Abstract—Compact exponentiation algorithms are required for resource constrained cryptographic devices. Historically, binary windowing algorithms have been among the best, and only they are possible in the most restricted cases. However, with minimally more area, mixed base representations of the exponent give rise to algorithms with competitive speeds. Implementations of them using the same area are compared for both RSA and elliptic curve applications, and when processing the exponent digits in either direction. Walter's space-preserving duality (CT-RSA 2012, [34]) simplifies the comparison to considering one direction plus secondary space and time issues that distinguish the two directions. The duality led to a new, compact left-to-right exponentiation method derived from the right-to-left division chain method (Arith 13, [28]). Here, simulation results for this method show it to be faster in constrained environments than the traditional table-based algorithms for mixed base representations. Overall, the classical binary algorithms are faster for RSA, but the new mixed base algorithm is faster for the elliptic curves under the chosen memory restrictions.

Keywords—Scalar multiplication, multi-base representation, mixed base representation, exponentiation, dual addition chain, division chain.

Compact Exponentiation Schemes

Colin D. Walter, *Senior Member, IEEE*

1 Introduction

It is always useful to have machinery to generate new algorithms and to have a wide selection of algorithms from which to choose when there are so many different optimisations and conflicting pressures on the target computational platform. Cryptographic groups often have some operations which are cheaper than the basic group operation but, as well as space and time pressures, embedded devices may have to achieve some measure of side channel resistance. Here the main implementation choices available for these are discussed in relation to computing platforms with essentially minimal space resources. The simplest exponentiation method, binary square-and-multiply, requires just two locations for storing elements of the group in which the exponentiation is performed. This investigation looks at the case of having three locations. A new duality [34] between left-to-right and right-to-left exponentiation methods is used to derive some novel ways of processing the recoded exponents and hence speeding up and/or randomising the computation.

There are few comparative treatments of exponentiation using different schemes. Joye [14] makes a comparison of left-to-right (L2R) and right-to-left (R2L) methods but in general there is rarely more detail for an individual algorithm than the total number of group operations, perhaps split into the number of squaring and non-squaring operations, or a count of the underlying field squarings and multiplications. However, in addition to squaring being cheaper than a general group multiplication, many groups have other favoured cheap operations: inversion, the Frobenius endomorphism and composite multiple-and-add operations on an elliptic curve are examples. There is a general framework in which all of these operations can be used effectively with the greatest efficiency, and which allows the use of all the usual exponentiation algorithms, namely that of a *mixed base* representation. These are derived from a division chain [28], for which recoding choices are made to satisfy the various efficiency and other requirements for the target platform.

In most circumstances processing in one or other direction will have useful time, space or side channel resistance advantages. Usually the standard left-to-right table-based m-ary or sliding window exponentiation method is used. However, left to right processing of recodings with a fixed radix can be attacked through side channel leakage because use of the table entries may reveal the values of the corresponding digits in the representation of the secret key [29]. This may be possible even when the key is used only once and under the assumption that input messages to the exponentiation are blinded and therefore unknown to the attacker. Although a table based method may have side channel disadvantages, the table values might be put into special forms which make group operations significantly faster. Among these possibilities, are the recently developed double-and-add, triple-and-add and quintuple-and-add composite elliptic curve operations [4], [6], [9], [12], [23], but other fast operations benefit the opposite right-to-left direction [27].

The original purpose here had just been to compare the two directions in terms of speed and area over their most compact implementations. It was already known that algorithms could be translated from one direction to the other using the *transposition* method [2], [18] which preserves "time", i.e. there is the same count of squarings and multiplications in either direction when the method is applied carefully[1], but the method does not treat space. However, during the research, it turned out to be possible to modify the method to treat space considerations as well. The resulting new duality described in [34] shows the two directions require essentially the same time *and* space, with differences arising only at a secondary level of detail. The main such differences are catalogued here, as they have a modest effect on the most compact cases. In particular, as well as certain operations being faster in one direction than the other, L2R exponentiation may suffer a greater space overhead for storing the recoded exponent than R2L, whereas the R2L version has more reading and writing to memory than L2R.

The new duality also gives rise to a new L2R algorithm derived from Walter's R2L division chain recoding technique [28]. This is investigated in more detail than its brief mention in [34]. Classical, binary-oriented algorithms are generally better for residue rings, as in RSA, but, where there is a cheap inverse – as in elliptic curve cryptography – this compact mixed base L2R algorithm is faster. The overhead is small since multi-base methods

Information Security Group, Royal Holloway, University of London (retired)

1. Knuth's example of 70 ([18], §4.6.3) does not even achieve this property.

are straight-forward, with little extra development time and almost no extra code. An added benefit is that a randomised recoding to a mixed base is an efficient way of imparting some side channel resistance to the exponentiation [30], [31]. Moreover, with its much greater flexibility in a wider range of contexts than classical methods, the new algorithm has the potential of generating new record speeds for resource constrained devices.

2 Notation & Mixed Bases Examples

Here a fresh, i.e. unseen, exponent D of cryptographic size is assumed for every exponentiation, or at least a fresh, randomly different scheme is required for the exponentiation in order to avoid side channel analysis. If D were used repeatedly and side channel leakage were not a problem, time could be spent searching for an efficient addition chain, with the cost of this amortized over all instances of its use. For the single instances here it is not possible to do this and still retain the requirement for time efficiency. For a multiplicative group G, we wish to compute g^D for some fresh, random element $g \in G$. Thus, it is not possible to pre-compute and store any powers of g and spread the cost over a number of exponentiations using different values of D. This would, moreover, contradict the desire for space efficiency.

To construct the addition chain used to express an exponentiation scheme for g^D, it appears that one is forced to break D iteratively into two parts, the smaller of which contributes to the addition chain, and the larger of which is fed into the next iteration. Addition sub-chains corresponding to the smaller parts are then concatenated into an addition chain for D. Specifically, in all known schemes, this appears to require D to be broken into either i) $D \bmod r$ and $\lfloor D/r \rfloor$ where r is selected from a limited set of non-zero integers which are close to 0, or ii) $D-R$ and R where R is selected from a limited set of integers with finitely expressible relevant properties, at least one of which is close to D. The former is the approach taken by Walter [28] with the *mixed base representation*

$$D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots+d_1)r_0+d_0 \qquad (1)$$

The latter is the approach pursued by Dimitrov [8] in his *multi-base representation*

$$D = \sum_{i=0}^{n-1} d_i R_i \ . \qquad (2)$$

In these representations $d_i \in \mathcal{D}$ for some pre-defined digit set $\mathcal{D}$, $r_i \in \mathcal{R}$ for some pre-defined radix set $\mathcal{R}$ of non-zero positive integers, and $R_i \in \mathcal{R}^*$ is a product of elements from $\mathcal{R}$. The case of $\mathcal{R} = \{2\}$, $\mathcal{D} = \{0, 1\}$ yields the normal binary representation used in the standard square-and-multiply exponentiation schemes.

Knuth [18] and Gordon [11] provide coverage of the most important classical scalable exponentiation methods, and all relevant algorithms follow the mixed base pattern (1) above. As well as the usual binary square-and-multiply method (processing exponent bits in either direction), there are its generalisations to the left-to-right table-based m-ary method of Brauer [3] and its dual, namely the right-to-left m-ary method of Yao [36]. These arise by taking $\mathcal{R} = \{m\}$ and $\mathcal{D}$ any set which contains at least a complete set of residues modulo m, such as, typically, $\mathcal{D} = \{0, 1, \ldots, m-1\}$ or $\mathcal{D} = \{\lceil -(m-1)/2 \rceil, \ldots, -1, 0, 1, \ldots, \lceil (m-1)/2 \rceil\}$. The sliding window versions of both these algorithms [19] correspond to $\mathcal{R} = \{2, m\}$ where $m = 2^w$ is a 2-power, and non-zero odd digits being removed from $\mathcal{D}$. There is also the non-adjacent form (NAF) recoding [16], [1] which yields two algorithms, one to process the digits in each of the two directions. It uses an extended digit set to allow the redundancy which enables any pair of consecutive digits to include 0. The tableless R2L method of Walter [28] takes, *inter alia*, $\mathcal{R} = \{2, 3, 5, 17, 33, 49, 65, 97, 129, 257, 513, 1025, \ldots\}$, whereas that of Dimitrov [7] uses, *inter alia*, $\mathcal{R} = \{2, 3, \ldots\}$ with $d_i = 0$ if $r_i \neq 2$. Gebotys & Longa [21] deliberately select the mixed base representations of [7] in order to make use of composite elliptic curve operations which are cheaper than the sum of the costs of their constituent group operations. Dimitrov *et al.* [9], [23] previously used such operations to provide speed-ups for the multi-base representation (2). Such applications are among those of greatest pertinence here.

