S. Vaudenay, A. M. Youssef (Eds.): *Selected Areas in Cryptography – SAC 2001,*
Springer-Verlag LNCS 2259, pp. 165–180, ISBN 3-540-43066-0.

Algorithms for Multi-exponentiation

Bodo Möller

Technische Universität Darmstadt, Fachbereich Informatik
moeller@cdc.informatik.tu-darmstadt.de

Abstract. This paper compares different approaches for computing power products $\prod_{1\le i\le k} g_i^{e_i}$ in commutative groups. We look at the conventional simultaneous exponentiation approach and present an alternative strategy, interleaving exponentiation. Our comparison shows that in general groups, sometimes the conventional method and sometimes interleaving exponentiation is more efficient. In groups where inverting elements is easy (e.g. elliptic curves), interleaving exponentiation with signed exponent recoding usually wins over the conventional method.

1 Introduction

A common task in implementations of many public-key cryptosystems is multi-exponentiation in some commutative group G, i.e. evaluating a product

$$\prod_{1\le i\le k} g_i^{e_i}$$

where $k \ge 2$ is a small integer, each g_i is an element of G, and each e_i is an integer (typically a few hundred up to a few thousand bits long). We require that the e_i be non-negative (otherwise, invert g_i). Example groups include $(\mathbb{Z}/n\mathbb{Z})^*$ for some integer n, e.g. for verification of ElGamal [11] or DSA [17] signatures; groups of rational points on elliptic curves over finite fields, e.g. for verification of ECDSA [1] signatures; and class groups of imaginary-quadratic orders, e.g. for verification of RDSA [2][7] signatures. We have $k = 2$ for DSA and ECDSA verification and $k = 3$ for ElGamal and RDSA verification. Larger values of k appear in protocols of Brands [4]. In the present paper, we allow $k = 1$ as well for algorithms; efficiency considerations may ignore this case. It is well known that in general it is unnecessarily inefficient to compute the powers $g_i^{e_i}$ separately and then multiply them. Instead, specific algorithms for multi-exponentiation are usually applied.

We assume that the e_i consist of independent random bits up to a respective maximum bit-length b_i; i.e., e_i is a uniformly distributed random integer in the interval $[0, 2^{b_i} - 1]$. (In practice the actual distribution may differ, but for typical cases this simplified assumption is reasonably close.) In this setting, we consider general algorithms for arbitrary exponents; we do not examine algorithms based on tailor-made addition chains in $\mathbb{Z}^k$ for given $e_1, \ldots, e_k$ (cf. [3]). (Note that even if an exponent is fixed in a cryptographic protocol, it is sometimes desirable to perform computations using varying exponents in order to thwart side-channel attacks that try to use timings [12] or power consumption measurements [13] or other extra data to gain knowledge on secret exponents. To avoid constant exponents, g^e can be rewritten as $g^{n\cdot\mathrm{ord}(g)+e}$ and $\prod g_i^{e_i}$ can be rewritten as $(\prod g_i)^n \prod g_i^{e_i-n}$ for arbitrary integers n.)

Like window-based algorithms for single exponentiations, the algorithms that we analyse work in two stages: First, in the *precomputation stage,* an auxiliary table of group elements is computed from the elements g_i; then, in the *evaluation stage,* the final result is computed using these auxiliary values.

The usual approach for multi-exponentiation combines all input group elements g_i with each other in the precomputation stage ([11], [20], [21]); then the evaluation stage looks at all exponents simultaneously. In the present paper, we discuss an alternative approach where the g_i are treated separately in the precomputation stage. In this approach, the evaluation stage uses an interleaving of the generators and exponents for the various i rather than handling multiple i simultaneously.

We collectively refer to the multi-exponentiation methods described in [11], [20] and [21] as *"simultaneous exponentiation"*. Section 2 describes these methods. Section 3 presents two variants of our alternative approach, which we dub *"interleaving exponentiation":* a basic method and an alternative method that can be used in groups where inverting elements is easy. In section 4, we compare the efficiency of simultaneous exponentiation methods and interleaving exponentiation methods. Section 5 discusses variants that can be advantageous when all bases g_i are fixed.

In specific groups, additional useful efficiently computable endomorphisms are available besides squaring and possibly inversion (see e.g. [19]); this may lead to better multi-exponentiation algorithms for these groups. Such special groups are out of the scope of the present paper.

1.1 Notation

We write $e[j]$ for bit j of a non-negative integer e. For negative j, we define that $e[j] = 0$. We write $e[j \ldots j']$ for the integer consisting of the concatenation of bits j down to j' of e; e.g., if $e = 10111_2 = 23$, then $e[3 \ldots 1] = 011_2 = 3$ and $e[1 \ldots -2] = 1100_2 = 12$.

2 Simultaneous Exponentiation Methods

We look at two multi-exponentiation methods using simultaneous exponentiation (as opposed to interleaving exponentiation, which is introduced in section 3): Straus's 2^w-ary method (section 2.1) and the sliding window method of Yen, Laih, and Lenstra (section 2.2). (The method known as "Shamir's trick" appears as a special case of both of these.)

As noted in the introduction, all algorithms that we consider are related and work in two stages: First, the *precomputation stage* prepares an auxiliary table of group elements; then, the *evaluation stage* computes the final result using this table. For comparing different methods, we examine the two stages separately.

When examining simultaneous exponentiation algorithms, we assume that b_i is the same for all i. Let at least one e_i be non-zero, and let b be the bit-length of the longest of the e_i. Parameter w is always a positive integer, the "window size"; larger window sizes make the precomputation stage less efficient, but speed up the evaluation stage. It is not possible to give a general rule for selecting an optimal w (cf. section 4).

