]:

Algorithm 2.2 Multi-exponentiation with a sliding window

INPUT: A window size w, integers $e_1, \ldots, e_d$ as in (1) and a set $\mathcal{E}$ of elements of the group G of the form $\prod_{i=1}^{d} g_i^{k_i}$ including $g_1, \ldots, g_d$ *(the set $\mathcal{E}$ depends on w and on the chosen representation of the integers e_i: see Remarks 2.3 (4) and Subsections 3.2 and 3.3.)*
OUTPUT: $\prod_{i=1}^{d} g_i^{e_i}$

1. $t \leftarrow n$, $W \leftarrow w$ and $x \leftarrow 1 \in G$

2. **if** ($e_{i,t-1} = 0$ for $i = 1, 2, \ldots, d$) **then {**

(a) $t \leftarrow t-1$ and $x \leftarrow x^2$

} else {

(b) **if** $t \geq W$ **then** $t \leftarrow t - W$ **else {** $W \leftarrow t$ and $t \leftarrow 0$ **}**

(c) **for** $i = 1, 2, \ldots, d$ **do** $f_i \leftarrow \sum_{j=0}^{W-1} e_{i,t+j} 2^j$

(d) Let s be the largest integer $s \geq 0$ such that $2^s | f_i$ for all i

(e) **for** $i = 1, 2, \ldots, d$ **do** $f_i \leftarrow f_i / 2^s$

(f) (i) $x \leftarrow x^{2^{W-s}}$; (ii) $x \leftarrow x \cdot \prod_{i=1}^{d} g_i^{f_i}$ and (iii) $x \leftarrow x^{2^s}$ **}**

3. **if** $t = 0$ **then** `return` x **else** `goto` **Step 2**

Remarks 2.3 (1) *The variable W is initially set to w. If, at the last iteration, the amount t of bits still to be parsed is smaller than w, then W will be set to t to avoid reading past the end of the e_i.*

(2) *At the beginning of Step 2 (c) f_i is the integer represented by a string of W consecutive signed bits from the exponent e_i. Now s is the largest non-negative integer such that $e_{i,t+u} = 0$ for all i and all u with $0 \leq u \leq s$. The* normalization *Step 2 (e) is performed such that at least one of the integers f_i is odd, in order to reduce the number of elements of $\mathcal{E}$ without impacting the total number of operations done in Step 2 (f).*

(3) *In Step 2 (f) the first time it is $x = 1$, so a multiplication can be saved and only s squarings are needed.*

(4) *If the exponents are written in base 2, using only the unsigned bits 0 and 1, then $\mathcal{E}$ consists of all elements of the form $\prod_{i=1}^{d} g_i^{k_i}$ such that $0 \leq k_i < 2^w$ and at least one of the k_i is odd. Then Step 2 (f) is done with one table look-up, one multiplication and W squarings, where W is always equal to w, except possibly for the last iteration. In Subsections 3.2 and 3.3 a detailed analysis is given of Algorithm 2.2 coupled with the NAF and the JSF respectively.*

3 Complexity analysis

In this section we are concerned only with Algorithm 2.2 and its complexity.

Definition 3.1 *A* column *is a d-tuple of digits $e^{(t)} = (e_{1,t}, \ldots, e_{d,t})$ of the representation of integers given in* (1)*. The* joint representation *of the d exponents $e_1, \ldots, e_d$. is the ordered sequence $e^{(n-1)}, e^{(n-2)}, \ldots, e^{(0)}$.*

If $e^{(n-1)} \neq 0$ then the joint representation is said to be proper *and n is its* length*.*

The number of nonzero columns in the joint representation is called its Hamming weight*, and its* density *is the ratio of the Hamming weight to the length.*

For simplicity we require that the joint representation of the exponents $e_1, \ldots, e_d$ is proper. Thus at the first iteration of Step (2), substeps (b)–(f) are always performed. To evaluate the number of squarings one should not consider those which can be avoided in the first iteration, which are w minus the expected first value of s.

Algorithm 2.2 scans the joint representation of the d exponents $e_1, \ldots, e_d$ one column at a time, starting with the column formed by the most significant digits in the chosen representation. Step 2 is iterated until the joint representation has been read completely. At every iteration, first one column is read, determining which of two possible distinct states the algorithm assumes:

$\mathcal{S}_0$. A zero column is found, so the scanning advances by one column (Step 2 (a)).

$\mathcal{S}_1$. A nonzero column is found and the scanning advances by w columns (Steps 2 (b)–(f)).

The amount of multiplications performed in Step 2 (f,ii) during a multi-exponentiation equals the number of times the algorithms enters the second state. Let π be the probability that the column read in Step 2 is zero. After m iterations, the expected number of columns read by the scanning process is $(\pi + w(1-\pi))m$. Suppose that for some m this number is n. The number of multiplications performed by Algorithm 2.2 in Step 2 (d) is then $(1-\pi)m - 1$ (remember that the first multiplication can be replaced by an assignment) *i.e.*

$$n \cdot \frac{1-\pi}{\pi + w(1-\pi)} - 1 \ . \tag{2}$$

This is, with some adaptations, the approach followed in the next two subsections.

Definition 3.2 *Let $e = \sum_{j=0}^{n-1} e_j 2^j$ be an integer. We say that an algorithm scans (generates, rewrites...) the* bits e_j right-to-left *(resp.* left-to-right*) if it scans (generates, rewrites...) them from the least significant ones to the most significant ones, i.e. first e_0, then e_1, e_2, etc. (resp. from the most significant ones to the least significant ones, i.e. first e_{n-1}, then e_{n-2}, and so on).*

Similar definitions hold for algorithms which deal with the columns of a joint representation of several integers.

Remark 3.3 *Algorithm 2.2 processes the columns of the chosen joint representation of the exponents left-to-right. However, most methods for producing signed binary representations rewrite the exponents right-to-left, including Reitwiesner's algorithm [27] and Solinas' algorithm for the JSF. In such situations recoding and (multi-)exponentiation cannot be interleaved, and the recoded representations must be stored explicitly.*

3.1 Unsigned binary inputs

Here the exponents are written in base 2, *i.e.* $e_{i,j} \in \{0,1\}$. The set $\mathcal{E}$ consists of all elements of the form $\prod_{i=1}^{d} g_i^{k_i}$ such that $0 \le k_i < 2^w$ and at least one of the k_i is odd. It has cardinality $2^{wd} - 2^{(w-1)d}$.

