Right-to-Left or Left-to-Right Exponentiation?

Colin D. Walter
Information Security Group, Royal Holloway, University of London
Colin.Walter@rhul.ac.uk

Abstract. The most recent left-to-right and right-to-left multibase exponentiation methods are compared for elliptic curve and modular residue groups to gauge the value and cost of switching from the normal left-to-right mode to the more side channel resistant right-to-left direction in a resource constrained environment.

Key Words. Scalar multiplication, exponentiation, addition chain.

1 Introduction

P.-A. Fouque and F. Valette [2] and others have suggested that right-to-left (R2L) exponentiation is more resistant to side channel analysis than the more usual left-to-right (L2R) processing. Although some attacks work equally well in either direction, these usually make the assumption of known or chosen input [3], and are defeated by the straightforward countermeasure of message blinding. Other attacks without this assumption tend to work more successfully in the left-to-right direction [8]. So the purpose here is to compare the two directions in terms of speed and area over their most compact implementations using current state-of-the-art exponentiation methods to see if there is a safer R2L method which is as efficient as L2R methods.

For the L2R direction pre-processing is used to create a table of digit powers $M^d$ of the input message $M$ which are then accessed whenever the left-to-right algorithm encounters a non-zero digit $d$. Weakness arises because, when the table element is fetched or used, there is a danger that enough of its Hamming weight, address or other identifying information will leak for an adversary to determine which exponent digits are equal, and thereby recover the secret key [8].

As the dual to the more standard L2R algorithm, the R2L $m$-ary method of Yao [10, 4] also has one register associated with each non-zero digit. These are updated whenever the corresponding digit is encountered in the base $m$ representation of the exponent. Specifically, for the $i$th digit $d_i$, the register for $d_i$ is multiplied by $M^{m^i}$ on the $i$th iteration where $M$ is the input message. Post-processing is used to multiply together the $d$th powers of the contents of the registers for $d$. Since the same operands are no longer used repeatedly, R2L is less subject to a side channel attack using Hamming weights. (We ignore the problem of executing different code for zero digits which is common to both directions. It can be avoided by minor, judicious recoding and the cost of another register [4].)
In resource constrained environments there is little space for either a pre-computed table or a register for each digit. Since $m$-ary exponentiation has an Area$\times$Time complexity of about $O(m)$ registers by $O(1+\frac{m-1}{m}\log_m2)$ multiplicative operations per exponent bit, it becomes increasingly poor value for money as $m$ grows, and gives an insignificant increase in speed except for the smallest values of $m$. Hence implementations with the fewest number of registers are the target of study here, and attention is effectively restricted to tables with just one element.

We assume key $k$ and message $M$ (i.e. exponent and base) are sufficiently fresh every time so that no useful pre-processing can be done in advance of constructing the usual table of digit powers of the message. The smallest implementation of L2R turns out to require marginally more area than R2L for comparable multibase algorithms. Moreover, L2R only becomes faster when given more storage or carefully chosen mixed coordinate representations. This means that improved side channel resilience and efficiency should be obtained for ECC-type scalar multiplication in the most compact implementations by working right to left. For RSA-type contexts, 4-ary exponentiation is altogether more efficient whichever direction is chosen, but the increase in security afforded by R2L has the extra time penalty of not being able to apply more efficient combined double and add formulae. The methods are straightforward, and so mean little extra development time and almost no extra code.

2 Multibase Exponentiation

There has been recent research to develop efficient composite double-and-add, triple-and-add etc. operations for elliptic curve cryptography, e.g. [6]. These can be used when the exponent $k$ has a multi-base representation of the form

$$k = (((d_{n-1}r_{n-2} + d_{n-2})r_{n-3} + ... + d_2)r_1 + d_1)r_0 + d_0$$ (1)

[1, 7] where the bases (or radixes) $r_i$ belong to some small set $S$ such as $\{2,3\}$ or $\{2,3,5\}$ and the digits $d_i$ are integers satisfying constraints which depend on the properties required of the recoding. This representation appears in the L2R method of Longa and Gebotys [5] with $d_i = 0$ if $r_i \neq 2$ and odd or zero digits $d_i$ for $r_i = 2$ such that $|d_i| < 2T$ for the table size $T$. The representation also appears in the R2L method of Walter [9] where the digits typically satisfy the usual $0 \leq d_i < r_i$ but bases $r_i$ are randomly chosen.