Relevant features of the precomputation stage are the number of group operations required for computing the auxiliary table, and the number of table entries. For group operations, we differentiate between squarings and general multiplications, since the former often can be computed more efficiently. The precomputed tables will always contain the values $g_1, \ldots, g_k$, all of which are trivially available and hence can be neglected. It will be visible that computing each additional table entry requires one multiplication or, for some of the table entries in the simultaneous 2^w-ary method, one squaring. In addition to this, k squarings are needed by the simultaneous sliding window method if $w > 1$.

The evaluation stage requires both squarings and multiplications. For each multi-exponentiation method, we look at the number of squarings and the expected number of general multiplications for given k, b, and w. w is assumed to be small in comparison to b (otherwise the precomputation stage would become unreasonably expensive).

It should be noted that a slight optimisation for the precomputation stage is possible in all methods by first looking which table entries are actually needed (either during the evaluation stage, or because other precomputed table entries that are needed in the evaluation stage depend on them) and limiting precomputation to these. As this optimisation will usually only have a small effect in practice, we neglect it in our comparisons.

For the number of squarings in the evaluation stage, we assume that the following optimisation is used: As initially variable A is 1_G (the neutral element of G) in all algorithms, squarings can easily be avoided until a different value has been assigned to A.

Formulas for the expected number of multiplications during the evaluation stage given in the following are actually asymptotics for large b/w rather than precise values (we do not take into account the special probability distributions encountered at both ends of the exponents). As in practice w will be much smaller than b, the error is negligible for our purposes.

Just as squarings can be eliminated in the evaluation stage while A is 1_G, the first multiplication of A by a table entry can be replaced by an assignment. This minor optimisation is not used in our figures below; note that it applies similarly to all algorithms discussed in this paper (and does not affect asymptotics), so comparisons between different methods remain just as valid.

2.1 Simultaneous 2^w-Ary Method

The simultaneous 2^w-ary exponentiation method [20] (see also [15]) looks at w bits of each of the exponents for each evaluation stage group multiplication, i.e. kw bits in total. The special case where $w = 1$ is also known as "Shamir's trick" since it was described in [11] with a reference to Shamir.

Precomputation Stage. Precompute $\prod_{1 \le i \le k} g_i^{E_i}$ for all non-zero k-tuples $(E_1, \ldots, E_k) \in \{0, \ldots, 2^w - 1\}^k$.

Number of non-trivial table entries: $2^{kw} - 1 - k$. Of these, $2^{k(w-1)} - 1$ can be computed by squaring other table entries (all the E_i are even). The remaining $2^{kw} - 2^{k(w-1)} - k$ entries require one general multiplication each.

No additional squarings are required.

Evaluation Stage.

```
$A \leftarrow 1_G$
for $j = \lfloor (b-1)/w \rfloor w$ down to 0 step $w$ do
  for $n = 1$ to $w$ do
    $A \leftarrow A^2$
  if $\bigl(e_1[j+w-1 \ldots j], \ldots, e_k[j+w-1 \ldots j]\bigr) \neq (0, \ldots, 0)$ then
    $A \leftarrow A \cdot \prod_i g_i^{e_i[j+w-1 \ldots j]}$ {multiply $A$ by table entry}
return $A$
```

Number of squarings: $\left\lfloor \dfrac{b-1}{w} \right\rfloor w$.

Expected number of multiplications: $b \cdot \dfrac{1 - \frac{1}{2^{kw}}}{w}$.

2.2 Simultaneous Sliding Window Method

The simultaneous sliding window exponentiation method of Yen, Laih, and A. Lenstra [21] is an improvement of the 2^w-ary method described in section 2.1. Due to the use of a sliding window, table entries are required only for those tuples $(E_1, \ldots, E_k)$ where at least one of the E_i is odd. (Note that while values g_i^2 no longer appear in the precomputed table, the precomputation stage now needs them as intermediate values unless $w = 1$.) Also the expected number of multiplications required in the evaluation stage is reduced. Like the 2^w-ary method, this method looks at w bits of each of the exponents for each evaluation stage group multiplication (kw bits in total). For $w = 1$, this again is "Shamir's trick". For $k = 1$, this is the usual sliding window method for a single exponentiation (see e.g. [15]).

Precomputation Stage. Precompute $\prod_{1 \le i \le k} g_i^{E_i}$ for all k-tuples $(E_1, \ldots, E_k) \in \{0, \ldots, 2^w - 1\}^k$ where at least one of the E_i is odd.

Number of non-trivial table entries (multiplications): $2^{kw} - 2^{k(w-1)} - k$.

Number of squarings: k if $w > 1$; none otherwise.

Evaluation Stage.

```
$A \leftarrow 1_G$
$j \leftarrow b - 1$
while $j \geq 0$ do
  if $\forall i \in \{1, \ldots, k\}\colon e_i[j] = 0$ then
    $A \leftarrow A^2$; $j \leftarrow j - 1$
  else
    $j_{\mathrm{new}} \leftarrow \max(j - w, -1)$
    $J \leftarrow j_{\mathrm{new}} + 1$
    while $\forall i \in \{1, \ldots, k\}\colon e_i[J] = 0$ do
      $J \leftarrow J + 1$
    {now $j \geq J > j_{\mathrm{new}}$}
    for $i = 1$ to $k$ do
      $E_i \leftarrow e_i[j \ldots J]$
    while $j \geq J$ do
      $A \leftarrow A^2$; $j \leftarrow j - 1$
    $A \leftarrow A \cdot \prod_i g_i^{E_i}$ {multiply $A$ by table entry}
    while $j > j_{\mathrm{new}}$ do
      $A \leftarrow A^2$; $j \leftarrow j - 1$
return $A$
```

Number of squarings: $b - w$ up to $b - 1$.
Expected number of multiplications: $b \cdot \dfrac{1}{w + \sum_{n \geq 1} \frac{1}{2^{kn}}} = b \cdot \dfrac{1}{w + \frac{1}{2^k - 1}}$.