The bits in each representation are assumed to be zero or one with equal probability and independent from the adjacent bits, so $\pi = 2^{-d}$. To evaluate the number of squarings in the main loop of the algorithm we must determine the expected value of s at the first iteration. As all the bits are independent from each other, $s \ge u$ with $1 \le u < w$ with probability 2^{-ud}. Hence the expected value of s is $\sum_{u=1}^{w-1} 2^{-ud} = \frac{1-2^{-d(w-1)}}{2^d-1}$. We have thus the following result:

Theorem 3.4 *Suppose that in Algorithm 2.2 the unsigned binary representation is used for the exponents, and that their joint representation has length n. Then the set $\mathcal{E}$ has cardinality $2^{wd} - 2^{(w-1)d}$ and its computation requires $2^{wd} - 2^{(w-1)d} - d$ multiplications and d squarings. The expected number of multiplications in the algorithm's main loop is $n\frac{1}{w+(2^d-1)^{-1}} - 1$ and that of the squarings is $n - w + \frac{1-2^{-d(w-1)}}{2^d-1}$.*

Remark 3.5 *In the case $w = d = 2$, the set $\mathcal{E}$ consists of the values $g_1^a g_2^b$ with $0 \le a, b \le 3$ and at least one of a, b odd. To determine them one has to compute and store g_1^2 and g_1^3, as well as g_2^2 and g_2^3. This requires 2 squarings and 2 multiplications. Computing the remaining 8 values requires 8 further multiplications.*

3.2 Using the NAF

A *non-adjacent form* (abbreviated as NAF) is the signed binary representation of an integer $e = \sum_{j=0}^{n-1} b_j 2^j$ with $b_j \in \{0, \pm 1\}$ and $b_j b_{j-1} = 0$. Each integer admits a NAF, which is uniquely determined. It is a signed binary representation of minimal Hamming weight and it has expected density 1/3 (see [22] and [2] for proofs).

Theorem 3.6 *Suppose that in Algorithm 2.2 the exponents are input in NAF, and that their joint representation is n bits long.*

The set $\mathcal{E}$ has cardinality $(I_w^d - I_{w-1}^d)/2$ where $I_w = \frac{2^{w+2}-(-1)^w}{3}$.

The expected number of squarings in the main loop of the algorithm is $n - w + \langle s \rangle$ where $\langle s \rangle = \left(\frac{2}{3}\right)^d \sum_{t=1}^{w-1} \left(\frac{1}{2}\right)^{(w-1-t)d} \left(1 - (-\frac{1}{2})^t\right)\left(1 - (-\frac{1}{2})^{t+2}\right)^{d-1} + O(2^{-n})$. The value of $\langle s \rangle$ approaches $\left(\frac{2}{3}\right)^d \frac{1}{1-2^{-d}}$ for large values of w.

In the cases $d = 1, 2$ and 3 respectively, the expected number of multiplications is $n \cdot \frac{1-\pi^{(d)}}{w-(w-1)\pi^{(d)}} - 1$ where

$$\pi^{(1)} = \frac{4\left(2^w - (-1)^w\right)}{7 \cdot 2^w - 4 \cdot (-1)^w}, \quad \pi^{(2)} = \frac{16\left(4^w - 1\right)}{43 \cdot 4^w + 24 \cdot (-2)^w - 16} \quad \text{and}$$
$$\pi^{(3)} = \frac{64\left(2^w + (-1)^w\right)\left(8^w - (-1)^w\right)}{253 \cdot 16^w + 397 \cdot (-8)^w + 324 \cdot 4^w + 80 \cdot (-2)^w - 64}. \tag{3}$$

In particular, for $d = 1$ this equals $n \cdot \frac{1}{w+\frac{4}{3}\left(1-(-\frac{1}{2})^w\right)} - 1$.

Remark 3.7 *In the case $w = d = 2$, the set $\mathcal{E}$ consists of the values $g_1^a g_2^b$ with either $0 < a \leq 2$ and $-2 \leq b \leq 2$ where at least one of a, b odd, or $a = 0$ and $b = 1$. A chain for computing $\mathcal{E}$ by 6 multiplications or multiplications with the inverse is*

$$\left\{ g_1, \quad g_2, \quad g_1 g_2, \quad g_1 g_2^{-1}, \quad g_1 g_2^2, \quad g_1 g_2^{-2}, \quad g_1^2 g_2, \quad g_1^2 g_2^{-1} \right\}.$$

For the special case $d = 1$, a related analysis for fixed windows (rather than sliding windows as in Algorithm 2.2) appears in [12].

The remainder of this subsection is devoted to the proof of Theorem 3.6.

First note that the largest integer representable by a w-bit number in NAF is $(10\ldots01)_2$ for odd w and $(10\ldots10)_2$ for even w: it is easy to see that this number is $T_w = (2^{w+2} - 3 - (-1)^w)/6$. Hence, there are $I_w = (2^{w+2} - (-1)^w)/3$ integers in the interval $[-T_w, \ldots, T_w]$. Now $\mathcal{E}$ consists of *all elements of the form $\prod_{i=1}^d g_i^{k_i}$ such that $|k_i| \leq T_w$ for $i = 1, 2, \ldots, d$, at least one of the k_i is odd and the first nonzero element in the sequence $k_1, k_2, \ldots, k_p$ is positive.* In this way, if in Step 2 (f,ii) the first nonzero f_i is positive we compute $x \leftarrow x \cdot \prod_{i=1}^d g_i^{f_i}$ otherwise we compute $x \leftarrow x / \prod_{i=1}^d g_i^{-f_i}$. Hence we need only $(I_w^d - I_{w-1}^d)/2$ elements in $\mathcal{E}$.

Definition 3.8 *A joint representation of integers in NAF will be called a* joint NAF.

We will model the left-to-right scanning of the joint NAF performed by Algorithm 2.2 as a Markov chain. In each iteration one column is read and the algorithm assumes one of $d+1$ possible distinct states, defined by the number of nonzero entries in the columns:

$\mathcal{S}'_0$. A zero column is found, so the scanning advances by one column.

$\mathcal{S}'_k$ (for $1 \leq k \leq d$). A column is found with exactly k nonzero entries and the scanning advances by w columns.

To determine the transition probability from state $\mathcal{S}'_\ell$ to state $\mathcal{S}'_k$ we need a few preliminary results.

We begin reviewing the probably most common method to compute the NAF $\sum_{j=0}^n \nu_j 2^j$ of a given integer e with $|e| < 2^n$.

At the beginning we put $j = 0$. We repeat the following two steps until $e = 0$:

(i) If e is even, then halve e, put $\nu_j = 0$ and increment j by one;

(ii) If e is odd, then we can always choose $\nu_j \in \{+1, -1\}$ such that $e \equiv \nu_j \bmod 4$. Then we subtract ν_j from e: at this point e is divisible at least by 4 and so (i) is repeated at least twice, hence the output is guaranteed to satisfy the non-adjacency property.