Such representations are generated very easily using the normal change-of-base algorithm, as follows. Suppose $k_i$ is the value of $k$ remaining after generating the digits $d_0, d_1, ..., d_{i-1}$. The iterative step starts by selecting base $r_i$ using some suitable criterion such as divisibility of $k_i$ by $r_i$. Then digit $d_i$ is chosen from the residue class $k_i \mod r_i$ using a suitable criterion, such as that of least absolute value or that of least non-negative value. Obviously the next step repeats this process on $k_{i+1} = (k_i - d_i)/r_i$.

A close to optimal choice for the representation is described by Longa et al. [5]. Usually the best decision is to select a radix $r$ for which the digit is 0 since it
saves a multiplication, and otherwise take $r = 2$. Within the restrictions imposed by availability of digits, the same re-coding algorithm was used for the L2R and R2L results here and it was close to that of [5].

For R2L exponentiation we must consider a more compact algorithm than Yau’s in which there are only two quantities being updated: the cumulative product $P$ and a digit-independent power $M_i = M^{R_i}$ for the radix product $R_i = \prod_{j=0}^{i-1} r_j$. The iterative step is to form $M_{i+1} = M_i^{r_i}$, calculating $M_i^{d_i}$ en route from $M_i$ and multiplying $M_i^{d_i}$ onto $P$ when it becomes available. To avoid extra multiplications, digits $d_i$ are only allowed if the associated powers appear naturally when raising to the power $r_i$. As noted above, for L2R exponentiation the iterative step is to take the current cumulative product $P$, raise it to the power $r_i$, and multiply it by the entry for $d_i$ in a precomputed table containing the digit powers $M^{d_i}$ of the message $M$. If $r_i = m$ for all $i$, this is just $m$-ary exponentiation. However, to achieve similar compactness to that of R2L only two registers can be used. This limits the table size to 1 and hence restricts the choice of digits. Briefly, ignoring pre- and post-processing the algorithms are:

2.1 Time and Space

For both these methods the core of the iterative step has essentially the same time: the number of group operations required to raise an element to the power $r_i$ plus one group operation when the digit is non-zero. Even if we differentiate between squaring and non-squaring operations, this cost is still the same for the two directions. However, forming the digit powers may have different costs in the two directions. This section is mostly a discussion of this problem.

For $d_i \in \{0, 1\}$ the cost of forming the digit power is free. With L2R, the cost is also zero for digits represented in the table such as $M_1$, but other digits can be generated at greater cost. An extra register is needed for raising to the power 3 or 5. On completion of raising to the power $r_i$ this register becomes available to compute $M_i^{d_i}$ from table entries if it is needed but was not in the table. If the table only contains $M_1$, the L2R algorithm for doing this can also be used to determine $M_i^{d_i}$ from $M_i$ in the R2L case and multiply it into $P$ before starting to construct $M_i^{r_i}$. Thus the iterative R2L step is no more expensive than that in the L2R case for a table containing just $M$.

In the R2L direction the addition chains $1+1 = 2$, $1+2 = 3$ for base 3 and $1+1 = 2$, $1+2 = 3$, $2+3 = 5$ or $1+1 = 2$, $2+2 = 4$, $1+4 = 5$ for base 5 show that any digit power $M_i^{d_i}$ can be constructed en route to $M_i^{r_i}$ and therefore all are free when $0 \leq d_i < r_i$ and $r_i \in \{3, 5\}$. Note that there should be no extra space (or time) penalty for multiplying $P$ by $M_i^{d_i}$ during the construction of $M_i^{r_i}$ rather

---

1 We consider $-d$ as represented by the table if $d$ is present and inversion is free in the group in which the exponentiation is performed.
than after its construction (which is when the L2R digit-related multiplication is done) because it requires no extra resources beyond what are already required for the next multiplication in forming $M_i^{r_i}$. This makes the non-trivial digits for an odd base cheaper for R2L than L2R if the table has just one entry. When inversion is possible, the negative digits are also available allowing a choice of two digits for bases 3 and 5. This always enables the next value of $k_i$ to be made a multiple of 2 in the R2L case – saving a multiplication. However, in the L2R case there are insufficient digits in the small table for this to be possible for any non-zero residue class of odd $r_i$.