There are specialised mixed base recodings for improving side-channel resistance. Some simply reduce data-dependent variation by performing a multiplication at every conditional step in the loop body of algorithms such as those in Figs. 1 and 2. This may be a dummy multiplication whose result is discarded when $d_i = 0$. It becomes part of the exponentiation as a result of how the base/digit pair (r_i, d_i) is translated into an addition sub-chain. Alternatively, a digit set $\mathcal{D}$ may be chosen which does not contain 0. Other counter-measures include a random input to vary the recoding unpredictably. This may allow limited re-use of the same exponent. Most random recodings use a fixed power of 2 as the base; some have a variable power of 2. They include those of Liardet & Smart [20], Oswald & Aigner [26] and Itoh *et al.* [13], among many. Those using a radix set containing more than powers of 2 are restricted to the MIST algorithm [30] where the choice of bases is random, typically with $\mathcal{R} = \{2, 3\}$ or $\mathcal{R} = \{2, 3, 5\}$. In such cases, (1) is called a *randomary* representation.

3 Recoding to a Mixed Base or Multi-Base Representation

Exponentiation is a two stage process. In the first stage, a recoding is used to convert the exponent into an addition chain [18] which provides rules for constructing powers of the input in order to obtain the required output. In the second, which may be executed in parallel with the first, those addition chain rules are applied to the

given group element to raise it to the required exponent. With the space constraints here, the addition chain must be annotated with register details which show how to perform the computation in the space available.

Derivation of a mixed base form (1) is straight-forward using the usual change-of-base algorithm with the (possibly variable) base r_i to generate the digits d_i from least to most significant. Suppose D_i is the value of D remaining after generating the digits $d_0, d_1, ..., d_{i-1}$. The iterative step starts by using a simple finite automaton to select a base $r_i \in \mathcal{R}$. It applies some suitable rule such as exact divisibility of D_i by r_i. Next, digit $d_i \in \mathcal{D}$ is chosen from the residue class $D_i \bmod r_i$ using some appropriate rule, such as that of least absolute value, that of least non-negative value, making D_{i+1} a multiple of 2, or a random selection. The step is then repeated on $D_{i+1} = (D_i - d_i)/r_i$ until eventually $D_n = 0$ for some n. The recoding time is usually quadratic in $\log(D)$ as it involves a "short" division of the number $D_i - d_i$ with average size $\sqrt{D}$ by the base r_i which has length $O(1)$[2]. In most of the expected applications, a product in the group G will also take quadratic time, so that the recoding will take time equivalent to a very small constant number of group operations whereas exponentiation in G will take $O(\log(D)^3)$ time. Recoding time can then be ignored except for the smallest values of $\log(D)$ for which it is, in any case, insignificant.

The detail of the finite automaton driving the recoding (1) depends on device requirements, such as speed, space or side channel resistance. For efficiency, a choice giving $d_i = 0$ usually saves a multiplication in the exponentiation, and taking r_i as a power of 2 increases the proportion of squarings, which are cheaper than multiplications. Choosing the appropriate value for r_i may allow a Frobenius map to be applied, i.e. raising to a power equal to the characteristic of the underlying finite field. The cost of inversion or division in G will determine whether negative digits can be employed. Cheap double-and-add, triple-and-add and quintuple-and-add composite elliptic curve operations [4], [6], [9], [12], [23] could be employed when, for example, (r_i, d_i) can be chosen to be $(2, \pm 1)$, $(3, \pm 1)$ or $(5, \pm 1)$ respectively, but this has slight detrimental repercussions on the code area of the device. More precise recoding details for this and how to optimise the choices are described in [33].

The multi-base approach (2) is difficult to put into practice without imposing additional structure such as that given in the mixed base representation (1) [7]. Digits are generated from most to least significant, but the choice of R_i is problematic. The main exception is when every R_i is a power of the radix r in which D is presented but, strictly speaking, that generates a single base representation rather than a multi-base representation, and this is of type (1). A left-to-right windowing algorithm is the simplest non-trivial example of such a multi-base recoding, whereas the right-to-left windowing algorithm illustrates a mixed base recoding.

2. The finite automaton chooses a digit in $O(1)$ time, there are $n = O(\log D)$ digits and each division $D_{i+1} = (D_i - d_i)/r_i$ normally takes $O(\log D)$ time. However, if every r_i divides a power of the radix used in the initial presentation of D then this division just requires moving a pointer to the representation of D, which takes $O(1)$ time and makes the recoding algorithm linear in $\log(D)$.

4 THE EXPONENTIATION

The second phase in an exponentiation is to apply an addition chain corresponding to the recoding of D to $g \in G$ in order to compute g^D. This is performed in one of several ways. First, an algorithm specification is required to show how results for individual base/digit pairs (r_i, d_i) are combined to yield g^D. The standard left-to-right (L2R) and right-to-left (R2L) choices are illustrated in Figs. 1 and 2 respectively. These include Brauer's m-ary algorithm [3] and its dual by Yao [36] as special cases. Secondly, a process must be described for converting base/digit pairs into an addition chain which is annotated with the memory locations used for each operation. Specifically, for Figs. 1 and 2 this would mean prescribing how to raise group elements to the powers r_i and d.

It is now apparent that the recoding and exponentiation processes can often be run in parallel, as in sliding windows [19]. Then the recoding generates the elements of the addition chain just before use and in the same order as that in which they are consumed. However, the recoding process may have to generate the digits in the opposite order from consumption, in which case the two processes must run sequentially. In particular, this would also require storing the recoding.

Large tables in Figs. 1 and 2 would be too expensive for most embedded systems, and the speed-up they give is negligible beyond two or three entries. If maximising throughput for a given area is the objective, it is normally better to implement several exponentiators with very small tables. Hence the greatest interest in exponentiation methods falls naturally on the most compact cases. The binary square-and-multiply algorithm (in either direction) is the only sensible choice when the minimum number of locations, *viz.* 2, are available for storing elements of G. The main interest here is therefore in algorithms which use three such locations. So the target device has:

i) space for the recoded representation of the exponent D (if necessary);
ii) two places T_0, T_1 (say) for group elements, used for temporary storage or table elements;
iii) space P for the partially computed value of g^D; and
iv) any additional temporary working space required for performing a group operation.

The recoding space (i) is discussed in more detail in §6.9. Only one of the three locations T_0, T_1, P might technically be a register and the other two somewhere in memory, with values possibly only read or stored one word at a time. By performing a word-by-word swap between

Inputs: $g \in G$, $D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots +d_1)r_0+d_0 \in \mathbb{N}$, where $d_i \in \mathcal{D}$, $r_i \in \mathcal{R}$.
Output: g^D

1 Initialise table: $T[d] \leftarrow g^d$ **for all** $d \in \mathcal{D} \setminus \{0\}$.
2 $P \leftarrow 1_G$
3 **for** $i \leftarrow n-1$ **downto** 0 **do** {
3b **if** $i \neq n-1$ **then** $P \leftarrow P^{r_i}$
3a **if** $d_i \neq 0$ **then** $P \leftarrow P \times T[d_i]$ }
4 **return** P

Figure 1. Left-to-Right (L2R) m-ary, Sliding Window or Mixed Base Exponentiation

Inputs: $g \in G$, $D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots +d_1)r_0+d_0 \in \mathbb{N}$, where $d_i \in \mathcal{D}$, $r_i \in \mathcal{R}$.
Output: g^D

1 Initialise table: $T[d] \leftarrow 1_G$ **for all** $d \in \mathcal{D} \setminus \{0\}$.
2 $P \leftarrow g$
3 **for** $i \leftarrow 0$ **to** $n-1$ **do** {
3a **if** $d_i \neq 0$ **then** $T[d_i] \leftarrow T[d_i] \times P$
3b **if** $i \neq n-1$ **then** $P \leftarrow P^{r_i}$ }
4 **return** $\prod_{d \in \mathcal{D}\setminus\{0\}} T[d]^d$

Figure 2. Right-to-Left (R2L) m-ary, Sliding Window or Mixed Base Exponentiation

locations, we can survive with very little register space and with only one location being accessible to write results into during a group operation. This means that the read and write costs of different locations need to be accounted for when assigning weights to each base/digit choice in the recoding. It also means that copy and swap operations are required in the set of operations available for manipulating group elements.

The working space (iv) is necessarily rather ill-defined. For some groups it may be possible to discard input words once they have been used and use the released space to store the output words as they are computed, thereby saving space. It may also be possible to use part of the output register as working space until it is needed for the computed output. This level of detail is beyond the scope of the investigation, although we flag the differences between algorithms where this may be an issue that affects fair comparison.

5 Binary Algorithms

Suppose $\mathcal{D}$ were to contain only powers of 2. In this case, the step $P \leftarrow P^{r_i}$ in Fig. 1 or 2 would not require space to store an extra group element while another group operation was being computed. Hence both T_0 and T_1 can be used to store digit-related "table" values.