In practice, the variable e is not modified and we work with a carry bit.

Since a random integer is even with probability $\frac{1}{2}$, and congruent to 1 or -1 modulo 4 with probability in each case $\frac{1}{4}$, we have the following result.

Lemma 3.9 *The probability that in a NAF the digit immediately to the left of a 0 is another 0 is $\frac{1}{2}$ and that it is 1 or -1 is in each case $\frac{1}{4}$.*

For our analysis we need however the probabilities that the digits to the *right* of a given one are zeros or nonzeros. By the Lemma just proved, an (infinite) random NAF can be obtained essentially by outputting digit strings of length 1 or 2:

$$\begin{array}{ll} (0) & \text{with probability } 1/2\,, \\ (1\,0) & \text{with probability } 1/4\,, \text{ and} \\ (\bar{1}\,0) & \text{with probability } 1/4\,. \end{array}$$

But we can regroup in a different way the digits of the output of the above process. This is the same as outputting the "reversed" strings

$$\begin{array}{ll} (0) & \text{with probability } 1/2\,, \\ (0\,1) & \text{with probability } 1/4\,, \text{ and} \\ (0\,\bar{1}) & \text{with probability } 1/4\,. \end{array}$$

This equivalence shows that random NAFs look essentially the same read in both directions, so an analogue of Lemma 3.9 holds with right in lieu of left.

Lemma 3.10 *The probability that in a NAF the digit immediately to the right of a* 0 *is another* 0 *is* $\frac{1}{2}$ *and that it is* 1 *or* -1 *is in each case* $\frac{1}{4}$.

We now generalize this last result by determining the probabilities that a bit $e_{j,i-w}$ (resp. $e_{j,i+w}$) which is w places to the right (resp. left) of $e_{j,i}$ is zero or one, depending on the value of $e_{j,i}$ and w.

Lemma 3.11 *If* $e_{j,i} = 0$, *then* $e_{j,i-w} = 0$ *with probability* $\pi_{w,0}$ *and* $e_{j,i-w} \neq 0$ *with probability* $\pi_{w,*}$, *where*

$$\begin{aligned} \pi_{w,0} &= \frac{2}{3} + \frac{1}{3}\left(-\frac{1}{2}\right)^w \quad \textit{and} \\ \pi_{w,*} &= 1 - \pi_{w,0} = \frac{1}{2}\pi_{w-1,0} = \frac{1}{3} - \frac{1}{3}\left(-\frac{1}{2}\right)^w \ . \end{aligned} \tag{4}$$

Since a nonzero bit is always followed by a zero, we also have that if $e_{j,i} \neq 0$, *then* $e_{j,i-w} = 0$ *with probability* $\pi_{w-1,0}$ *and* $e_{j,i-w} \neq 0$ *with probability* $\pi_{w-1,*}$.

The same probabilities hold with $i - w$ *replaced by* $i + w$.

Proof. Clearly $\pi_{w,0} + \pi_{w,*} = 1$. By Lemma 3.10 we have $\pi_{1,0} = \pi_{1,*} = \frac{1}{2}$ and

$$\begin{cases} \pi_{i-1,0} = \pi_{i,*} + \dfrac{1}{2}\pi_{i,0} = 1 - \dfrac{1}{2}\pi_{i,0} \\ \pi_{i-1,*} = \dfrac{1}{2}\pi_{i,0} \ . \end{cases}$$

Now (4) follows easily by induction. The last claim follows by using Lemma 3.9 instead of Lemma 3.10. □

We are now in the position to model the left-to-right scanning process as a Markov chain with states $\mathcal{S}'_0, \ldots, \mathcal{S}'_d$. Denote by $\tau_{\ell,k}$ the transition probability from state $\mathcal{S}'_\ell$ to state $\mathcal{S}'_k$.

Suppose that a zero column is read. Then no window is being formed and at the next iteration the scanning algorithm will read the next column to the right. The probability $\tau_{0,k}$ that this column contains exactly k nonzero entries is $\binom{d}{k}\frac{1}{2^k}$.

On the other hand suppose that a column $\mathbf{c}$ with exactly $\ell \neq 0$ nonzero entries has been read. The numbers represented by this column and the next $w-1$ columns at its right are the exponents $f_1, \ldots, f_d$ in Step 2 (c). The next column checked by the left-to-right scanning process, say $\mathbf{c}'$, will then be the one exactly w places to the right of $\mathbf{c}$. Now $\tau_{\ell,k}$ is the probability that $\mathbf{c}'$ has exactly k nonzero entries (where $0 \leq k \leq d$). For some integer r, in exactly r of the positions occupied by the ℓ nonzero digits in $\mathbf{c}$ there will be nonzero bits in the respective positions in $\mathbf{c}'$, and in the positions of the remaining $\ell - r$ nonzero bits in $\mathbf{c}$ there will be zeros in $\mathbf{c}'$. Therefore, to exactly $k-r$ of the zero bits in $\mathbf{c}$ will correspond nonzero bits in $\mathbf{c}'$, and to the other $d-\ell-(k-r)$ zeros of $\mathbf{c}$ will correspond zeros in $\mathbf{c}'$. Thus, we have

$$\begin{aligned}\tau_{\ell,k} &= \sum_{\substack{r\,:\,0\leq r\leq \ell \\ 0\leq k-r\leq d-\ell}} \binom{\ell}{r}\binom{d-\ell}{k-r} \pi_{w-1,*}^{r}\pi_{w-1,0}^{\ell-r}\pi_{w,*}^{k-r}\pi_{w,0}^{d-\ell-(k-r)} \\ &= \sum_{r=\max\{0,k+\ell-d\}}^{\min\{\ell,k\}} \binom{\ell}{r}\binom{d-\ell}{k-r} 2^{\ell-r}\pi_{w,*}^{\ell+k-2\,r}\times \\ &\qquad\qquad \times \left(1-\pi_{w,*}\right)^{(d-\ell-k)+r}\left(1-2\pi_{w,*}\right)^{r} \ .\end{aligned}$$

Put

$$T_d = (\tau_{\ell,k})_{\ell,k=0}^{d} = \begin{pmatrix} 1/2^d & \tau_{1,0} & \tau_{2,0} & \cdots & \tau_{d,0} \\ \binom{d}{1}/2^d & \tau_{1,1} & \tau_{2,1} & \cdots & \tau_{d,1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \binom{d}{d-1}/2^d & \tau_{1,d-1} & \tau_{2,d-1} & \cdots & \tau_{d,d-1} \\ 1/2^d & \tau_{1,d} & \tau_{2,d} & \cdots & \tau_{d,d} \end{pmatrix} \ .$$