Several consecutive choices of base 2 yields the equivalent of a 2-power value for $r_i$ but exponentiation by a 2-power requires one less register than exponentiations by 3 or 5. Hence there is an additional register available for the construction or storage of further digit powers during powering by such $r_i$. For L2R the value in $P$ is of no use in constructing this digit power (except perhaps initially), so there is only space to perform the usual binary square and multiply using the other two registers – in which case there is no advantage in combining the consecutive bases 2: digits 0 and 1 (and $-1$, if available) with base 2 will be just as efficient. The exception is if a non-trivial digit, such as 3, were to recur in a continuous long sequence of bases 2. Then $M^3$ can be retained for re-use, but this is unlikely to make more than a negligible difference unless 4-ary (or 8-ary) exponentiation is at least as efficient (see below). However, in the R2L case of a 2-power base, we can simply switch to Yau’s algorithm to achieve the same digit power. Indeed, for addition chains which require two registers for storage, there is an easy duality between L2R cases which must preserve $M$ and R2L cases which are only allowed to square $M_i$. This duality includes all the L2R possibilities, but with R2L we are not forced into only squaring $M_i$ since a 2-power base $r_i$ iteration only needs to finish with the 2-powering of $M_i$. Thus, for example, the addition chain $1, 2, 3, 5, 8$ could be employed for R2L, but it has no equivalent for L2R. Hence, for a number of consecutive with base 2, L2R can gain over R2L by preserving an extra digit power whereas R2L has extra addition chains available to it, some of which may provide extra efficiency.

As noted earlier, L2R with base 3 and/or 5 required at least two registers in addition to those for table entries. By allowing only the base 2, i.e. $B = \{2\}$, the second register could be re-assigned to hold a further table entry permanently. The L2R version equivalent to the most compact R2L version will therefore have two table entries, namely $M^1$ and (most probably) $M^3$. This provides 4-ary or 8-ary exponentiation (depending on whether inversion in the multiplicative group is free of not) which, as noted in §1, takes one squaring plus $O(\frac{1}{4} \times \frac{1}{2})$ or $O(\frac{1}{8} \times \frac{1}{4})$ multiplications per exponent bit. From the simulation results in the top two lines of Table 1, this turns out to be faster than the multibase algorithm for RSA-type situations where inversion is very expensive, but it is slower in typical ECC applications than the mixed base algorithms where inversion is relatively free. Thus resource-constrained contexts without cheap inversion should choose 4-ary exponentiation whatever the direction.
Overall this suggests a potential speed advantage for the R2L direction when the L2R table contains only the single element $M$. However, the pre-computed L2R table entries can also be manipulated into special forms (such as affine rather than projective coordinates) which may make all the multiplications by the digit powers cheaper in the L2R direction than the R2L direction [6, 5]. Here “cheaper” may mean less space (e.g. two coordinates instead of three) and/or faster (e.g. from specialised mixed coordinate operations). Indeed, one might save up to 5 of 16 field operations in the point additions (Jacobian coords) [5].

Regarding the choice of $B$, bases which are not powers of 2 are inherently less efficient to use and only make much difference to efficiency if small, such as 3 and 5, and when associated with the digit 0. Larger bases are only of use in restricted circumstances, such as repeated use of the same key when the cost of recoding $k$ can be amortised over the life of the key. Due to space restrictions, we only give numerical results here for $B = \{2, 3\}$. However, 5 and 9 are useful, the latter replacing two instances of 3 when squaring is cheaper than multiplication since raising to the power 9 can be done with 3 squarings and a multiplication instead of the 2 squarings and 2 multiplications of two cubings.

### 2.2 Recoding Costs

Space may also be required for storing the exponent $k$ in its double (or triple) base representation. As this is a redundant representation (i.e. not unique) it requires more space than the binary representation. The precise space depends on the recoding, but a typical choice might be the digit set $\{0, \pm 1, \pm 3\}$ for base 2 and only digit 0 for bases 3 and 5. This provides 8 possibilities, requiring 3 bits storage per digit of the representation. Since 2 is by far the most commonly selected base, this means the recoding would effectively occupy three times the space of $k$ – the space of one elliptic curve point in projective coordinates, or three table entries for RSA or Diffie-Hellman applications.

In many protocols the exponent is random, and so could be generated on-the-fly already in multibase form. In this case the space required by a recoding of $k$ can be ignored. But suppose the binary form is already given. Then the multibase representation must be generated from right to left. This is the order the digits are consumed by the R2L method. So no additional storage space is required beyond the working space which is common to both directions. However, for the standard L2R direction the whole multibase representation needs to be generated in advance so that space is required to store it.

The computational effort of recoding is clearly the same for both directions. If we let $\pi$ be a product of powers of the base elements in $B$ which is close to the maximum value of a machine word, then several multibase digits can be decided at each iterative step using the value of $k_i \mod \pi$. If $\pi$ contains the bases with multiplicities close to those in the multibase representation then we can expect to maximise the number of digit decisions before the next division of $k_i$ by the product of the associated bases. (That division yields $k_i + \ldots \mod \pi$ for deciding the next batch of digits.) Hence the total number of word operations should be the same as that for two or three multiplications of integers the size of $k$. 


although those word operations are slower divisions rather than products. Apart from the cost of the exponentiation itself, this is the main time penalty for using a multibase representation rather than a purely base 2 form.