Suppose m is the largest power of 2 in $\mathcal{D}$ and 2 itself is also in $\mathcal{D}$. We might expect to need a complete set of residues modulo m in any recoding. However, with a sliding window, digit $d_i = 0$ is chosen with base $r_i = 2$ whenever $D_i \equiv 0 \bmod 2$ in the recoding algorithm. Hence only odd non-zero digits are required. This reduces the table size to at most $m/2$.

A further half of the table space may be saved if division in the group G has essentially the same cost as multiplication. Then the complete set $\{1-m/2, 3-m/2, \ldots, m/1-3, m/2-1\}$ of odd residues requires a supporting table with entries only for the positive odd residues $1, 3, \ldots, m/2-1$. Its size is just $m/4$. The multiplication in step (3b) of Fig. 1 is then replaced by the division $P \leftarrow P/T[-d_i]$ when the digit d_i is negative. Similarly the multiplication in step (3a) of Fig. 2 is replaced by $T[-d_i] \leftarrow T[-d_i]/P$ when d_i is negative, and the final product is, of course, only taken over the digits represented in the table T.

This extra halving of the table size occurs for elliptic curves since point additions and point subtractions have similar costs, but not for RSA as modular division is much more expensive than modular multiplication. So it is now apparent that the limit of 2 on the table size translates into a maximum base $m = 8$ for ECC applications and $m = 4$ for RSA applications if m is a base for which all non-zero residue classes are to be represented by digit values.

5.1 Time – Recoding Choices

With two table locations available, the usual binary square-and-multiply algorithm does not make full use of the given resources, and is consequently slower than what could be achieved: neglecting end issues, this binary method requires asymptotically one squaring and half a multiplication per exponent bit in either direction.

However, the table is large enough to use a base 4 sliding window with $\mathcal{R} = \{2, 4\}$ and digit set $\mathcal{D} = \{0, 1, 3\}$. Recoding is most easily done from the least significant end, but can be done in the opposite direction [16], [1]. On average, half the base choices have base $r_i = 2$ and so half the choices have base $r_i = 4$. The digit is 0 for $r_i = 2$ and non-zero for $r_i = 4$. Thus one third of bits are consumed by loop iterations involving one squaring and no multiplication, whereas the other two thirds of bits are consumed by loop iterations involving two squarings and one multiplication. So in addition to the one squaring per bit there is on average only $\frac{1}{3}$rd of a multiplication per bit. This is faster than the binary square-and-multiply.

When division is feasible, the recoding can use a 3-bit window with $\mathcal{R} = \{2, 8\}$ and $\mathcal{D} = \{-3, -1, 0, 1, 3\}$. The table T again contains elements indexed by the digit subset $\{1, 3\}$. With a similar argument to that above, base choices 2 and 8 are equally likely, and so one quarter of bits are consumed by loop iterations involving one squaring and no multiplication, whereas the other three quarters of bits are consumed by loop iterations involving three squarings and one multiplication or division. So in addition to the one squaring per bit there is on average

only $\frac{1}{8}$th of a multiplication and $\frac{1}{8}$th of a division per bit – a total of $\frac{5}{4}$ operations per bit. This is cheaper than the base 4 sliding window with only non-negative digits if the cost of one division is less than $\frac{5}{3}$rds the cost of a multiplication.

Finally, the w-NAF recoding has $\mathcal{R} = \{2, 2^w\}$ and $\mathcal{D} = \{d : |d| < 2^w, d \text{ odd}\}$, with base 2 always having digit 0, and adjacent digits not being both non-zero. The recoding chooses $d_i = D_i \bmod 2^w$ or $d_i = (D_i \bmod 2^w) - 2^w$ to make D_{i+1} even when D_i is odd. Pairing each non-zero digit with the following zero digit gives a window of $w+1$ bits. Hence w-NAF is the same as the base 2^{w+1} sliding window recoding described above for which $\mathcal{R} = \{2, 2^{w+1}\}$ but taking $\mathcal{D}$ as for w-NAF. Thus, it does not provide a distinct method to be added to those already under consideration.

5.2 Time – Direction Issues

Still considering only radix choices which are powers of 2, with a bit of care, the same number of squarings and (non-squaring) multiplications should occur between both the algorithms of Figures 1 and 2. Clearly, most loop iterations corresponding to the same digit/base pair will require the same number of squarings and multiplications. So, for the two algorithms, we will count how many extra operations are required over and above those expected for the loop iterations, assuming t tabulated digits $1, 3, 5, \ldots, 2t-1$. Without loss of generality, assume that $d_{n-1} \neq 0$ since any optimisation should adjust n to make this true.

In the L2R algorithm, initialisation should involve one squaring to obtain g^2 (temporarily stored in P, say) and $t-1$ multiplications to obtain the powers $3, 5, \ldots, 2t-1$ of g. In the first loop iteration, one multiplication is saved by using an initialisation when P is first updated from 1_G. This totals one squaring and $t-2$ multiplications more than what is naïvely expected from the loop.

In the finalisation step of the R2L algorithm, the return value R is accumulated in the space used by $T[2t-1]$ to avoid using extra space, and an auxiliary product is constructed in P:

$P \leftarrow T[2t-1]$
$R \leftarrow P$;
for $i \leftarrow t-1$ **downto** 1 **do**
{ $P \leftarrow P \times T[2i-1]$
if $i = 1$ **then** $R \leftarrow R \times R$
$R \leftarrow R \times P$
}

At the end of iteration $i > 1$ this has created $T[2i-1] \times \ldots \times T[2t-1]$ in P and $T[2i-1]^1 \times T[2i+1]^2 \times \ldots \times T[2t-1]^{t+1-i}$ in R. So it outputs the value $R = T[1]^1 \times T[3]^3 \times T[5]^5 \times \ldots \times T[2t-1]^{2t-1}$, as required. The code performs one squaring and $2t-2$ multiplications. However, there are t iterations of the main R2L loop in which the table values are updated from 1 to a non-trivial value. If these are replaced by initialisations directly to the required values, then the number of extra multiplications drops to $t-2$.

Thus both algorithms use the same number of multiplications and squarings. This becomes automatic when the duality mechanism of [34] is applied (*see* §6.4)[3]. That mechanism even changes the appropriate loop multiplications in one direction into initialisations in the other. The equality in operation counts means that the most significant time differences between the two processing directions is in

- the use of special forms for group elements and group operations;
- reading and writing data to and from memory.

In the L2R version time could be spent putting the table elements g^d into a special form which makes the multiplications by $T[d]$ more efficient. Similarly, in the R2L version there may be a special form that can be used for P that is preserved under squaring and which speeds up the squaring or the multiplication of $T[d]$. Both these possibilities would normally reduce the overall time by a small linear factor. Examples include the double-and-add operation [10], [4].

For each operation, reading/writing is effectively linear in the size of the group elements whereas the operations themselves are typically quadratic in the element size. Hence the overall difference in speed arising from this will be negligible once cryptographic lengths are reached. It is worth observing, though, that in the main loop the L2R version writes only to P and only reads from T whereas the R2L version reads from and writes to both. This means potentially more data movement in the R2L case.

5.3 Integrating Recoding with Exponentiation

The binary recoding algorithms of §5.1 are usually applied to D from least to most significant digit. Hence, for the R2L direction, the recoding can be done in parallel with the exponentiation and there is no need to store the recoding. This is also true for the L2R direction if D is presented in binary. Avanzi [1] shows how to generate from left to right a sliding window recoding of similar efficiency and resource characteristics to the above R2L recodings. This is as follows.

Suppose division is expensive, so that no negative digits are to be chosen. Then the digit set can again be taken to be $\mathcal{D} = \{0, 1, 3, 5, \ldots, m-3, m-1\}$, which consists of 0 and all the odd numbers less than m, but take $\mathcal{R} = \{2, 4, 8, \ldots, m\}$. If D' is the remaining unrecoded suffix if D, then the recoding algorithm chooses base $r = 2$ and digit $d = 0$ when the leading bit of D' is 0. Otherwise, let d' ($\neq 0$) be the value of the leading $w = \log_2 m$ bits of D' and suppose $2^s || d'$ where '$||$' denotes the highest power giving exact division. Then take base $r = 2^{-s}m$ and digit $d = 2^{-s}d'$. So, as d is odd,

3. This explains why the total counts in Möller's Tables 1 and 2 [24] are equal.

$d \in \mathcal{D}$. This is then repeated on the rest of D. As the next s bits of D are 0, the next s choices would be of base 2 with digit 0. Once fewer than w bits remain, the same procedure is applied with w equal to the number of remaining bits.