The limiting probabilities σ_k, $1 \leq k \leq d$, of the algorithm being in state $\mathcal{S}'_k$, satisfy $\sum_{k=1}^{d}\sigma_k = 1$ and $T_d \cdot (\sigma_0 \cdots \sigma_d)' = (\sigma_0 \cdots \sigma_d)'$. Hence, upon putting

$$U_d = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ d/2^d & \tau_{1,1}-1 & \tau_{2,1} & \cdots & \tau_{d,1} \\ \binom{d}{2}/2^d & \tau_{1,2} & \tau_{2,2}-1 & \cdots & \tau_{d,2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d/2^d & \tau_{1,d-1} & \tau_{2,d-1} & \cdots & \tau_{d,d-1} \\ 1/2^d & \tau_{1,d} & \tau_{2,d} & \cdots & \tau_{d,d}-1 \end{pmatrix} \ ,$$

we have $U_d \cdot (\sigma_0 \cdots \sigma_d)^{\perp} = (1,0,\ldots,0)^{\perp}$. Hence, provided that U_d is invertible, $(\sigma_0 \cdots \sigma_d)^{\perp} = U_d^{-1} \cdot (1,0,\ldots,0)^{\perp}$ and in particular σ_0 is the value in the top left corner of U_d^{-1}.

We are interested in U_d only in the cases $d = 1, 2$ and 3. Upon putting $\alpha = 2^w$ and $\beta = (-1)^w$ we obtain

$$U_1 = \begin{pmatrix} 1 & 1 \\ \frac{1}{2} & \frac{\alpha+2\beta}{3\alpha}-1 \end{pmatrix} \quad , \quad U_2 = \begin{pmatrix} 1 & 1 & 1 \\ \frac{1}{2} & \frac{4\alpha^2+\alpha\beta+4\beta^2}{9\alpha^2}-1 & \frac{4(\alpha-\beta)(\alpha+2\beta)}{9\alpha^2} \\ \frac{1}{4} & \frac{(\alpha-\beta)(\alpha+2\beta)}{9\alpha^2} & \frac{(\alpha+2\beta)^2}{9\alpha^2}-1 \end{pmatrix} \quad \text{and}$$

$$U_3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{8} & \frac{(2\alpha+\beta)(2\alpha^2-\alpha\beta+2\beta^2)}{9\alpha^3} - 1 & \frac{4(\alpha^3-\beta^3)}{9\alpha^3} & \frac{4(\alpha-\beta)^2(\alpha+2\beta)}{9\alpha^3} \\ \frac{3}{8} & \frac{2(\alpha^3-\beta^3)}{9\alpha^3} & \frac{(\alpha+2\beta)(2\alpha^2-\alpha\beta+2\beta^2)}{9\alpha^3} - 1 & \frac{2(\alpha-\beta)(\alpha+2\beta)^2}{9\alpha^3} \\ \frac{1}{8} & \frac{(\alpha-\beta)^2(\alpha+2\beta)}{27\alpha^3} & \frac{(\alpha-\beta)(\alpha+2\beta)^2}{27\alpha^3} & \frac{(\alpha+2\beta)^3}{27\alpha^3} - 1 \end{pmatrix}.$$

The above matrices have been written down using `maple` [7]. With the same software it is immediate to verify that the matrix U_d is invertible for $d = 1, 2$ and 3, and to compute σ_0, *i.e.* the value of π in the introductory part of this section. We thus obtain the values $\pi = \pi^{(d)}$ given in equation (3), Theorem 3.6.

We now estimate the value of s at the first iteration of the main loop.

By virtue of Lemma 3.11, if $e_{j,n-1} = 0$ then $e_{j,n-1-t} = 0$ with probability $\frac{2}{3} + \frac{1}{3}(-\frac{1}{2})^t$, whereas if $e_{j,n-1} = 1$ then $e_{j,n-1-t} = 0$ with probability $\frac{2}{3} + \frac{1}{3}(-\frac{1}{2})^{t-1}$. Now one of the bits $e_{j,n-1}$ for $j = 1, ..., d$ is non-zero by assumption (the joint representation is proper) and all others can be considered randomly distributed: We now determine the exact probabilities with which the bits 0 and ± 1 occur as the most significant coefficient in a n-bit NAF, in order to compute the probability that, in the corresponding expansions, the $(n-1-t)$-th bit is 0.

We know that the interval $[-T_n, \ldots, T_n]$ contains $I_n = \frac{1}{3}\big(2^{n+2} - (-1)^n\big)$ integers, and it is easy to see that exactly $I_{n-1} = \frac{1}{3}\big(2^{n+2} - (-1)^n\big)$ of those lie in $[-T_{n-1}, \ldots, T_{n-1}]$, and $I_n - I_{n-1} = \frac{1}{3}\big(2^{n+1} - 2(-1)^n\big)$ do not. Hence, the probability that a random integer $\sum_{i=0}^{n-1} e_i 2^i$ of up to n bits in NAF is actually of length at most $n-1$ (or, in other words, that $e_{n-1} = 0$) is $\frac{2^{n+1}+(-1)^n}{2^{n+2}-(-1)^n} = \frac{1}{2} + O(2^{-n})$, where the implied constant in the error term is absolute and can be made explicit. (In fact $\big|\frac{2^{n+1}+(-1)^n}{2^{n+2}-(-1)^n} - \frac{1}{2}\big| \le \frac{c}{2^n}$ for all $c > \frac{3}{8}$ and n large enough. If $c = \frac{2}{5}$ then the inequality holds for all $n \ge 1$.) The same consideration holds for the probability that such an integer be exactly of length n.

Therefore, the probability that the t-th leftmost column $e^{(n-1-t)}$ of the joint representation is zero is

$$\chi_t = \left(\frac{2}{3} + \frac{1}{3}\Big(-\frac{1}{2}\Big)^{t-1}\right)\left(\frac{2}{3} + \frac{1}{12}\Big(-\frac{1}{2}\Big)^{t-1} + O(2^{-n})\right)^{d-1}$$

and the probability that the columns $e^{(n-w)}, e^{(n-w+1)}, \ldots, e^{(n-1-t)}$ are all zero is $\big(\frac{1}{2}\big)^{(w-1-t)d}\chi_t$ by virtue of Lemma 3.10. It follows that the expected number of zero columns at the beginning of the sequence formed by the columns $e^{(n-w)}$, $e^{(n-w+1)}, \ldots, e^{(n-2)}$ (recall that by assumption $e^{(n-2)}$ is nonzero) is $\langle s\rangle = \sum_{t=1}^{w-1}\big(\frac{1}{2}\big)^{(w-1-t)d}\chi_t$, i.e.