3 Simulation Results

In the L2R case an algorithm similar to that of Longa [5] was selected for base set \( B = \{2, 3\} \) to provide close to optimal recodings\(^2\) for several likely ratios of the cost \( Sq/Mu \) of squarings compared with multiplications in the group and using the minimal table of one element. From the above discussion this algorithm for L2R already requires more space than the R2L direction because of having to store the recoded \( k \). The algorithm used only one table entry, namely \( M \), but did not allow a second table entry to be retained between repeated use of base 2, although this would be possible. Equivalent algorithms were selected for R2L recoding, but replacing pre-computed table entries with digit entries that are computed on the fly.

The table below lists speeds in both directions for cases where inversion (\( INV \)) is free (e.g. ECDSA), which allows \(-d\) whenever \( d \) is possible, and where inversion is prohibitive (e.g. RSA), which makes repeated use of negative digits infeasible. The multiplications column (\( \#Mults \)) is the number of squarings scaled by the ratio \( Sq/Mu \) plus the number of non-squaring operations. There are no differential initialisation costs to take into account since the table size is 1. (No pre-computations.) Increased access to non-trivial digits for all base choices in the R2L direction results in consistently faster speeds than for L2R if efficiency is measured purely in terms of multiplication counts.

<table>
<thead>
<tr>
<th>Sq/Mu</th>
<th>Key Length</th>
<th>Dir</th>
<th>Inv</th>
<th>#Mults</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>160</td>
<td>L2R</td>
<td>Y</td>
<td>203.9</td>
</tr>
<tr>
<td>1.0</td>
<td>160</td>
<td>R2L</td>
<td>Y</td>
<td>190.5</td>
</tr>
<tr>
<td>0.8</td>
<td>160</td>
<td>L2R</td>
<td>Y</td>
<td>173.4</td>
</tr>
<tr>
<td>0.8</td>
<td>160</td>
<td>R2L</td>
<td>Y</td>
<td>159.6</td>
</tr>
<tr>
<td>0.5</td>
<td>160</td>
<td>L2R</td>
<td>Y</td>
<td>126.7</td>
</tr>
<tr>
<td>0.5</td>
<td>160</td>
<td>R2L</td>
<td>Y</td>
<td>113.2</td>
</tr>
<tr>
<td>1.0</td>
<td>1024</td>
<td>L2R</td>
<td>N</td>
<td>1415.2</td>
</tr>
<tr>
<td>1.0</td>
<td>1024</td>
<td>R2L</td>
<td>N</td>
<td>1388.7</td>
</tr>
<tr>
<td>0.8</td>
<td>1024</td>
<td>L2R</td>
<td>N</td>
<td>1220.6</td>
</tr>
<tr>
<td>0.8</td>
<td>1024</td>
<td>R2L</td>
<td>N</td>
<td>1196.0</td>
</tr>
<tr>
<td>0.5</td>
<td>1024</td>
<td>L2R</td>
<td>N</td>
<td>936.7</td>
</tr>
<tr>
<td>0.5</td>
<td>1024</td>
<td>R2L</td>
<td>N</td>
<td>907.2</td>
</tr>
</tbody>
</table>

As noted above, 4-ary L2R exponentiation with a table containing \( M^1 \) and \( M^3 \) is faster than 2,3-ary exponentiation where inversion is expensive (see the right hand columns). Inclusion of more bases can make the multibase exponentiation faster [7] but the recoding then becomes too expensive for a cryptographic token.

\(^2\) Finding an optimal recoding is an NP-hard problem.
4 Conclusion

The ability to generate digit powers cheaply often enables right-to-left exponentiation to be performed significantly faster and in less space than left-to-right methods using the same but restricted space. The speed difference is minimally in favour of right to left when using 4-ary exponentiation for RSA and DH applications for which inversion is relatively expensive but for ECC applications with cheap inversion a multibase method is faster. Then the left-to-right algorithm is clearly slower and, moreover, more space is required because the recoded exponent must be stored. The only compact implementations for which left-to-right exponentiation is faster seem to require special combined square and multiply (or cube and multiply, etc.) operations which are faster than applying the individual operations separately. These occur for some choices of elliptic curve coordinates. Overall, the two directions have very comparable speeds for comparable compact cases. Hence it makes sense to consider the apparently more secure right to left direction when side channel leakage is likely to be a problem.

References