On the other hand, if division is relatively cheap then negative digits are employed and the digit set can be taken as $\mathcal{D} = \{0, \pm 1, \pm 3, \ldots, \pm(m/2-1)\}$. Again, $\mathcal{R} = \{2, 4, 8, \ldots, m\}$. At any point in the L2R recoding, suppose D' is the unrecoded suffix of D and $\overline{D'}$ the complementary recoded prefix. So $D = \overline{D'}||D'$. Suppose the recoded digits generated so far have value $\overline{D'}+b'$ where b' is the leading bit of D'. (This is an integer addition, not a concatenation of binary strings interpreted as an integer.) A 0 bit is prepended to D if necessary so that this holds initially when no digits have been generated. Suppose $D' = b'||D''$ and b'' is the leading bit of D''. If $b' = b''$ then base $r = 2$ and digit $d = 0$ are chosen. The generated digits then have value $r(\overline{D'}+b')+d = \overline{D''}+b''$ where $\overline{D''}$ is the prefix complementary to suffix D''. So our required loop invariant is preserved in this case. Otherwise, suppose d'' is the value of the leading $w = \log_2 m$ bits of D', D'' is the remaining suffix of D' with complement $\overline{D''}$, and D'' has leading bit b''. Then $D' = d''||D''$ as strings and $m\overline{D'}+d'' = \overline{D''}$ as values. Define $d' = d''-b'm+b''$. Then $0 \le d' \le m/2$ if $b' = 0$, whereas $-m/2 \le d' \le 0$ if $b' = 1$. So $d' \in \mathcal{D}$ when d' is odd or zero. In this case choose base $r = m$ and digit $d = d'$. Then the digits generated so far have the value $r(\overline{D'}+b')+d = \overline{D''}+b''$ so that the required loop invariant holds again. If d' is even and $2^s||d'$ then d' is split up as before into an odd digit $d'2^{-s}$ with base $m2^{-s}$ followed by s digits 0 with base 2. After this the loop invariant holds as before.

Thus the main binary recoding algorithms can, with only minor differences, be performed digit by digit in either direction.

5.4 Side Channel Leakage

Implementation weakness arises because when the table element is fetched, used, or written there is a danger that enough of its Hamming weight, address or other identifying information will leak through power variation or electromagnetic waves for an adversary to determine which exponent digits are equal, and thereby recover the secret key D. For L2R, the table elements and their Hamming weights are fixed throughout the computation, whereas for R2L, although there is more movement of "table" data, only the addresses are fixed. Countermeasures might blind this information leakage, but R2L has a clear advantage without such blinding. R2L can use random relocation of the data as a cheap and easy way of blinding the addresses, whereas data blinding for L2R is much more expensive.

We ignore the leakage problems of executing different code for zero digits. This is common to both directions and can be avoided by minor, judicious recoding (such as dummy multiplications) and perhaps the cost of another register as well as more time if zero were excluded from $\mathcal{D}$ [15][4]. Indeed, one rather weak solution is to include $T[0]$ initialised to the identity element 1_G, and remove the conditional restriction in the main loop so that multiplications by/of $T[0]$ are performed. However, if different non-zero digits cannot be distinguished in a cost effective way by the attacker, then knowing the positions of zero digits is probably insufficient in many cases to reduce the search space for a secret key to a computationally feasible size. Specifically, in our target platform there is a table of two elements, representing two or four digits. The attacker has to distinguish correctly which one is used in each multiplication in order to discover the secret exponent. This is extremely difficult even with strong leakage which distinguishes individual digits [32].

Inputs: $g \in G$, $D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots +d_1)r_0+d_0 \in \mathbb{N}$, where $d_i \in \mathcal{D}$, $r_i \in \mathcal{R}$.
Output: g^D

```
1   T ← 1_G
2   P ← g
3   for i ← 0 to n−1 do in parallel {
3a      if d_i ≠ 0 then T ← T×P^{d_i}
3b      P ← P^{r_i} }
4   return T
```

Figure 3. Compact Right-to-Left (R2L) Mixed Base Exponentiation [28]

6 Mixed Base Exponentiation

Let us now consider using a set $\mathcal{R}$ of bases which are not all powers of 2. There has been recent research to develop efficient composite double-and-add, triple-and-add, quintuple-and-add, etc. operations for elliptic curve cryptography. These can be used in a left-to-right evaluation when the exponent D has a mixed base representation which includes the bases 2, 3 and 5 respectively, *e.g.* [9], [23], [21]. Usually the best strategy is to select a radix r for which the digit is 0 since it saves a multiplication, and otherwise take $r = 2$. So $d_i = 0$ if $r_i \ne 2$ and odd or zero digits d_i for $r_i = 2$ such that $|d_i| < 2t$ for the table size $t = |T|$. Within the restrictions imposed by availability of digits, this re-coding was used for the L2R and R2L results in Table 2 and it is close to that of [21].

Figures 1 and 2 are presented in a form which shows how to apply the recoding to perform exponentiation. (We discuss these in more detail below.) However, there is also a more compact tableless form given in Figure 3 in which there are only two quantities being stored or updated: the usual cumulative product in T and a

4. Another register would be needed if a digit were to be added to $\mathcal{D}$ to compensate for the loss of 0.

digit-independent power $g_i = g^{R_i}$ for the radix product $R_i = \prod_{j=0}^{i-1} r_j$ in location P. The iterative step forms $g_{i+1} = g_i^{r_i}$ in P, sharing as much as possible of the calculation with that for $g_i^{d_i}$, and multiplies the latter onto T when formed. To avoid extra multiplications it is best if the only digits allowed are those that appear naturally when raising to the power r_i (§6.1). They need to appear in an addition chain for r_i, ideally of minimal length. In §6.2 the dual compact L2R mixed base algorithm is constructed. It has a table of just one element, corresponding to T in the R2L form, and, in a similar way to there, the underlying addition chains for r_i simulate having a much larger table. By using dual addition chains, the exponentiation has the same cost in terms of multiplications and squarings for the two directions.

6.1 Digit Availability

For this section it is assumed that not all bases are powers of 2. So extra memory has to be available for storing intermediate powers of $g \in G$ when raising to non-2-power exponent r. The space for this is normally equivalent to one table entry, say P', and represents the first major hidden space difference between binary and properly mixed base exponentiations. So let P and P' denote the two registers (or other memory) used for constructing and storing the rth powers and, as before, let T be an array, indexed by a subset of digits, for holding the digit-related table or accumulating products. When T holds just one element then this satisfies the space restrictions permitted in §4 for the target platform. One is apparently therefore restricted to only the digit values $\mathcal{D} = \{0, 1\}$ or $\mathcal{D} = \{0, \pm 1\}$, the latter when inversion is cheap. However, the table entry can be multiplied in at any time during step (3b) of the three figures so far rather than as a separate step (3a) before or after. This turns out to be equivalent to having more digits available.

Several minimal length addition chains are of particular interest: $1{+}1 = 2$ for base 2, $1{+}1 = 2$, $1{+}2 = 3$ 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. These show that any digit power P^d can be constructed *en route* to the radix power P^r and at no further extra expense when $0 \le d < r$ and $r \in \{2, 3, 5\}$. Once P^d is obtained in Fig. 3, it is multiplied onto T without incurring any space penalty because it requires no extra resources beyond what are already required for the next multiplication in forming P^r. For digit powers created in this way, the number of group operations required to evaluate corresponding loop iterations will be the same for the exponentiation schemes of Figs. 1, 2 and 3. This number is just that required for step (3b), plus one when the digit is non-zero. Moreover, the cost will still be the same even if we differentiate between squaring and non-squaring operations.

However, for bases larger than 5 there may be digits which are not in a shortest addition chain for the chosen base. In any of the exponentiation algorithms, digits not included in the table or obtained when raising to the power r can be assigned an extra cost, such as the number of extra group operations required to obtain the requisite power using a longer addition chain – perhaps ∞ if there is insufficient space for its computation. For example, in the L2R algorithm of Fig. 1, once the rth power has been created in P, the working space P' becomes available for constructing any g^d: the usual binary algorithm can be used, holding the product in P' and multiplying in g from T when necessary. For the R2L algorithms of Figs. 2 and 3 a similar process is possible. Specifically, for $h \in G$ initially in P, any h^d can be created in P' using the binary algorithm while retaining h in P. This is then multiplied into T and subsequently h^r is created in P. Thus, every digit is available, although not necessarily at an acceptable price.

The need for such costly extra digits can often be avoided without incurring any penalty simply by a better recoding. For example, any digits d larger than the base r could be replaced by $d{-}r$ and a carry of 1 generated to the next digit up. In this way, the extra group operations needed for creating these digit powers are used instead to multiply their more naturally occurring constituent components directly into T.