$$\begin{aligned}\langle s\rangle &= \sum_{t=1}^{w-1}\Big(\frac{1}{2}\Big)^{(w-1-t)d}\left(\frac{2}{3} + \frac{1}{3}\Big(-\frac{1}{2}\Big)^{t-1}\right)\left(\frac{2}{3} + \frac{1}{12}\Big(-\frac{1}{2}\Big)^{t-1} + O(2^{-n})\right)^{d-1} \\ &= \Big(\frac{2}{3}\Big)^d \sum_{t=1}^{w-1}\Big(\frac{1}{2}\Big)^{(w-1-t)d}\left(1 - \Big(-\frac{1}{2}\Big)^t\right)\left(1 - \Big(-\frac{1}{2}\Big)^{t+2}\right)^{d-1} + O(2^{-n}) \qquad (5)\end{aligned}$$

where the implied constant in the error term in (5) depends on d alone, and, for large w:

$$\begin{aligned}\langle s\rangle &= \Big(\frac{2}{3}\Big)^d \sum_{t=1}^{w-1}\Big(\frac{1}{2}\Big)^{(w-1-t)d} + O(\sigma^{-w} + 2^{-n}) \\ &= \Big(\frac{2}{3}\Big)^d \frac{1}{1-2^{-d}} + O(\sigma^{-w} + 2^{-n}) \qquad (6)\end{aligned}$$

for some constant $\sigma > 1$ which can be determined explicitly. □

Remark 3.12 *Consider the case $d = w = 2$, which is quite interesting in view of the applications. Computing $\langle s \rangle$ exactly assuming $n = 40$, resp. 80 and with formula (5) gives results which differ by less than 2^{-39}, resp. 2^{-79}. For larger values of n the difference is accordingly smaller. For this choice of the parameters $\langle s \rangle \approx \frac{3}{4}$ as computed by (5). If $d = 2$ and $w = 3$ then $\langle s \rangle \approx \frac{1}{2}$ and $d = 3$ and $w = 2$ we have $\langle s \rangle \approx \frac{9}{16}$. For these choices of the parameters d and w, the values given by (6) are extremely imprecise.*

Remark 3.13 *The following simple, alternative proof of Lemma 3.10, is by Robert Israel.*

To establish dependency probabilities between adjacent digits in all NAFs, we can consider random infinite NAFs. Hence we consider representations of real numbers: $x = \sum_{i=i_0}^{\infty} a_i 2^{-i}$. The numbers starting with 0. (i.e. with $i_0 = 0$ and $a_0 = 0$) range from $0.\bar{1}0\bar{1}0\ldots = -\frac{1}{2} - \frac{1}{8} - \ldots = -\frac{2}{3}$ to $0.1010\ldots = \frac{2}{3}$. Of these, those beginning with $0.\bar{1}$ range from $-\frac{2}{3}$ to $0.\bar{1}0101\ldots = -\frac{1}{3}$, i.e. lie in the interval $\left[-\frac{2}{3},-\frac{1}{3}\right]$, those starting with 0.0 are in $\left[-\frac{1}{3},\frac{1}{3}\right]$, and those beginning with 0.1 are in $\left[\frac{1}{3},\frac{2}{3}\right]$. The same proportions occur for any initial segment ending in 0. So, after a 0, the probabilities for the next digit are $\frac{1}{4}$ for $\bar{1}$, $\frac{1}{2}$ for 0 and $\frac{1}{4}$ for 1.

3.3 Using the JSF

The Joint Sparse Form has been introduced by Solinas [33] to make the Straus-Shamir trick more effective for elliptic curves. It applies however to all groups where inversion is for free. It has been defined only for *pairs* of integers. Accordingly, we will restrict ourselves to the case $d = 2$ here. We shall also assume that $w = 2$: this assumption fits naturally with the defining properties of the JSF, and by a good stoke of luck this brings the highest improvement (over the methods studied before) for exponents in the range in which we are interested. For precise comparisons see Subsection 4.1, in particular the row for $w = 2$ in Table 1 and Remark 4.1 (1).

In this subsection we prove the following theorem.

Theorem 3.14 *Suppose that in Algorithm 2.2 Solinas' JSF is used for the exponents, and $w = d = 2$. Assume further that the JSF of the exponents has length n.*

The expected number of multiplications in the main loop of the algorithm is $\frac{3}{8}n - 1$, and the heuristically expected number of squarings is $n - 2 + \frac{1}{2} = n - \frac{3}{2}$.

The set $\mathcal{E}$ consists of the 12 elements $g_1^a g_2^b$ with: (i) *$a = 0$ and $b = 1$;* (ii) *$a = 1$ and $-2 \leq b \leq 2$;* (iii) *$a = 2$ and $b \in \{\pm 1, \pm 3\}$ and* (iv) *$a = 3$ and $b = \pm 2$. A chain for precomputing $\mathcal{E}$ is*

$$\begin{gathered} \{\ g_1,\quad g_2,\quad g_1 g_2,\quad g_1 g_2^{-1},\quad g_1 g_2^{2},\quad g_1 g_2^{-2}, \\ g_1^2 g_2,\quad g_1^2 g_2^{-1},\quad g_1^2 g_2^{3},\quad g_1^2 g_2^{-3},\quad g_1^3 g_2^{2},\quad g_1^3 g_2^{-2}\ \}\,. \end{gathered} \tag{7}$$

We assume that the reader is acquainted with the results in Solinas' cited technical report, from which we recall however a few important facts. The joint Hamming weight of the JSF of two integers is minimal among all (un)signed joint binary representations of the same pair of integers. Its average density is 1/2 – which gives the heuristic estimate of the squarings in the main loop – whereas that of the joint unsigned binary representation and of the joint NAF is 3/4 and 5/9 respectively. It is natural then to expect that using the JSF in Algorithm 2.2 would lead to an improvement over the complexities of the other two cases even if $w > 1$.

The JSF is uniquely determined by the following properties:

(JSF-1) Of any three consecutive columns, at least one is zero.

(JSF-2) Adjacent nonzero bits have the same sign, *i.e.* $e_{i,j+1}e_{i,j} = 0$ or 1.

(JSF-3) If $e_{i,j+1}e_{i,j} \neq 0$ then $e_{3-i,j+1} \neq 0$ and $e_{3-i,j} = 0$.

Solinas provides proofs for existence and uniqueness of the JSF, as well as an algorithm for determining it. His algorithm generates the JSF right-to-left.

Property **(JSF-1)** suggests that the representation is particularly suitable for an implementation of Algorithm 2.2 with a window width $w = 2$. As already announced we restrict ourselves to this case from now on. Further, this choice also simplifies the complexity analysis, by the following observation: Algorithm 2.2 scans a joint representation left-to-right in order to form windows, but *consecutive nonzero columns always belong to one window regardless of the direction in which we are scanning the joint representation.* In fact, by property **(JSF-1)** there can be at most two consecutive nonzero columns, which must be preceded and followed by zero columns or by the boundaries of the representation.