3 Interleaving Exponentiation Methods

Here, we look at two interleaving exponentiation algorithms: A basic algorithm suitable for arbitrary groups (section 3.1) and a special variant using signed exponent recoding that can be applied if inverting elements is easy (section 3.2).

The comments in the introduction to section 2 apply similarly, with the exception that we no longer assume all the b_i to be identical. Instead of a single window size w, in this section we have k possibly different window sizes w_i $(1 \leq i \leq k)$ used for the respective parts of the multi-exponentiation; each w_i is a small positive integer. Again we assume that initial squarings are eliminated while A is 1_G.

Note that for the algorithms described in this section, the precomputed table has disjoint parts for different bases g_i. If multiple multi-exponentiations have to be performed and some of the bases g_i appear again, then the corresponding parts of earlier precomputed tables can be reused.

3.1 Basic Interleaving Exponentiation Method

The basic interleaving exponentiation method is a generalization of the sliding window method for a single exponentiation (see e.g. [15]), to which it corresponds in case $k = 1$.

Precomputation Stage. For $i = 1, \ldots, k$, precompute g_i^E for all odd E such that $1 \leq E \leq 2^{w_i} - 1$.

Number of non-trivial table entries (multiplications): $\bigl(\sum_{1 \leq i \leq k} 2^{w_i - 1}\bigr) - k$.

Number of squarings: $\#\bigl\{i \in \{1, \ldots, k\} \mid w_i > 1\bigr\}$.

Evaluation Stage.

```
$A \leftarrow 1_G$
for $i = 1$ to $k$ do
  $window\_handle_i \leftarrow$ nil
for $j = b - 1$ down to $0$ do
  $A \leftarrow A^2$
  for $i = 1$ to $k$ do
    if $window\_handle_i =$ nil and $e_i[j] = 1$ then
      $J \leftarrow j - w_i + 1$
      while $e_i[J] = 0$ do
        $J \leftarrow J + 1$
      {now $j \geq J > j - w_i$ and $J \geq 0$}
      $window\_handle_i \leftarrow J$
      $E_i \leftarrow e_i[j \ldots J]$
    if $window\_handle_i = j$ then
      $A \leftarrow A \cdot g_i^{E_i}$ {multiply $A$ by table entry}
      $window\_handle_i \leftarrow$ nil
return $A$
```

Number of squarings: $b - \max_i w_i$ up to $b - 1$.

Expected number of multiplications:

$$\sum_{1 \leq i \leq k} b_i \cdot \frac{1}{w_i + \sum_{n \geq 1} \frac{1}{2^n}} = \sum_{1 \leq i \leq k} b_i \cdot \frac{1}{w_i + 1}.$$

3.2 wNAF-Based Interleaving Exponentiation Method

In some groups, elements can be inverted very efficiently so that division is not significantly more expensive than multiplication. (Inversion is cheap in case of elliptic curves or class groups of imaginary quadratic number fields, but not in $(\mathbb{Z}/n\mathbb{Z})^*$.) This can be exploited for making exponentiation algorithms more efficient by recoding the exponents into a signed representation. We use a technique introduced for single exponentiations independently in [18] and in [16] and apply it to the task of multi-exponentiation.

Given an exponent e_i and a window size w_i, we need a *width-*$(w_i + 1)$ *non-adjacent form* (*width-*$(w_i + 1)$ *NAF* or *wNAF*) of e_i, which is an array $N_i[b_i], \ldots, N_i[0]$ of integers such that

- each $N_i[j]$ is either 0 or odd with an absolute value less than 2^{w_i};
- $e_i = \sum_{0 \leq j \leq b_i} N_i[j] \cdot 2^j$;
- at most one of any $w_i + 1$ consecutive components of the array is non-zero.

A width-$(w_i + 1)$ NAFs always exists and is uniquely determined; it can be computed by the following algorithm [19]:

```
$c \leftarrow e_i$
$j \leftarrow 0$
while $c > 0$ do
  if $c[0] = 1$ then
    $u \leftarrow c[w_i \ldots 0]$
    if $u[w_i] = 1$ then
      $u \leftarrow u - 2^{w_i+1}$
    $c \leftarrow c - u$
  else
    $u \leftarrow 0$
  $N_i[j] \leftarrow u$; $j \leftarrow j + 1$
  $c \leftarrow c/2$
while $j \leq b_i$ do
  $N_i[j] \leftarrow 0$; $j \leftarrow j + 1$
return $N_i[b_i], \ldots, N_i[0]$
```

The maximum possible index for a non-zero component of the wNAF of a B-bit integer is B; i.e., the length of the wNAF without leading zeros may exceed the length of the binary expansion by one. The average density (proportion of non-zero components) in width-$(w_i + 1)$ NAFs is $1/(w_i + 2)$ for $B \to \infty$ [19].

Precomputation Stage. For $i = 1, \ldots, k$, precompute g_i^E for all odd E such that $1 \leq E \leq 2^{w_i} - 1$. (As inversion in G is assumed to be easy, this makes g_i^{-E} available as well.)

Number of non-trivial table entries (multiplications): $\big(\sum_{1 \leq i \leq k} 2^{w_i - 1}\big) - k$.

Number of squarings: $\#\big\{i \in \{1, \ldots, k\} \mid w_i > 1\big\}$.