6.2 A Location-Specific Matrix Representation for Addition Chains

As noted above, the addition chains for 2, 3 and 5 automatically yield the digit powers between 0 and the radix choice r. The required value is multiplied into T when created in the compact R2L scheme of Fig. 3. Hence there is no extra cost for allowing certain digits although table entries for them may not exist. Similarly, in an L2R scheme, if g^1 in $T[1]$ is multiplied into P at the right time, it gives rise to g^d at the end of the loop iteration, and so is equivalent to having a pre-calculated table value g^d. For example, with base 5, digit 4 and h initially in P, the multiplications to perform are: $P{\times}T[1] \to P'$, $P'{\times}P' \to P'$, $P'{\times}P' \to P'$, $P{\times}P' \to P$. Although this creates the desired $h^5 g^4$ in P, it is unclear what digit powers could be obtained in this way. In fact, all that is needed is to take the *dual* of an addition chain for (r, d).

There appears to be no concept of location-aware dual addition chains prior to [34][5]. A register-specific description is required to determine that a dual process uses similar execution space as well as time. Suppose t registers are available for computing powers of $g \in G$. Then addition chain operations are defined more precisely by corresponding $t{\times}t$ matrices indexed by the registers. These provide additional location information by specifying where the group elements are read from and written to. They act on a row or column vector containing the exponents of the powers of g in those

5. There is a well-known *transposition* method which reverses edges in the computational di-graph [18]. If done carefully, time can be preserved, but it fails to address any space issues.

registers. Suppose $M_{n-1}, M_{n-2}, \ldots, M_1, M_0$ are matrices representing the addition chain operations. (They may represent more complex group operations than simple multiplications.) For convenience, these are written in the same order as the recoding of D so that M_0 will normally denote the start of the chain and be part of the sub-chain for creating the least significant digit. Let the row vector v, indexed by the registers, represent the values stored initially in the registers. For the right-to-left (R2L) view the matrices are applied in right-to-left order on the left to vector v^{T} to evaluate $M_{n-1}\ldots M_1 M_0 v^{\mathsf{T}}$ and yield the values in the registers at the end of the computation. For the left-to-right (L2R) interpretation the matrices are applied in left-to-right order to the right of v to evaluate $vM_{n-1}\ldots M_1 M_0$.

For example, suppose g^a and g^b are initially in P and P' respectively. Then, in the R2L view, the addition chain operations corresponding to the multiplication $P\times P' \to P$ and the squaring $P\times P \to P$ are represented using matrices and vectors indexed by (P, P') as, respectively,

$$\begin{pmatrix} a+b \\ b \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix}$$

and

$$\begin{pmatrix} 2a \\ b \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix}$$

In Fig. 3 the contents of P and P' are entirely given by applying a sequence of such matrices. At some point during a loop iteration the required digit power h^d of the initial value h appears in P, say, with h^e, say, in P' (or *vice versa*). Then, at the end of the iteration, h^r appears in P and some other unwanted value h^* in P'. Without loss of generality, the matrices representing these addition sub-chains have the forms

$$\begin{pmatrix} d & 0 \\ e & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} r & 0 \\ * & 0 \end{pmatrix} = \begin{pmatrix} i & j \\ * & * \end{pmatrix}\begin{pmatrix} d & 0 \\ e & 0 \end{pmatrix}$$

respectively, where the matrix with row $(i\ j)$ corresponds to the part of the addition chain after d is generated. So $r = id+je$. After applying the addition chain corresponding to the matrix with column $(d\ e)^{\mathsf{T}}$, the element in register P is multiplied into T and the process continues with the addition chain corresponding to the other matrix. We would need to move up to 3×3 matrices indexed by (P, P', T) to represent this action. For the L2R dual, the components of the same matrix product are applied from left to right on a row vector, say $(a\ 0)$. So, once

$$(a\ 0)\begin{pmatrix} i & j \\ * & * \end{pmatrix} = (ai\ aj)$$

has been formed by applying the left hand factor to the row, the element g^1 in T is multiplied into P to yield $(ai{+}1\ aj)$. For convenience, we skip the 3×3 matrix view necessary to illustrate this. Then application of the second factor creates

$$(ai{+}1\ aj)\begin{pmatrix} d & 0 \\ e & 0 \end{pmatrix} = ((ai{+}1)d{+}aje\ 0) = (ar{+}d\ 0)\ .$$

This is exactly how P needs to be updated by a loop iteration in Fig. 1. In both directions this creates the illusion of having a table element for digit d.

6.3 The Dual Addition Chain

Ideally, the dual addition chain would be that specified by applying the matrices for each addition step in left to right order to a row vector instead of right to left on the corresponding column vector (or *vice versa*). Unfortunately, not every matrix which behaves as a multiplication in one direction behaves as a multiplication in the other direction. So some extra care is necessary. For example, in the R2L direction, matrix $\begin{pmatrix} 0 & 2 \\ 0 & 1 \end{pmatrix}$ acting on a column vector represents the operation of squaring the value in one register and writing the result into the other register. However, this becomes a composite square-and-multiply operation (or double-and-add) with an initialisation when applied in the opposite direction, i.e. to a row vector. So the permitted atomic addition chain operations are restricted to exclude such cases; they are replaced by compositions of simpler operations. If the target device cannot perform every allowed atomic operation – such as writing an output to the location of an input – then the individual operations can be combined into composite ones which the device can execute.

Since the location of values is important, some extra operations are required to move data around as well as the usual ones for squaring and multiplication. Altogether,

Definition 1: (*cf* [34], Def. 2) Seven classes of *atomic* operations are allowed in *location-aware addition chains*:

- Copying one register to another;
- Copying one register to another & then initialising the source register;
- Swapping the values of two registers;
- Inverting the value in one register;
- In-place squaring of the contents of one register;
- Multiplying two different registers into one of the input registers;
- Multiplying two different registers into one of the input registers, & then initialising the other input.

These operations do not change the values of any registers other than those mentioned. □

Any addition chain can be written using only the above operations whatever registers are prescribed for reading and writing. For example, if an operation, such as multiplication or squaring, needs to write to a register P'' which is not an input to the operation, then one of its inputs is simply copied into P'' first and then the operation performed using that copy. If multiplications or squarings can only be performed on and to certain locations, then swaps of arguments and results can be used to provide operations of greater generality. However, for convenience, in the following it is assumed that there

are no restrictions beyond those specified above. Including an inversion operation enables addition-subtraction chains to be represented and dualised as well. No initialisation only operation is required because it is incorporated when necessary into other operations. If desired, the squaring and inversion classes might be extended to include other unary powering operations such as a Frobenius map.

For a device with two locations, R2L examples of each class are, respectively,

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix},$$
$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \text{ and } \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$

This set is closed under transposition, and so the L2R interpretations of them are also atomic operations. If we define a location-aware addition chain as being a sequence of such matrices then the transpose provides a well-defined *dual* chain. Alternatively, a dual is obtained by applying the matrices in the opposite order to a column vector instead of a row vector of locations, or *vice versa*. Of course, independently of the matrix representations, the operators have transposes which enable the dual to be defined. Removing the location details and the copying operations leaves a classical addition chain consisting of only squaring and multiplication operations. So a dual for the classical addition chain is obtained.

6.4 Counting Multiplications in the Dual Chain

Ideally we would like to have the same number of multiplications and squarings in an addition chain and its dual. This requires some restrictions on the allowable location-aware chains to make them *normalised*. The form is achieved by repeatedly applying the following to a chain of atomic operations until none applies (*cf* [34] Defn. 6):

i) any operation with 1_G as an input should be replaced by a simpler one which does not require that argument;
ii) any operation with an argument which is not used subsequently should be replaced by an equivalent one that re-initialises that argument; and
iii) any operation whose result is unused must be deleted.

Note that none of these modifications changes the number or type of the multiplicative operations, nor the overall output. One more condition is imposed, namely,

iv) the number of inputs to the chain is the same as the number of outputs from the chain.

So, if the same registers are used for output as for input, there is a complementary set of non-I/O locations used solely for intermediate calculations.

Theorem 1: [34] Using the above construction for location-aware addition or addition-subtraction chains, the number of squarings and the number of multiplications is the same for a normalised chain and its dual. □

This is easily proved by counting the number of operations which initialise a location to 1_G and equating it to the number of operations which overwrite a location containing 1_G.

Removing the location details yields a corollary for the classical addition chain:

Theorem 2: A normalised addition chain and its dual have the same length. □

When the normalised chain computes only the value g^D, the matrix representing its action has a single non-zero value, namely D. So the dual also computes g^D (providing the reading and writing locations are interchanged). However, if the chain computes several values – a multi-exponentiation – then its dual may compute different values. Clearly, if inputs and outputs are to be via the same locations for a chain and its dual, then the two chains compute the same values if, and only if, the matrix representing the action of one is symmetric, i.e. equal to its transpose.