Therefore to estimate the number of nonzero windows (which corresponds to the number of multiplications performed by Algorithm 2.2 plus one) we scan our input right-to-left. In the analysis of his algorithm Solinas considers three states which he calls States A, B and C. In State C this algorithm outputs a zero column. In States A or B it outputs nonzero columns. State A is always followed by State B, State B by State C, and the transition probabilities are: $\mathcal{P}(C \mapsto A) = 1/4$, $\mathcal{P}(C \mapsto B) = 1/2$ and $\mathcal{P}(C \mapsto C) = 1/4$. We thus consider a Markov chain with *three* states:

$\mathcal{S}_0^*$. A nonzero column is output by State A of Solinas' algorithm: this column will be the second column in a "square" window when read left-to-right, as the next state in Solinas' algorithm is always State B.

$\mathcal{S}_1^*$. A nonzero column is output by State B of Solinas' algorithm: this column will be the first column in a window when read left-to-right, whereas the second column is non-zero if we are coming from state A or zero if we come from State C.

$\mathcal{S}_2^*$. A zero column is output by State C of Solinas' algorithm.

The number of times we enter in $\mathcal{S}_1^*$ equals the number of windows formed and thus to the number of multiplications performed by the algorithm. The transition probability matrix is

$$T = \left(\mathcal{P}(\mathcal{S}_i^* \mapsto \mathcal{S}_j^*)\right)_{i,j=0}^{2} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1/4 & 1/2 & 1/4 \end{pmatrix}$$

which yields limiting probabilities $\pi_0 = \frac{1}{8}$, $\pi_1 = \frac{3}{8}$ and $\pi_2 = \frac{1}{2}$. Hence the expected number of multiplications performed by Algorithm 2.2 is $\frac{3}{8}n - 1$ with n-bit inputs.

According to the defining properties of the JSF, the admissible nonzero columns $\binom{e_{1,j}}{e_{2,j}}$ and windows $\left(\begin{smallmatrix} e_{1,j} & e_{1,j-1} \\ e_{2,j} & e_{2,j-1} \end{smallmatrix}\right)$ with both columns non zero that, up to sign, can be found are

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ \pm 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ \pm 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & \pm 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ \epsilon & \epsilon \end{bmatrix} \text{ with } \epsilon = \pm 1, \text{ and } \begin{bmatrix} 1 & 1 \\ \pm 1 & 0 \end{bmatrix},$$

thus proving the statements about $\mathcal{E}$. □

4 Comparisons and Applications

4.1 Optimal parameters for double and triple exponentiation

We determine for which values of the parameter w the algorithms run fastest, given the bit length n of the inputs and the number d of the exponents. For simplicity we ignore the number of squarings performed in the main loop and we consider the performance of the algorithms only for $d = 2$ and 3.

Suppose first $d = 2$: Table 1 contains the cardinality of $\mathcal{E}$ and the sum of the number of operations needed to build it with the expected number of multiplications in the main loop of the algorithm. This performance parameter (similar to that used for instance in [20]) is a natural way of comparing exponentiation algorithms: it is easy to adapt it to the relative costs of squarings by adding $c_s\, n$, where c_s is the cost of a squaring relative to that of a multiplication. In the column for the JSF there is of course no entry for $w = 3$.

Table 2 collects the analogous data for $d = 3$. Note that the JSF, being defined only for $d = 2$, is not represented.

w	Unsigned $\#\mathcal{E}$ and # Ops		NAF $\#\mathcal{E}$ and # Ops		JSF $\#\mathcal{E}$ and # Ops	
1	3	$\frac{3}{4}n$	4	$1+\frac{5}{9}n$	4	$1+\frac{1}{2}n$
2	12	$9+\frac{3}{7}n$	8	$5+\frac{11}{27}n$	12	$9+\frac{3}{8}n$
3	48	$45+\frac{3}{10}n$	48	$45+\frac{32}{117}n$		

Table 1: Cardinality of $\mathcal{E}$ and number of operations for $d = 2$

w	Unsigned $\#\mathcal{E}$ and # Ops		NAF $\#\mathcal{E}$ and # Ops	
1	7	$3+\frac{7}{8}n$	13	$9+\frac{19}{27}n$
2	56	$52+\frac{7}{15}n$	49	$45+\frac{131}{297}n$
3	448	$444+\frac{7}{22}n$	603	$599+\frac{1082}{3645}n$

Table 2: Cardinality of $\mathcal{E}$ and number of operations for $d = 3$

Remarks 4.1 (1) *Assume $d = 2$ and consider Table 1. Using the unsigned binary representation, the optimal choice of w is $w = 1$ for $n \leq 28$, and $w = 2$ for $28 \leq n \leq 280$. In particular the parameter $w = 2$ is optimal for the exponents sizes which interests us.*

With the NAF the thresholds are $n = 27$ and $n = \frac{14040}{47} = 298.72$ respectively.

With the JSF the parameter $w = 1$ is optimal for $n \leq 64$. Furthermore, using the JSF with $w = 2$ is better than using the NAF with either $w = 2$ or 3 when $124 < n \leq 354$: in

the range which concerns us most using the NAF can be marginally slower but requires fewer precomputations.

(2) *In the case* $d = 3$ *(see Table 2) the thresholds are higher, as intuition suggests. Using the unsigned binary representation, the optimal choice of* w *is* $w = 1$ *for* $n \leq 120$*, and* $w = 2$ *for* $121 \leq n \leq 2640$*. In the NAF case,* $w = 1$ *is optimal for* $n \leq 137$ *and* $w = 2$ *for* $138 \leq n \leq 3841$*.*

If $w = 1$*, the NAF leads to better performance as long as* $n > 35$*, if* $w = 2$ *the NAF will always yield a better algorithm. If* $w = 3$*, the unsigned binary representation is better for* $n \leq 7264$*.*

Remark 4.2 *We mention other (multi-)exponentiation techniques, which can also be applied to all groups where our methods can be used. One approach has been called* interleaved exponentiation *by Möller [20]. It's actually an idea which has been rediscovered several times: for an earlier description see [19], and a two base case appears in [30]. In terms of exponent recording it can be seen as Algorithm 2.1 applied to a different, less dense, representation of the exponents: As an example we combine it with signed sliding windows [9, 8, 31, 32] of width* $w = 4$*. The resulting algorithm performs double exponentiations in a group with free inversion with* $15 + \frac{n}{3}$ *multiplications and about* n *squarings, for* n*-bit exponents, using* 16 *precomputed values. For* $n = 160$ *the performance is similar to that of the method analyses in this paper with the JSF, but the memory usage is greater. Möller [21] also extends the sliding window methods a bit to get possibly better performance, by allowing a few more precomputations: His* signed fractional window method *with* $w = 3.5$ *applied to interleaved double exponentiation uses* 12 *precomputed values, exactly as sliding windows of width* 2 *over a JSF, but uses* $9 + \frac{4}{11}n$ *multiplications (and about* n *squarings), which is* $\frac{1}{88}n$ *less multiplications than sliding windows over a JSF.*

4.2 Applications

As mentioned in the introduction, in cryptography multi-exponentiation are normally used to verify digital signatures, and in this case only double and triple exponentiations are needed.