Evaluation Stage.

```
$A \leftarrow 1_G$
for $i = 1$ to $k$ do
  $N_i[b], \ldots, N_i[b_i + 1] \leftarrow 0, \ldots, 0$
  $N_i[b_i], \ldots, N_i[0] \leftarrow$ width-$(w_i + 1)$ NAF of $e_i$
for $j = b$ down to 0 do
  $A \leftarrow A^2$
  for $i = 1$ to $k$ do
    if $N_i[j] \neq 0$ then
      $A \leftarrow A \cdot g_i^{N_i[j]}$ {multiply $A$ by [inverse of] table entry}
return $A$
```

Number of squarings: $b - \max_i w_i$ up to b.

Expected number of multiplications: $\sum_{1 \leq i \leq k} b_i \cdot \dfrac{1}{w_i + 2}$.

We can compare wNAFs with the sliding window technique of the basic interleaving exponentiation algorithm. Windows can be represented by components of an array as in the wNAF approach: In the algorithm description of section 3.1, E_i provides component values; array indexes are given by $\mathit{window\_handle}_i$. With the array filled in accordingly, we can use the same evaluation stage algorithm as in the wNAF-based method. The average density is $1/(w_i + 1)$ (each window covers w_i bits, and the number of additional zero bits between neighbouring windows is 1 on average). With wNAFs, the average density goes down to $1/(w_i + 2)$ for exactly the same precomputation. Thus using wNAFs effectively increases the window size by one.

4 Comparison of Simultaneous and Interleaving Exponentiation Methods

There is no general rule for selecting window sizes for the multi-exponentiation algorithms that we have looked at. Various factors have to be considered: First of all, absolute memory constraints can impose limits on possible window sizes. Second, even if a particular window size appears to minimise the total amount of computation for a multi-exponentiation, sometimes slightly smaller windows may improve the actual performance; this is because larger window sizes mean larger precomputed tables, i.e. possibly additional memory allocation overhead and less effective memory caching. Last but not least, implementations can use different representations for group elements during different stages of the multi-exponentiation: For instance, extra effort may be spent during the precomputation stage in order to obtain representations of precomputed elements that speed up multiplication with them in the evaluation stage (e.g. affine rather than projective representations of points on elliptic curves [8]).

These effects, however, do not mean that we cannot compare algorithms without looking at concrete cases: We can compare different aspects separately (table size, precomputation stage efficiency, evaluation stage efficiency) and look if an algorithm wins on all counts.

For the following comparisons, we assume that all maximum exponent lengths b_i are the same (an assumption that we made in section 2 on simultaneous exponentiation methods, but not in section 3 on interleaving exponentiation methods). As before, let b be the length of the largest of the exponents e_i.

In section 4.1, we compare the simultaneous 2^w-ary method with the basic interleaving method and show that the latter is usually more efficient for $k = 2$ if squarings are about as costly as multiplications. In section 4.2, we compare the simultaneous sliding window method with the wNAF-based interleaving method and show that the latter is more efficient for $k = 2$ and $k = 3$, assuming that computing and storing the wNAFs is not too costly. Section 4.3 briefly discusses the alternative multi-exponentiation method from [10] and shows that is obviated by our interleaving exponentiation methods. Finally, in section 4.4, we look at some concrete figures for the number of multiplications required by different methods for example values of k and b.

4.1 Comparison between the Simultaneous 2^w-Ary Method and the Basic Interleaving Method

While the simultaneous sliding window method is more efficient than the simultaneous 2^w-ary method, this section focuses on the latter. The reasons is that the 2^w-ary method is often used in practice (e.g. [6]), possibly because it is perceived to be simpler to implement. The basic interleaving exponentiation method is not too complicated (in particular, indexes into the precomputed table are easy handle), and as we will see, it is often more efficient than the simultaneous 2^w-ary exponentiation method. So when the intention is to avoid the simultaneous sliding window method, the basic interleaving method appears preferable for many applications.

Assume that, given k and b, a certain w turns out to provide optimal efficiency for the simultaneous 2^w-ary exponentiation method (section 2.1) when performed in a specific environment. Then the precomputation table requires $2^{kw} - 1 - k$ non-trivial entries, $2^{k(w-1)} - 1$ of which can be computed with one squaring each (while each of the remaining entries requires one general multiplication).

For the basic interleaving exponentiation method (section 3.1), we can use uniform window sizes $w_1 = \ldots = w_k = kw$. Then the precomputation table has $k2^{kw-1} - k$ non-trivial entries, each of which requires one general multiplication; also k additional squarings are needed (unless $k = w = 1$).

Thus in case $k = 2$, the table grows from $2^{2w} - 3$ to $2^{2w} - 2$ non-trivial entries, and instead of $2^{2w} - 3$ group operations of which $2^{2(w-1)} - 1$ are squarings, we need 2^{2w} group operations of which only 2 are squarings. If squarings are about as expensive as general multiplications, then for $k = 2$ the overall cost of precomputation is comparable for these two multi-exponentiation methods.

The number of squarings in the evaluation stage is always nearly b for both methods. The expected number of general multiplications in the evaluation stage is smaller for the interleaving method (except if $k = w = 1$, in which case both algorithms do exactly the same): Dividing the value for the basic interleaving exponentiation method by the value for the simultaneous 2^w-ary exponentiation method yields

$$\frac{k}{kw+1} \cdot \frac{w}{1 - \frac{1}{2^{kw}}} = \frac{kw}{kw+1} \cdot \frac{2^{kw}}{2^{kw} - 1},$$

and this is less than 1 for $kw > 1$ (the minimum is $64/75$ at $kw = 4$).