6.5 Some Duality Examples

Suppose a device has locations $R0$, $R1$ and $R2$ with $R1$ used for I/O. Let $\leftarrow_I$ denote assignments which include initialisation of any input location which is not the location being assigned to. Then the following is a location-aware chain which outputs g^3 in $R1$ if $g \in G$ is initially in $R1$. It follows the construction rules provided in the previous section.

$$R2 \leftarrow R1; \quad R2 \leftarrow R1{+}R2; \quad R0 \leftarrow_I R1;$$
$$R1 \leftarrow_I R2; \quad R1 \leftarrow_I R0{+}R1;$$

This includes two multiplications and no squarings. The R2L matrix equivalent acting on the column vector $(R0, R1, R2)^{\mathsf{T}}$ is

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$
$$\times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Applying this L2R on a row vector $(R0, R1, R2)$ gives the dual location-aware chain

$$R0 \leftarrow R1; \quad R2 \leftarrow_I R1; \quad R1 \leftarrow_I R0;$$
$$R1 \leftarrow R1{+}R2; \quad R1 \leftarrow_I R1{+}R2;$$

which also has two multiplications and no squarings.

As a second example, the classical addition chain $1{+}1{=}2, 2{+}2{=}4, 4{+}1{=}5$ shows roughly how to implement the base/digit pair $(5, 4)$ in the R2L algorithm of Fig. 3. Adding the location information for registers T, P, P'

gives

$$P' \leftarrow P;\ P \leftarrow 2P;\ P \leftarrow 2P;\ T \leftarrow T{+}P;\ P \leftarrow_I P{+}P';$$

This has two I/O registers, namely T and P. It acts on $(g, h, *)$ to produce $(gh^4, h^5, 1_G)$ using two squarings and one multiplication. The dual of the chain is

$$P' \leftarrow P;\ P \leftarrow T{+}P;\ P \leftarrow 2P; P \leftarrow 2P;\ P \leftarrow_I P{+}P';$$

which acts on $(g, h, *)$ to produce $(g, h^5g^4, 1_G)$, also using two squarings and one multiplication. This is the updating required for a left-to-right algorithm.

6.6 The Montgomery Powering Ladder

Constructing a dual for the Montgomery Powering Ladder [25], [17], [35] illustrates some of the subtleties of normalisation which, if ignored, may lead to a different number of some types of operation in the dual.

A slightly modified version of the original algorithm is given in Fig. 4. Taking the transpose yields the remarkedly similar algorithm of Fig. 5 which computes the same value. The correctness proof for the L2R version simply involves checking that $T[0] = g^{D_i}, T[1] = g^{D_i+1}$ where $D_i = \sum_{j=i}^{n-1} d_j 2^j$ at the end of the loop iteration using d_i. That for the R2L version involves checking that $T[0] = g^{2^i - D_i{}'}, T[1] = g^{D_i{}'}$ where $D_i{}' = \sum_{j=0}^{i-1} d_j 2^j$ at the end of the loop iteration using d_{i-1}.

The usual requirement for $d_{n-1} = 1$ has been dropped and the initialisation simplified in Fig. 4 to give a more uniform presentation. Those conditions would need to be restored to ensure that each loop iteration consisted of a non-trivial multiplication followed by a non-trivial squaring. However, the main problem for normalising the usual version of the Ladder is the redundant operation on $T[1]$ in the last loop iteration – the final value of $T[1]$ remains unused.

Observe that if $D \equiv 0 \mod 2^a$ then the last $a{+}1$ loop iterations have non-trivial, unused multiplications in step (2a) of Fig. 4, whereas the corresponding first $a{+}1$ loop iterations in Fig. 5 have trivial multiplications by 1_G in step (2a). The cost of the two algorithms is therefore different unless there are extra conditions. Here, requiring $d_0 \neq 0$ and, dually, $d_{n-1} \neq 0$ for both algorithms limits the trivial and superfluous operations at each end of the algorithms so that the numbers of squarings and multiplications could be equated. Under these extra conditions the initialisation and finalisation could be easily modified to include the first and last loop iterations and yield normalised, regular algorithms.

An interesting difference between the algorithms is that the loop operations in the L2R version can be computed in parallel whilst those in the dual can not. On the other hand the dual version can use a more efficient double-and-add operation on an elliptic curve. The left-to-right version can also compute g^D on an elliptic curve using only the x- and z- projective coordinates using the known coordinates of $T[0]$, $T[1]$ and $T[1]/T[0] = g$ to recover the y-coordinate. Unfortunately,

Inputs: $g \in G$, $D = \sum_{i=0}^{n-1} d_i 2^i$ where $d_i \in \{0, 1\}$ for $0 \leq i < n-1$.
Output: g^D

1a $T[0] \leftarrow 1_G$
1b $T[1] \leftarrow g$
2 **for** $i \leftarrow n{-}1$ **downto** 0 **do** $\{$
2a $\quad T[1{-}d_i] \leftarrow T[0]{\times}T[1]$
2b $\quad T[d_i] \leftarrow T[d_i]^2$ $\}$
3 **return** $T[0]$

Figure 4. Left-to-Right (L2R) Montgomery Powering Ladder [17]

Inputs: $g \in G$, $D = \sum_{i=0}^{n-1} d_i 2^i$ where $d_i \in \{0, 1\}$ for $0 \leq i < n-1$.
Output: g^D

1a $T[0] \leftarrow g$
1b $T[1] \leftarrow 1_G$
2 **for** $i \leftarrow 0$ **to** $n{-}1$ **do** $\{$
2b $\quad T[d_i] \leftarrow T[d_i]^2$
2a $\quad T[d_i] \leftarrow T[0]{\times}T[1]$ $\}$
3 **return** $T[1]$

Figure 5. Quasi-Dual Right-to-Left (R2L) Montgomery Powering Ladder

this is not possible in the dual version as the coordinates of $T[1]/T[0] = g^{2^{n-1}}$ are unknown. This example of duality only uses two locations, however, and so falls outside the main scope of this study.

6.7 The New Compact L2R Mixed Base Algorithm

For any given base r it is relatively straight-forward to generate addition chains for r and see which digit values d are produced by each chain. When register information is added, these enable the R2L division chain algorithm of Fig. 3 to be executed. The construction of the dual location-aware addition chain enables the algorithm to process the digits in the opposite L2R order with the same space resources and the same numbers of squarings and multiplications. Moreover, all the digits available for the R2L direction are also available for the L2R direction. The resulting new compact L2R algorithm achieves the same results at the end of each loop iteration as the algorithm of Fig. 1, but requires a table T containing only the initial $g \in G$, not every digit power. The algorithm, first given in [34], is outlined in Fig. 6 and the loop iteration detail is provided by using a dual chain for the base/digit pair, as illustrated in §6.5.

For completeness, Table 1 provides the L2R register-level instructions suitable for the most common base/digit pairs (r, d), $d \leq 5$. The notation for the operation sequences on the three registers T, P, P' is

Inputs: $g \in G$, $D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots +d_1)r_0+d_0 \in \mathbb{N}$, where $d_i \in \mathcal{D}$, $r_i \in \mathcal{R}$.
Output: g^D

1 $T \leftarrow g$
2 $P \leftarrow 1_G$
3 **for** $i \leftarrow n-1$ **downto** 0 **do**
3a $\quad P \leftarrow P^{r_i} \times T^{d_i}$
4 **return** P

Figure 6. Compact Left-to-Right (L2R) Mixed Base Exponentiation [34]

self-explanatory. All initialisations have been removed because none has practical value except when taking the dual. The last column indicates one way of assigning the operation types, including double-and-add (DA), triple-and-add (TA) and quintuple-and-add (QA) composite operations. (Following the elliptic curve context in which efficient implementations of these may be available, additive notation has been used to describe them rather than multiplicative notation.) Since there may be time to put the table element in a special form (such as affine rather than projective coordinates), multiplication by the table element may be faster than multiplication by a random element. This is denoted by A rather than A. If the digit range includes negative digits and division costs the same as multiplication, then more of the loop iterations could make use of this special form with the cheaper operation A.