However, in some algebraic structures the computation of a single exponentiation can be reduced to such a product: In fact the arguments of [14] essentially apply to the problem of computing g^e in a group G of order ℓ, where G admits an efficiently computable automorphism σ acting on G like the exponentiation by an integer s. This integer is a root of the characteristic polynomial of σ modulo ℓ. If σ is such that for every $e \in [0, \ldots, \ell - 1]$ one can write $e \equiv e_1 + e_2 s \pmod{\ell}$ with $e_1, e_2 = O(\sqrt{\ell})$, then computing $g^{e_1} \cdot \sigma(g)^{e_2}$ is usually faster than computing of g^e. Gallant, Lambert and Vanstone reduce this problem to that of finding a short vector in a lattice, which they solve using the extended euclidian algorithm. A gap in their arguments is filled in [29]. In general it is possible to write $e = \sum_{i=0}^{d-1} e_i s^i$ if the minimal polynomial of σ has degree at least d, where the bounds on the e_i depend on the coefficients of the minimal polynomial [29, Theorem 5].

Apart from some families of elliptic [14] and hyperelliptic [17, 26, 10] curves, there are other groups which profit from such techniques. Notable examples are the XTR group and trace zero varieties, which we now briefly recall.

The XTR cryptosystem [37] makes use of the subgroup G of order $p^2 - p + 1$ of the multiplicative group of the finite field extension $\mathbb{F}_{p^6}/\mathbb{F}_p$. In the original XTR cryptosystem

elements from the field $\mathbb{F}_{p^6}$ are replaced by their traces over $\mathbb{F}_{p^2}$, leading to very good performance. Lenstra and Stam in [34] construct the field extension by means of an optimal normal basis and work directly with the group elements instead of the traces. With this representation inversion is for free and exponentiations can be reduced to double exponentiations with exponents bounded by the square root of the group order.

A trace zero variety is an Abelian variety constructed by Weil Descent [13] from an elliptic curve [23] or the Jacobian of a hyperelliptic curve [17, 18]: Let $\mathcal{C}$ be a hyperelliptic curve of genus g defined over a prime field $\mathbb{F}_p$, and assume that the characteristic polynomial of the Frobenius endomorphism σ on the Jacobian $\mathcal{J}_\mathcal{C}$ of $\mathcal{C}$ is known. Let d be a (small) odd prime. The *trace zero variety* (of $\mathcal{J}_\mathcal{C}$ relative to the extension $\mathbb{F}_{p^d}/\mathbb{F}_p$) is the kernel of the map $D \mapsto (\sigma^d - 1)/(\sigma - 1)(D)$ on the group of rational divisor classes of $\mathcal{C}$ over the finite field extension $\mathbb{F}_{p^d}$. It is an Abelian subvariety of $\mathcal{J}_\mathcal{C}$ of dimension $(d-1)g$ over $\mathbb{F}_p$, which we shall denote by G. We call G_0 the subgroup of G of large prime order ℓ in which we actually implement the cryptographic primitives. It is $\ell \approx p^{(d-1)g}$.

It has been noted [23, 18] that if $(d-1)g \leq 4$ then the best attacks known to work on trace zero varieties have complexity $O(\sqrt{G_0})$. Hence we restrict our attention to the cases $g = 1$ with $d = 3$ or 5 and $g = 2$ with $d = 3$.

The automorphism σ of G_0 induced from the Frobenius automorphism of $\mathbb{F}_{p^d}/\mathbb{F}_p$ has degree d. Hence a single exponentiation in G_0 can be computed as a $(d-1)$-uple exponentiation with exponents $O(\ell^{1/d})$, which require less group operations. Furthermore, group operations on a trace zero variety are faster than on an elliptic or hyperelliptic curve of comparable size: this is a consequence of the fact that the field operations to be performed are the same for elliptic or hyperelliptic curves over prime fields, but the arithmetic in the extension field is faster than in a prime field of the same size (most arguments of [3] apply to this situation, too). This makes trace zero varieties interesting and worthy of deeper analysis for consideration in public-key cryptosystems.

ACKNOWLEDGEMENTS. The author expresses his gratitude to Gerhard Frey for his steady encouragement and support, to Tanja Lange, who drew the author's attention to Solinas' work and proofread the manuscript, and to Martijn Stam, for finding an imprecision in an early version. The author would like to thank the anonymous referees for their very useful comments. Many thanks also to Arjen Lenstra for kindly providing a reprint of [38], and to Robert Israel and Helmut Prodinger for interesting (electronic) discussions. Bernstein's treatment [4] of Pippenger's exponentiation algorithm has been useful for correctly crediting some ideas. Some computations have been performed using the `maple` computer algebra system [7]. The author's gratitude goes also to Alessandro Baldaccini for reviewing his English: All remaining mistakes are the author's fault.

NOTE. During the preparation of the final version of this paper, the author became aware of [15], then a preprint, containing a computation of a minimal joint signed bit representation of several integers, and where some properties of left-to-right computations are discussed.

References

[1] American National Standards Institute, *ANSI X9.62: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. 1999.

[2] S. Arno and F.S. Wheeler, *Signed digit representations of minimal Hamming weight*. IEEE Transactions on Computers **42** (1993), pp. 1007–1010.

[3] R. Avanzi and P. Mihăilescu, *Improved Arithmetic Algorithms for Processor Adequate Finite Fields and related algebraic structures*. To appear in: *Proceedings of Selected Areas in Cryptography 2003*, Carleton University, Ottawa, Ontario, Canada, August 14-15, 2003.

[4] D.J. Bernstein, *Pippenger's exponentiation algorithm*. Preprint. Available from `http://cr.yp.to`

[5] A.D. Booth, *A signed binary multiplication technique*. The Quarterly Journal of Mechanics and Applied Mathematics **4** (1951), pp. 236–240.

[6] E.F. Brickell and K.S. McCurley, *Interactive identification and digital signatures*, AT&T Technical Journal, 1991, pp. 74–86.