Note that using $w_1 = \ldots = w_k = kw$ is not necessarily an optimal choice of window sizes for the basic interleaving exponentiation method; using smaller or larger windows might lead to better performance. (Indeed, if we look just at the number of operations and ignore memory usage, then there is no reason why window sizes should depend on k.) While the above proof only covers the case $k = 2$, there are actually many other cases where the basic interleaving method is more efficient than the simultaneous 2^w-ary method, even if general multiplications are much more expensive than squaring; see table 1 in section 4.4. Also note that the precomputation effort grows exponentially in k in simultaneous methods, but not in interleaving methods.

4.2 Comparison between the Simultaneous Sliding Window Method and the wNAF-Based Interleaving Method

Similarly to section 4.1, assume that a certain w provides optimal efficiency for the simultaneous sliding window exponentiation method (section 2.2) for given k and b. In the following analysis, we require $k > 1$. The precomputation table has $2^{kw} - 2^{k(w-1)} - k$ non-trivial entries, each of which requires one general multiplication to compute. In addition to this, k squarings are required for precomputation unless $w = 1$.

For the wNAF-based interleaving exponentiation method (section 3.2), we can use window sizes $w_1 = \ldots = w_k = kw - 1$. This leads to a precomputation table with $k2^{kw-2} - k$ non-trivial entries, requiring one general multiplication each. In addition to this, we need k squarings unless $kw = 2$.

The difference between the number of non-trivial tables entries (and general multiplications) for these two methods is

$$(2^{kw} - 2^{k(w-1)} - k) - (k2^{kw-2} - k) = 2^{kw}\Big(1 - 2^{-k} - \frac{k}{4}\Big).$$

This is positive for $k \leq 3$ and negative for $k \geq 4$. Thus, with the w_i chosen like this, the precomputation stage of the wNAF-based interleaving exponentiation method is more efficient if $k = 2$ or $k = 3$ (except for the case $k = 3$, $w = 1$, where the wNAF-based interleaving exponentiation method saves one general multiplication, but requires three additional squarings).

The evaluation stage requires close to b squarings for both methods. The expected number of general multiplications is smaller for the wNAF-based interleaving method: $b/(w + 1/k)$ instead of $b/(w + \frac{1}{2^k-1})$.

The wNAF-based interleaving method with this choice of window sizes will often provide better performance than the simultaneous sliding window method for $k \geq 4$ as well: If additional memory allocation is not a problem, then the efficiency gain of the evaluation stage usually compensates for the growth of the precomputed table.

Similar to the situation in the preceding section, $w_1 = \ldots = w_k = kw - 1$ is not necessarily an optimal choice, and smaller or larger window sizes might be better (see section 4.4).

4.3 Comparison between the Dimitrov-Jullien-Miller Multi-Exponentiation Method and Interleaving Exponentiation

A multi-exponentiation method for the case $k = 2$ requiring two precomputed values (in addition to g_1 and g_2) if inverting is easy, or six precomputed values if inversions have to be done during the precomputation stage, was described by Dimitrov, Jullien, and Miller in [10]. This algorithm is related to the simultaneous sliding window exponentiation method of Yen, Laih, and Lenstra [21] (section 2.2 in the present paper), but uses signed recoding of exponents in order to reduce the size of the precomputed table. While the Yen-Laih-Lenstra method with a window size of 1 requires an expected number of $b \cdot 0.75$ general multiplications during the evaluation stage, the new method requires only about $b \cdot 0.534$ multiplications according to [10] (the number of squarings stays about the same). Yen-Laih-Lenstra with a window size of 2 needs only $b \cdot 3/7 \approx b \cdot 0.429$ multiplications (table 3 of [10] erroneously assumes a value of $b \cdot 0.625$), but has the disadvantage of requiring more precomputed elements, which may be a problem in some constrained environments.

We do not examine the algorithm of [10] in detail; note that it is outperformed by the wNAF-based interleaving method of section 3.2 with $w_1 = w_2 = 2$ if inversion is cheap (two precomputed values, $b \cdot 0.5$ multiplications) and by the basic interleaving method of section 3.1 with $w_1 = w_2 = 3$ otherwise (six precomputed values, $b \cdot 0.5$ multiplications).

4.4 Examples

As noted before, endless variations are possible for defining optimisation goals. In this section, we ignore memory usage and squarings and the issue of different element representations; we make comparisons based just on the expected number of general multiplications required by the various methods, precomputation and evaluation stage combined. (Window sizes are chosen such that this cost measure is minimised.) Note that the number of squarings is approximately the same for the simultaneous sliding window method, the basic interleaving method, and the wNAF-based interleaving method: No more than k squarings are needed in the precomputation stage, and close to b squarings are needed in the evaluation stage. The simultaneous 2^w-ary method requires $2^{k(w-1)} - 1$ squarings for precomputation and again close to b evaluation stage squarings; so ignoring the cost of squaring tends to favour this method.

Table 1 compares the number of general multiplications needed by these four methods for various k and b values. The entries for the most efficient methods in a particular configuration are printed in bold: For groups where inversion is easy so that the wNAF-based method can be used, it wins in all of these examples; for general groups, sometimes the simultaneous sliding window method and sometimes the basic interleaving method requires the least number of multiplications. (Remember that for $w = 1$ there is no difference between the simultaneous 2^w-ary method and the simultaneous sliding window method; for $w > 1$, the former is always less efficient.)