(r,d)	*Operation Sequence*			*Operations*
(2,0)	2P $\to$ P ;			D
(2,1)	2P $\to$ P ;	T+P $\to$ P ;		D*A*
(3,0)	P $\to$ P′;	2P′ $\to$ P′;	P+P′ $\to$ P ;	T
(3,1)	P $\to$ P′;	2P′ $\to$ P′;	P+P′ $\to$ P ; P+T $\to$ P ;	T*A*
(3,2)	P $\to$ P′;	T+P′ $\to$ P′;	2P′ $\to$ P′; P+P′ $\to$ P ;	*A*, DA
(5,0)	P $\to$ P′;	2P′ $\to$ P′;	2P′ $\to$ P′; P+P′ $\to$ P ;	Q
(5,1)	P $\to$ P′;	2P′ $\to$ P′; P+P′ $\to$ P ;	2P′ $\to$ P′; P+T $\to$ P ;	Q*A*
(5,2)	P $\to$ P′;	2P′ $\to$ P′; 2P′ $\to$ P′;	T+P′ $\to$ P′; P+P′ $\to$ P ;	D*A*, DA
(5,3)	P $\to$ P′;	T+P′ $\to$ P′; P+P′ $\to$ P′;	P+P′ $\to$ P ; P+P′ $\to$ P ;	*A*, A, A, A
(5,4)	P $\to$ P′;	T+P′ $\to$ P′; 2P′ $\to$ P′;	2P′ $\to$ P′; P+P′ $\to$ P ;	*A*, D, DA

TABLE 1
Register Instructions for Loop Iteration in the L2R Compact Mixed Base Exponentiation

It is, of course, always possible to include a larger table T in the compact L2R algorithm. For example, including g^3 in the table would allow a cheaper implementation of base 4 with digit 3. However, the space limitation to three registers makes this impossible on the target device. Space permitting, the dual R2L algorithm to this would also have a larger T used in a way similar to that in Fig. 2.

6.8 Recoding Time

Some mixed base recoding algorithms are presented in [7], [28], [30] and above. Generally, the algorithm decides the ith base/digit pair (r_i, d_i) using the value of $D_i \bmod \pi$ where π is divisible by every element in $\mathcal{B}$. This value may enable several digits, say j, to be decided at one go if π is increased sufficiently, ideally to just less than the maximum value of a machine word. Then determining $D_i \bmod \pi$, base/digit pairs i to $i+j-1$, and D_{i+j} uses the school book short division algorithm and will typically require between $2n_i$ and $4n_i$ word$\times$word divisions or multiplications where D_i is n_i words long, plus a similar number of additive operations. It will also make D_{i+j} around half a word or so shorter than D_i, say, the actual amount depending on how fussy the recoding algorithm is, i.e. how large π must be to decide one digit, and therefore how many digits can be decided with the chosen π. On average n_i is half the word length of D, and so the total number of word operations for the recoding should be no more than that for around half a dozen multiplications of integers the size of D – very comparable with the cost of a single elliptic curve point addition. This cost is the main time penalty for using a mixed base representation rather than a purely base 2 sliding window which requires at most a partitioning of the exponent bits. However, if D is already represented in base π, then much of this cost can be saved.

This method for the recoding clearly has a word operation complexity of $O(n^2)$ where $n = O(\log D)$ is the key length, i.e. the number of digits in D. A multiplication in G is also typically $O(n^2)$, so that the exponentiation has a word complexity of $O(n^3)$. Consequently, for cryptographic key lengths the recoding cost should be a relatively insignificant part of the whole computation.

6.9 Recoding & Other Space Issues

Given a binary representation of D, the mixed base representation can only be generated from left to right, forcing the storage of the recoding if an L2R exponentiation method is used. The space required depends on the recoding function, but is generally under about $3\log_2 D$. Since $\prod_{i=0}^{n-1} r_i \approx D$, the binary representations for the sequence $r_0, r_1, \ldots$ will take about $\log_2 D$ bits, and, if there are at most r digit choices for base r then the sequence $d_0, d_1, \ldots$ takes another $\log_2 D$ bits, approximately. Another $\log_2 D$ bits or so may be used to mark the start of each base/digit pair.

In practice some simple compression may reduce this considerably. For $\mathcal{R} = \{2, 3\}$ and digits in the range 0 to $r-1$, a single bit could be used for the base, one bit for a base 2 digit, and two bits for a base 3 digit. If

every base choice were 2, then the space taken would be $2\lceil \log_2 D \rceil$ bits. If every base choice were 3, then the space taken would be $3\lceil \log_3 D \rceil \approx 3 \log_3 2 \log_2 D < 2 \log_2 D$. Whatever the ratio of choosing base 2 against base 3, the total will be between these limits, and therefore bounded by $2\lceil \log_2 D \rceil$ bits.

Another typical choice might be $\mathcal{R} = \{2, 3, 5\}$ with the digit set $\{0, \pm 1\}$ for base 2 and only digit 0 for bases 3 and 5. One bit would indicate whether base 2 was chosen. In the case of base 2, two further bits would indicate the digit choice. Otherwise, one extra bit would be used to indicate the choice of base 3 or 5. Then there would be an upper bound of $3\lceil \log_2 D \rceil$ bits for the recoding space. This could be reduced if every non-zero digit had to be followed by a digit 0.

When the recoding is generated on-the-fly for R2L methods, the only extra storage requirement is that for the value of D_i, which is the same as that for D. So L2R requires more space than R2L if elements of $\mathcal{R}$ are not all powers of a single number. However, in many protocols the exponent is random, and might be generated on-the-fly in mixed base form. In this case the space required by a recoding of D can be ignored, making L2R and R2L space requirements identical.

Finally on space issues, unlike the binary case with D given in binary, D is destroyed by the mixed base recoding. Hence, for both directions, extra space may be required in the mixed base case to store the initial value of D. Also, unlike the L2R cases, the input "message" $g \in G$ is destroyed by all the R2L methods, and so further space may be required to preserve it.

6.10 Composite Elliptic Curve Operations

The duality between L2R and R2L suggests speeds should be almost identical. However, particularly with reference to the group of points on an elliptic curve, the pre-computed L2R table entries might be manipulated into special forms (such as affine rather than projective coordinates [5]) which may make all the multiplications[6] by the digit powers in T cheaper in the L2R direction than the R2L direction [9], [23], [21]. 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) [22]. On the other hand, for R2L every operation bar that involving T is a powering of P and there are faster methods also for those cases [12], [27], [14]. In general one can expect some composite operations in the group G to be cheaper than the sum of the costs of their constituent operations. How well these can be applied seems to determine whether R2L or L2R will be faster since duality shows that the same number of multiplications and squarings occur in either direction. A more efficient Frobenius operation (raising to a power equal to the underlying field characteristic) applies equally well in either direction, and is something that may be taken advantage of in a recoding which favours such powers for either digit or radix.

7 Simulation Results

Sections 5 and 6 describe the best binary and mixed base exponentiation algorithms when only three memory locations are available for storing powers of the input element g. This section compares the approaches. Since the mixed base recoding for an L2R algorithm typically occupies space equal to another element of G, the comparison also includes binary algorithms with an extra location, i.e. four in total, so that the L2R versions can be compared more fairly.

Mixed base recoding is a Markov process with a transition matrix determined by the residue of D_i modulo the lowest common multiple of the base choices. For any recoding algorithm it is therefore straightforward to determine the asymptotic relative frequencies of the various base/digit pairs, and hence the average numbers of squarings, multiplications or any composite operation per exponent bit. The results are given in Table 2 for base set $\mathcal{B} = \{2, 3\}$ and the recoding given in the first paragraph of §6. A better recoding is possible, but improves the speed only marginally. The *#Multns* column combines the numbers of squarings and multiplications for several likely ratios Sq/Mu of their relative costs. The cost of table and other initialisation (such as recoding) is omitted.

The results are divided into two sets according to whether or not inversion is essentially free in the group. This allows application of the conclusions to both the RSA and elliptic curve cases. The speed of the left-to-right compact mixed base algorithm (*Compact*) is compared to the normal table-based mixed base version for a table of size 1 (*MB Table*), as it occupies the same space. The sliding window algorithm of equivalent space has a table of between two and three elements (*SWin*2 and *SWin*3 resp.), depending on whether or not space is needed for the mixed base recoding of D.

The table shows that sliding windows gives the best speed for slightly less space when there is no cheap inversion. However, the compact mixed base algorithm is fastest when inversion is free. Being new[7], it should enable new records to be set for elliptic curve point multiplication on the smallest devices by using combined double-and-add or triple-and-add formulae as described by Longa and Gebotys [21] who only used the traditional table-based version for their record.

6. The reader is reminded that the group G is being represented multiplicatively. So, on an elliptic curve, ''multiplication'' here (and in Table 2) means what is more usually described as ''point addition'' and ''squaring'' means ''point doubling''.