[7] B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan and S.M. Watt, *Maple V Language Reference Manual*. Springer, 1991.

[8] H. Cohen, *Analysis of the flexible window powering algorithm*. Preprint.
Available from: `http://www.math.u-bordeaux.fr/~cohen/`

[9] H. Cohen, A. Miyaji and T. Ono, *Efficient elliptic curve exponentiation*. In *Proceedings ICICS'97*, LNCS 1334, Springer-Verlag, 1997, pp. 282–290.

[10] I. Duursma, P. Gaudry and F. Morain, *Speeding up the discrete log computation on curves with automorphisms.* In *Advances in Cryptology, Asiacrypt 99*. LNCS 1716, pp. 103–121. Springer Verlag, 1999,.

[11] T. ElGamal, *A public-key cryptosystem and a signature scheme based on discrete logarithms*. IEEE Transactions on Information Theory IT-31 (1985), pp. 469-472.

[12] Ö. Eğecioğlu and Ç. K. Koç, *Exponentiation using canonical recoding.* Theoretical Computer Science, 129(2), pp. 407–417, 1994.

[13] G. Frey, *Applications of arithmetical geometry to cryptographic constructions.* In *Finite fields and applications (Augsburg, 1999)*, pp. 128–161. Springer, Berlin, 2001.

[14] R.P. Gallant, R.J. Lambert, and S.A. Vanstone, *Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms* In *Advances in Cryptology – CRYPTO 2001 Proceedings*, LNCS 2139, pp. 190–200. Springer Verlag, 2001.

[15] P. Grabner, C. Heuberger, and H. Prodinger, *Distribution results for low-weight binary representations for pairs of integers*. Theoretical Computer Science, to appear.

[16] D. E. Knuth, *The art of computer programming. Vol. 2, Seminumerical algorithms*, third ed., *Addison-Wesley Series in Computer Science and Information Processing.* Addison-Wesley, Reading, MA, 1997.

[17] T. Lange, *Efficient Arithmetic on Hyperelliptic Curves*. Ph.D. Thesis, Universität Essen, 2001.

[18] T. Lange, *Trace-Zero Subvarieties for Cryptosystems*. To appear in: Ramanujan Journal of Mathematics.

[19] C.H. Lim, *Efficient multi-exponentiation and application to batch verification of digital signatures*. Unpublished manuscript. August 2000.
See: `http://dasan.sejong.ac.kr/~chlim/english_pub.html`

[20] B. Möller, *Algorithms for Multi-exponentiation*. In: S. Vaudenay, A.M. Youssef (Eds.), *Selected Areas in Cryptography - SAC 2001*. LNCS 2259, pp. 165-180. Springer-Verlag 2001.

[21] B. Möller, *Improved Techniques for Fast Exponentiation*. In: P.J. Lee, C.H. Lim (Eds.), *Information Security and Cryptology – ICISC 2002*, proceedings. LNCS 2587, pp. 298–312. Springer-Verlag 2002.

[22] F. Morain and J. Olivos, *Speeding up the computations on an elliptic curve using addition-subtraction chains*. RAIRO Inform. Theory **24** (1990), pp. 531–543.

[23] N. Naumann, *Weil-Restriktion abelscher Varietäten*. Master's thesis, Universität Essen, 1999.

[24] National Institute of Standards and Technology (NIST), *A proposed federal information processing standard for digital signatures standard (DSS)*, Federal Register 1991, vol. 56, pp. 42980–42982.

[25] K. Okeya and K. Sakurai, *Fast Multi-Scalar Multiplication Methods on Elliptic Curves with Precomputation using Montgomery Trick.* In: B.S. Kaliski et al (Eds.), *CHES 2002*. LNCS 2523, pp. 564–578. Springer-Verlag, 2003.

[26] Y-H. Park, S. Jeong and J. Lim. *Speeding Up Point Multiplication on Hyperelliptic Curves with Efficiently-computable Endomorphisms.* In L. Knudsen, editor, *Advances in Cryptology - Proceedings of EUROCRYPT 2002*. LNCS 2332, pp. 197–208. Springer-Verlag, 2002.

[27] G. W. Reitwiesner, *Binary arithmetic.* Advances in Computers **1**, pp. 231–308 (1960).

[28] C.P. Schnorr, *Efficient identification and signatures for smart cards*, in *Advances in Cryptology, Crypto'89*. LNCS 435, pp. 239–252. Springer-Verlag, 1990.

[29] F. Sica, M. Ciet and J.-J. Quisquater, *Analysis of the Gallant-Lambert-Vanstone Method based on Efficient Endomorphisms: Elliptic and Hyperelliptic Curves.* In *Proceedings of Selected Areas of Cryptography 2002*. LNCS 2595, pp. 21-36. Springer-Verlag, 2003.

[30] S.G.Sim and P.J.Lee. *An efficient implementation of two-term exponentiation in elliptic curves* In *Japan–Korea Joint Workshop on Information Security and Cryptology (JW-ISC 2000)*, Jan. 25-26, 2000, Naha, Okinawa, Japan, pp. 61–68.

[31] J.A. Solinas, *An improved algorithm for arithmetic on a family of elliptic curves.* In *Advances in Cryptology – CRYPTO '97*. LNCS 1294, pp. 357–371. Springer-Verlag 1997.

[32] J.A. Solinas, *Efficient arithmetic on Koblitz curves*. Designs, Codes and Cryptography **19** (2000), pp. 195–249.

[33] J.A. Solinas, *Low-Weight Binary Representations for Pairs of Integers*. Centre for Applied Cryptographic Research, University of Waterloo, Combinatorics and Optimization Research Report **CORR 2001-41**, 2001. Available from:
`http://www.cacr.math.uwaterloo.ca/techreports/2001/corr2001-41.ps`

[34] M. Stam and A.K. Lenstra, *Efficient subgroup exponentiation in quadratic and sixth degree extensions*. In *Proceedings of CHES 2002*. LNCS 2523, pp. 318-332. Springer-Verlag 2003.

[35] E.G. Straus, *Addition chains of vectors (problem 5125).* American Mathematical Monthly, vol. 71, 1964, pp. 806–808.

[36] E.G. Thurber, *On addition chains $l(mn) \leq l(n)b$ and lower bounds for $c(r)$*. Duke Mathematical Journal **40** (1973), 907–913. MR 48 #8429.

[37] E.R. Verheul and A.K. Lenstra, *The XTR public key system.* In *Advances in Cryptography – Crypto 2000*. LNCS 1880, pp. 1–19. Springer-Verlag, 2000.

[38] S.-M. Yen, C.-S. Laih and A.K. Lenstra, *Multi-exponentiation.* IEE proceedings: computers and digital techniques vol. 141, No. 6, november 1994.