5 Multi-Exponentiation with Fixed Bases

When many multi-exponentiations use the same bases $g_1, \ldots, g_k$, it is sufficient to execute the precomputation stage just once, and we can try to make the evaluation stage more efficient by investing more work in precomputation. We cannot easily reduce the number of general multiplications in the evaluation stage, but we can reduce the number of squarings by using exponent splitting (cf. [5] and [9]) or the Lim-Lee "comb" method [14]. (Which approach is the

Table 1. Expected number of general multiplications for a multi-exponentiation $\prod_{1 \leq i \leq k} g_i^{e_i}$ with exponents up to b bits (c_1: simultaneous 2^w-ary method, c_2: simultaneous sliding window method, c_3: basic interleaving method, c_4: wNAF-based interleaving method)

k		$b = 160$	$b = 256$	$b = 512$	$b = 1024$	$b = 2048$
1	c_1	44.5 $(w=4)$	64.6 $(w=5)$	114.2 $(w=5)$	199.0 $(w=6)$	353.3 $(w=7)$
	c_2	**39.0** $(w=4)$	**57.7** $(w=5)$	**100.3** $(w=5)$	**177.3** $(w=6)$	**319.0** $(w=7)$
	c_3	**39.0** $(w_i=4)$	**57.7** $(w_i=5)$	**100.3** $(w_i=5)$	**177.3** $(w_i=6)$	**319.0** $(w_i=7)$
	c_4	**33.7** $(w_i=4)$	**49.7** $(w_i=4)$	**88.1** $(w_i=5)$	**159.0** $(w_i=6)$	**287.0** $(w_i=6)$
2	c_1	85.0 $(w=2)$	130.0 $(w=2)$	214.0 $(w=3)$	382.0 $(w=3)$	700.0 $(w=4)$
	c_2	78.6 $(w=2)$	119.7 $(w=2)$	**199.6** $(w=3)$	**353.2** $(w=3)$	660.4 $(w=3)$
	c_3	**78.0** $(w_i=4)$	**115.3** $(w_i=5)$	200.7 $(w_i=5)$	354.6 $(w_i=6)$	**638.0** $(w_i=7)$
	c_4	**67.3** $(w_i=4)$	**99.3** $(w_i=4)$	**176.3** $(w_i=5)$	**318.0** $(w_i=6)$	**574.0** $(w_i=6)$
3	c_1	131.8 $(w=2)$	179.0 $(w=2)$	305.0 $(w=2)$	557.0 $(w=2)$	1061.0 $(w=2)$
	c_2	127.7 $(w=2)$	**172.5** $(w=2)$	**291.9** $(w=2)$	**530.9** $(w=2)$	1008.7 $(w=2)$
	c_3	**117.0** $(w_i=4)$	173.0 $(w_i=5)$	301.0 $(w_i=5)$	531.9 $(w_i=6)$	**957.0** $(w_i=7)$
	c_4	**101.0** $(w_i=4)$	**149.0** $(w_i=4)$	**264.4** $(w_i=5)$	**477.0** $(w_i=6)$	**861.0** $(w_i=6)$
4	c_1	161.0 $(w=1)$	251.0 $(w=1)$	491.0 $(w=1)$	746.0 $(w=2)$	1256.0 $(w=2)$
	c_2	161.0 $(w=1)$	251.0 $(w=1)$	483.7 $(w=2)$	731.5 $(w=2)$	**1227.0** $(w=2)$
	c_3	**156.0** $(w_i=4)$	**230.7** $(w_i=5)$	**401.3** $(w_i=5)$	**709.1** $(w_i=6)$	1276.0 $(w_i=7)$
	c_4	**134.7** $(w_i=4)$	**198.7** $(w_i=4)$	**352.6** $(w_i=5)$	**636.0** $(w_i=6)$	**1148.0** $(w_i=6)$
5	c_1	**181.0** $(w=1)$	**274.0** $(w=1)$	522.0 $(w=1)$	1018.0 $(w=1)$	2010.0 $(w=1)$
	c_2	**181.0** $(w=1)$	**274.0** $(w=1)$	522.0 $(w=1)$	1018.0 $(w=1)$	1994.7 $(w=2)$
	c_3	195.0 $(w_i=4)$	288.3 $(w_i=5)$	**501.7** $(w_i=5)$	**886.4** $(w_i=6)$	**1595.0** $(w_i=7)$
	c_4	**168.3** $(w_i=4)$	**248.3** $(w_i=4)$	**440.7** $(w_i=5)$	**795.0** $(w_i=6)$	**1435.0** $(w_i=6)$
6	c_1	**214.5** $(w=1)$	**309.0** $(w=1)$	**561.0** $(w=1)$	1065.0 $(w=1)$	2073.0 $(w=1)$
	c_2	**214.5** $(w=1)$	**309.0** $(w=1)$	**561.0** $(w=1)$	1065.0 $(w=1)$	2073.0 $(w=1)$
	c_3	234.0 $(w_i=4)$	346.0 $(w_i=5)$	602.0 $(w_i=5)$	**1063.7** $(w_i=6)$	**1914.0** $(w_i=7)$
	c_4	**202.0** $(w_i=4)$	**298.0** $(w_i=4)$	**528.9** $(w_i=5)$	**954.0** $(w_i=6)$	**1722.0** $(w_i=6)$
7	c_1	278.8 $(w=1)$	**374.0** $(w=1)$	**628.0** $(w=1)$	**1136.0** $(w=1)$	**2152.0** $(w=1)$
	c_2	278.8 $(w=1)$	**374.0** $(w=1)$	**628.0** $(w=1)$	**1136.0** $(w=1)$	**2152.0** $(w=1)$
	c_3	**273.0** $(w_i=4)$	403.7 $(w_i=5)$	702.3 $(w_i=5)$	1241.0 $(w_i=6)$	2233.0 $(w_i=7)$
	c_4	**235.7** $(w_i=4)$	**347.7** $(w_i=4)$	**617.0** $(w_i=5)$	**1113.0** $(w_i=6)$	**2009.0** $(w_i=6)$
8	c_1	406.4 $(w=1)$	502.0 $(w=1)$	**757.0** $(w=1)$	**1267.0** $(w=1)$	**2287.0** $(w=1)$
	c_2	406.4 $(w=1)$	502.0 $(w=1)$	**757.0** $(w=1)$	**1267.0** $(w=1)$	**2287.0** $(w=1)$
	c_3	**312.0** $(w_i=4)$	**461.3** $(w_i=5)$	802.7 $(w_i=5)$	1418.3 $(w_i=6)$	2552.0 $(w_i=7)$
	c_4	**269.3** $(w_i=4)$	**397.3** $(w_i=4)$	**705.1** $(w_i=5)$	**1272.0** $(w_i=6)$	**2296.0** $(w_i=6)$
9	c_1	661.7 $(w=1)$	757.5 $(w=1)$	1013.0 $(w=1)$	**1524.0** $(w=1)$	**2546.0** $(w=1)$
	c_2	661.7 $(w=1)$	757.5 $(w=1)$	1013.0 $(w=1)$	**1524.0** $(w=1)$	**2546.0** $(w=1)$
	c_3	**351.0** $(w_i=4)$	**519.0** $(w_i=5)$	**903.0** $(w_i=5)$	1595.6 $(w_i=6)$	2871.0 $(w_i=7)$
	c_4	**303.0** $(w_i=4)$	**447.0** $(w_i=4)$	**793.3** $(w_i=5)$	**1431.0** $(w_i=6)$	**2583.0** $(w_i=6)$
10	c_1	1172.8 $(w=1)$	1268.8 $(w=1)$	1524.5 $(w=1)$	**2036.0** $(w=1)$	**3059.0** $(w=1)$
	c_2	1172.8 $(w=1)$	1268.8 $(w=1)$	1524.5 $(w=1)$	**2036.0** $(w=1)$	**3059.0** $(w=1)$
	c_3	**390.0** $(w_i=4)$	**576.7** $(w_i=5)$	**1003.3** $(w_i=5)$	1772.9 $(w_i=6)$	3190.0 $(w_i=7)$
	c_4	**336.7** $(w_i=4)$	**496.7** $(w_i=4)$	**881.4** $(w_i=5)$	**1590.0** $(w_i=6)$	**2870.0** $(w_i=6)$