7. At the time of writing, the more detailed account in [34] had not appeared.

$\log_2 D = 160$, *with inversion*			$\log_2 D = 1024$, *without inversion*		
Sq/Mu	**Method**	**#Mults**	**Sq/Mu**	**Method**	**#Mults**
1.0	MB Table	203.9	1.0	MB Table	1415.2
1.0	Compact	190.5	1.0	Compact	1388.7
1.0	SWin2	200.0	1.0	SWin2	1365.3
1.0	SWin3	195.6	1.0	SWin3	1316.6
0.8	MB Table	173.4	0.8	MB Table	1220.6
0.8	Compact	159.6	0.8	Compact	1196.0
0.8	SWin2	168.0	0.8	SWin2	1160.5
0.8	SWin3	163.6	0.8	SWin3	1111.8
0.5	MB Table	126.7	0.5	MB Table	936.7
0.5	Compact	113.2	0.5	Compact	907.2
0.5	SWin2	120.0	0.5	SWin2	853.3
0.5	SWin3	115.6	0.5	SWin3	804.6

TABLE 2
Comparative Speeds for Sliding Window and Mixed Base Exponentiation

8 Conclusion

The most compact versions of mixed base exponentiation have been studied and compared to binary algorithms which use the same space. The comparison was simplified by applying a recent space-preserving duality between left-to-right and right-to-left algorithms which shows that, to a first order, the two directions have the same speed when given the same area. The main differences were identified, namely the space for the recoded exponent and preserving copies of the original exponent and input message, the time reading from and writing to memory, and differential costs for composite group operations in the two directions. Even taking into account the cost of recoding, the mixed base algorithms were shown to be faster where inversion is essentially free, such as in elliptic curve applications. So, in the presence of side channel leakage, the arguably more secure right-to-left direction of exponentiation may be used with little or no speed or area penalty, and perhaps even with some performance gain from not having to store the recoding. The duality gives rise to a compact, left-to-right mixed base algorithm which makes use of efficient composite curve operations and is a few per cent faster than that used recently by Longa and Gebotys [21] to obtain record speeds for scalar multiplication on elliptic curves. The ability to tune the recoding algorithm to make maximal use of cheap composite operations such the Frobenius map means that mixed base algorithms can provide better results in much wider contexts than the classical binary-based exponentiation algorithms.

References

[1] R. M. Avanzi, *A Note on the Signed Sliding Window Integer Recoding and a Left-to-Right Analogue*, SAC 2004, LNCS **3357**, Springer-Verlag 2005, pp. 130–143.

[2] D. J. Bernstein, *Pippenger's Exponentiation Algorithm*, http://cr.yp.to/papers/pippenger.pdf, 2002.

[3] A. Brauer, *On Addition Chains*, Bull. Amer. Math. Soc., **45** (10), 1939, pp. 736–739.

[4] M. Ciet, M. Joye, K. Lauter & P. L. Montgomery, *Trading Inversions for Multiplications in Elliptic Curve Cryptography*, Designs, Codes and Cryptography **39**(2), 2006, pp. 189–206.

[5] H. Cohen, A. Miyaji & T. Ono, *Efficient Elliptic Curve Exponentiation using Mixed Coordinates*, Asiacrypt 1998, LNCS **1514**, Springer-Verlag 1998, pp. 51–65.

[6] R. Dahab & J. Lopez, *An Improvement of the Guajardo-Paar Method for Multiplication on non-Supersingular Elliptic Curves*, Proc. SCCC 1998, IEEE Computer Society 1998, pp. 91–95.

[7] V. S. Dimitrov & T. V. Cooklev, *Two Algorithms for Modular Exponentiation using Nonstandard Arithmetics*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science, **E78-A**, no. 1, 1995, pp. 82–87.

[8] V. S. Dimitrov, G. A. Jullien & W. C. Miller, *Theory and Applications for a Double-Base Number System*, Proc. ARITH 13, IEEE Computer Society, 1997, pp. 44–51.

[9] V. Dimitrov, L. Imbert & P. K. Mishra, *Efficient and Secure Elliptic Curve Point Multiplication using Double Base Chain*, Asiacrypt 2005, LNCS **3788**, Springer-Verlag 2005, pp. 59–79.

[10] K. Eisenträger, K. Lauter & P. L. Montgomery, *Fast Elliptic Curve Arithmetic and Improved Weil Pairing Evaluation*, CT-RSA 2003, LNCS **2612**, Sringer-Verlag 2003, pp. 343–354.

[11] D. M. Gordon, *A Survey of Fast Exponentiation Algorithms*, Journal of Algorithms, **27**, 1998, pp. 129–146.

[12] J. Guajardo & C. Paar, *Efficient Algorithms for Elliptic Curve Cryptosystems*, CRYPTO '97, LNCS **1294**, Springer-Verlag, 1997, pp. 342–356.

[13] K. Itoh, J. Yajima, M. Takenaka & N. Torii, *DPA Countermeasures by Improving the Window Method*, CHES 2002, LNCS **2523**, Springer-Verlag 2002, pp. 303–317.

[14] M. Joye, *Fast Point Multiplication on Elliptic Curves Without Precomputation*, WAIFI 2008, LNCS **5130**, Springer-Verlag, 2008, pp. 36–46.

[15] M. Joye, *Highly Regular m-ary Powering Ladders* Selected Areas in Cryptography (SAC 2009), LNCS **5867**, Springer-Verlag, 2009, pp. 350–363.

[16] M. Joye & S.-M. Yen, *Optimal Left-to-Right Binary Signed Digit Recoding*, IEEE Trans. Comp. **49**(7), 2000, pp. 740–748.

[17] M. Joye & S.-M. Yen, *The Montgomery Powering Ladder*, Proc. CHES 2002, B.S. Kaliski Jr., Ç. Koç & C. Paar, (editors), LNCS **2523**, Springer-Verlag, 2003, pp. 291–302.

[18] D. E. Knuth, *The Art of Computer Programming*, vol. **2**, "Seminumerical Algorithms", 3rd Edition, Addison-Wesley, 1997. ISBN 0-201-89684-2.

[19] Ç. K. Koç, *Analysis of Sliding Window Techniques for Exponentiation*, Computers & Mathematics with Applications **30** (10), 1995, pp. 17–24.

[20] P.-Y. Liardet & N. P. Smart, *Preventing SPA/DPA in ECC Systems using the Jacobi Form*, Proc. CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), LNCS **2162**, Springer-Verlag, 2001, pp. 391–401.

[21] P. Longa & C. Gebotys, *Fast Multibase Methods and Other Several Optimizations for Elliptic Curve Scalar Multiplication*, PKC 2009, LNCS **5443**, Springer-Verlag, 2009, pp. 443–462.

[22] N. Meloni, *New Point Addition Formulae for ECC Applications*, WAIFI 2007, LNCS **4547**, Springer-Verlag, 2007, pp. 189–201.

[23] P. K. Mishra & V. Dimitrov, *Efficient Quintuple Formulas for Elliptic Curves and Efficient Scalar Multiplication using Multibase Number Representation*, Information Security – ISC 2007, LNCS **4779**, Springer-Verlag, 2007, pp. 390–406.

[24] B. Möller, *Improved Techniques for Fast Exponentiation*, Information Security and Cryptology – ICISC 2002, LNCS **2587**, Springer-Verlag, 2003, pp. 298–312.

[25] Peter L. Montgomery, *Speeding the Pollard and elliptic curve methods*

of factorization, Mathematics of Computation **48**(177), pp. 243–264, Jan. 1987.

[26] E. Oswald & M. Aigner, *Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks*, Proc. CHES 2001, LNCS **2162**, Springer-Verlag, 2001, pp. 39–50.

[27] Y. Sakai & K. Sakurai, *On the Power of Multidoubling in Speeding Up Elliptic Curve Multiplication*, SAC 2001, LNCS **2259**, Springer-Verlag, 2001, pp. 268–283.

[28] C. D. Walter, *Exponentiation using Division Chains*, Proc. 13th IEEE Symposium on Computer Arithmetic (ARITH 13), IEEE, 1997, pp. 92–98.

[29] C. D. Walter, *Sliding Windows succumbs to Big Mac Attack*, CHES 2001, LNCS **2162**, Springer-Verlag, 2001, pp. 286–299.

[30] C. D. Walter, *MIST: An Efficient, Randomized Exponentiation Algorithm for Resisting Power Analysis*, CT-RSA 2002, LNCS **2271**, Springer 2002, pp. 53–66.

[31] C. D. Walter, *Some Security Aspects of the MIST Randomized Exponentiation Algorithm*, Proc. CHES 2002, LNCS **2523**, Springer-Verlag, 2003, pp. 276–290.

[32] C. D. Walter, *Longer Keys May Facilitate Side Channel Attacks*, SAC 2003, LNCS **3006**, Springer-Verlag, 2004, pp. 42–57.

[33] C. D. Walter, *Fast Scalar Multiplication for ECC over GF(p) using Division Chains*, Proc. WISA 2010, LNCS **6513**, Springer-Verlag, 2010, pp. 61–75.

[34] C. D. Walter, *A Duality in Space Usage between Left-to-Right and Right-to-Left Exponentiation*, Proc. CT-RSA 2012, LNCS **7178**, Springer-Verlag, 2012, pp. 84–97.

[35] C. D. Walter, *The Montgomery and Joye Powering Ladders are Dual*, IACR ePrint Archive 2017, No. **1081**, 2017.

[36] A. C.-C. Yao, *On the Evaluation of Powers*, SIAM J. Comput. **5**(1), 1976, pp. 100–103.