most efficient depends on details of the situation such as exponent lengths, the permissible size of the precomputed table, the relative cost of squarings versus general multiplications, and whether the wNAF-based interleaving exponentiation method is applicable.)

Let m be an arbitrary positive integer. Assuming that fixed exponent length bounds b_i are known, we show how to evaluate power products $\prod_{1 \leq i \leq k} g_i^{e_i}$ in at most $m - 1$ evaluation stage squarings, using a precomputed table independent of the specific exponents e_i.

5.1 Exponent Splitting

Exponent splitting constructs a new power product representation by rewriting each factor as follows:

$$g_i^{e_i} = \prod_{0 \leq j < \lceil b_i/m \rceil} (g_i^{2^{jm}})^{e_i[jm+m-1 \ldots jm]}$$

This leads to power products consisting of $\sum_{1 \leq i \leq k} \lceil b_i/m \rceil$ factors. Any multi-exponentiation method can be used for evaluating these power products.

It is evident that for the multi-exponentiation methods described in this paper, exponent splitting does not help if k is already large and there are many large exponents. (In this case, instead of using precomputation table entries for additional bases, window sizes should be increased; then the evaluation stage will require more squarings, but fewer general multiplications than with exponent splitting.)

5.2 Lim-Lee Precomputation

To apply the Lim-Lee “comb” method, for every i we choose w_i such that $b_i \leq w_i m$ and precompute

$$G_i(S) := g_i^{\sum_{j \in S} 2^j}$$

for all subsets $S \subseteq \{0, m, 2m, \ldots, (w_i - 1)m\}$. Note that then every exponent up to b_i bits of length can be written as

$$e_i = \sum_{0 \leq j < m} N_i[j] \cdot 2^j$$

where each $N_i[j]$ is an integer of the form $\sum_{j \in S} 2^j$ with S as above. Thus we can use interleaving exponentiation with an evaluation stage algorithm similar to section 3.2, but with a reduced number of iterations. The $N_i[j]$ values for each iteration need not be stored in advance, they can be extracted from the e_i by tapping their bits in comb-shaped patterns; hence the nickname of this method.

A refinement of this (also from [14]) is based on the observation that the precomputed table can be reduced in size in exchange for additional evaluation stage multiplications: Partition $\{0, m, 2m, \ldots, (w_i - 1)m\}$ into v_i subsets

$T_{i,1}, \ldots, T_{i,v_i}$; now each of the above sets S can be written as $\bigcup_{1 \leq n \leq v_i} S_n$ with $S_n = S \cap T_{i,n}$, and then we have $G_i(S) = \prod_{1 \leq n \leq v_i} G_i(S_n)$. Thus it suffices to precompute $G_i(S_n)$ for all non-zero subsets $S_n \subseteq T_{i,n}$ for all n; from this precomputed data, $G_i(S)$ can be computed in at most $v_i - 1$ multiplications.

While Lim-Lee precomputation reduces the number of squarings, the expected number of general multiplications is larger than for the basic interleaving exponentiation method with a similarly sized precomputed table. (In the basic interleaving method, $2^{w_i - 1} - 1$ non-trivial precomputed values suffice to make sure that each evaluation stage multiplication covers w_i exponent bits, and we can skip many additional zero bits thanks to the sliding window. With Lim-Lee precomputation, we need at least $2^W - 2$ non-trivial precomputed values to be able to cover W exponent bits with each evaluation stage multiplication, and we lose the advantage of a sliding window.) Thus if k is large, using Lim-Lee precomputation is a disadvantage.

Note that it is possible to use Lim-Lee precomputation for some of the bases and standard precomputation (as in section 3) for others. This does not help for multi-exponentiation in these mixed cases, but precomputed data can then profitably be reused for pure Lim-Lee cases.

Without going into details, we remark that the Lim-Lee method can be considered an application of exponent splitting using specific multi-exponentiation algorithms suited for small exponents. For example, if $k = 1$, the simple Lim-Lee method uses "Shamir's trick", i.e. simultaneous exponentiation with a window size of 1. Further algorithmic variations are possible.

6 Conclusion

In many cases, the basic interleaving exponentiation method compares favourably to the simultaneous 2^w-ary method, in particular if $k = 2$ and squarings are about as costly as general multiplications. In groups where inverting elements is easy, the wNAF-based interleaving exponentiation method is available; its efficiency is superior even to the sliding window variant of simultaneous exponentiation both in the precomputation stage and the evaluation stage if $k = 2$ or $k = 3$, and it is usually more efficient for larger k as well. In all cases, interleaving exponentiation provides the following advantages over simultaneous exponentiation:

- Improved efficiency if the bit-lengths of the exponents e_i differ significantly.
- More flexibility in choice of the size of the auxiliary table (and, hence, the time spent on precomputation), particularly if k is large.
- Better handling of situations where one or more of the g_i are fixed while others are variable between multiple multi-exponentiation: A corresponding part of the precomputation has to be done only once. (This is the case in DSA, ECDSA, and RDSA signature verification if multiple signatures are verified that are based on the same underlying parameters.)

Thus, depending on circumstances, either the simultaneous sliding window method or one of the interleaving exponentiation methods may be advantageous.

It is easy to implement interleaving exponentiation for variable k. As the the special case $k = 1$ of the basic and wNAF-based interleaving exponentiation methods yields the usual sliding windows exponentiation method and wNAF-based exponentiation method, respectively, this makes it unnecessary to implement these separately.

References

1. American National Standards Institute (ANSI). Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA). ANSI X9.62, 1998.
2. Biehl, I., Buchmann, J., Hamdy, S., and Meyer, A. A signature scheme based on the intractability of extracting roots. *Designs, Codes and Cryptography*. To appear.
3. Bos, J., and Coster, M. Addition chain heuristics. In *Advances in Cryptology – CRYPTO '89* (1989), G. Brassard, Ed., vol. 435 of *Lecture Notes in Computer Science*, pp. 400–407.
4. Brands, S. *Rethinking Public Key Infrastructures and Digital Certificates – Building in Privacy.* MIT Press, 2000.
5. Brickell, Gordon, McCurley, and Wilson. Fast exponentiation with precomputation. In *Advances in Cryptology – EUROCRYPT '92* (1993), R. A. Rueppel, Ed., vol. 658 of *Lecture Notes in Computer Science*, pp. 200–207.
6. Brown, M., Hankerson, D., López, J., and Menezes, A. Software implementation of the NIST elliptic curves over prime fields. In *Progress in Cryptology – CT-RSA 2001* (2001), D. Naccache, Ed., vol. 2020 of *Lecture Notes in Computer Science*, pp. 250–265.
7. Buchmann, J., and Hamdy, S. A survey on IQ cryptography. In *Proceedings of Public Key Cryptography and Computational Number Theory, 2000.* To appear. Preprint available at `http://www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/TR/`.
8. Cohen, H., Ono, T., and Miyaji, A. Efficient elliptic curve exponentiation using mixed coordinates. In *Advances in Cryptology – ASIACRYPT '98* (1998), K. Ohta and D. Pei, Eds., vol. 1514 of *Lecture Notes in Computer Science*, pp. 51–65.
9. de Rooij, P. Efficient exponentiation using precomputation and vector addition chains. In *Advances in Cryptology – EUROCRYPT '94* (1995), T. Helleseth, Ed., vol. 950 of *Lecture Notes in Computer Science*, pp. 389–399.
10. Dimitrov, V. S., Jullien, G. A., and Miller, W. C. Complexity and fast algorithms for multiexponentiation. *IEEE Transactions on Computers 49* (2000), 141–147.
11. ElGamal, T. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory 31* (1985), 469–472.
12. Kocher, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology – CRYPTO '96* (1996), N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, pp. 104–113.
13. Kocher, P. C., Jaffe, J., and Jun, B. Differential power analysis. In *Advances in Cryptology – CRYPTO '99* (1999), M. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, pp. 388–397.
14. Lim, C. H., and Lee, P. J. More flexible exponentiation with precomputation. In *Advances in Cryptology – CRYPTO '94* (1994), Y. G. Desmedt, Ed., vol. 839 of *Lecture Notes in Computer Science*, pp. 95–107.

15. Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. *Handbook of Applied Cryptography.* CRC Press, 1997.

16. Miyaji, A., Ono, T., and Cohen, H. Efficient elliptic curve exponentiation. In *International Conference on Information and Communications Security – ICICS '97* (1997), Y. Han, T. Okamoto, and S. Qing, Eds., vol. 1334 of *Lecture Notes in Computer Science*, pp. 282–290.

17. National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). FIPS PUB 186-2, 2000.

18. Solinas, J. A. An improved algorithm for arithmetic on a family of elliptic curves. In *Advances in Cryptology – CRYPTO '97* (1997), B. S. Kaliski, Jr., Ed., vol. 1294 of *Lecture Notes in Computer Science*, pp. 357–371.

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

20. Straus, E. G. Problems and solutions: Addition chains of vectors. *American Mathematical Monthly 71* (1964), 806–808.

21. Yen, S.-M., Laih, C.-S., and Lenstra, A. K. Multi-exponentiation. *IEE Proceedings – Computers and Digital Techiques 141* (1994), 325–326.