SPA resistant left-to-right integer recodings

Nicolas Thériault

Department of Combinatorics and Optimization, University of Waterloo

Abstract. We introduce two new left-to-right integer recodings which can be used to perform scalar multiplication with a fixed sequence of operations. These recodings make it possible to have a simple power analysis resistant implementation of a group-based cryptosystem without using unified formulas or introducing dummy operations. This approach is very useful for groups in which the doubling step are less expensive than the addition step, for example with hyperelliptic curves over binary fields or elliptic curves with mixed coordinates.

1 Introduction

Side channel attacks are a constant threat to the implementations of a cryptosystem. This is particularly true for most discrete log based cryptosystems where the basic group operations are often easily distinguishable depending on the nature of their inputs. As a general practice, countermeasures must always be used against simple side channel analysis, even if using one-time keys.

In this paper, we look at the impact of integer recoding for cryptosystems based on the discrete logarithm problem in additive groups. We are particularly interested in groups where the doubling operation is significantly cheaper than the addition (for example hyperelliptic curves over binary fields [18, 19, 10, 14] or elliptic curves with mixed coordinates [8]). For these groups, the standard countermeasures against SPA attacks are particularly disappointing as they remove most of the saving due to efficient implementations of the group operations. Another particularity of many these additive groups (and which we take advantage of) is that the addition and the subtraction operations are almost identical.

We introduce the general situation of scalar multiplication for additive groups in Section 2. In Section 3 we describe some of the basic countermeasures to SPA attacks. We then present the most common forms of integer recoding in Section 4 and introduce our new recodings in Section 5. Finally, we compare the efficiency of the different recodings in Section 6.

2 Scalar multiplication

Many discrete log based public key cryptosystems are done on additive groups and required the multiplication of a group element D by a scalar e (the secret key). It is therefore very important to compute $[e]D$ as efficiently and as securely as possible. This is usually done through a variation of the double-and-add algorithm which relies on two basic group operations: Adding two distinct group elements (addition) and adding a group element to itself (doubling).

2.1 Double-and-add algorithms

For this paper, we consider only the left-to-right version of the double-and-add algorithm (i.e. most significant bit first). Although the right-to-left double-and-add can also be used, the left-to-right version is often more interesting, in particular when combined with integer recodings and applied to a fixed group element.

Given a n bits integer $e = \sum_{j=0}^{n-1} e_j 2^j$ (with $e_j \in \{0,1\}$), let $f_{n-i} = \sum_{j=i}^{n-1} e_j 2^{j-(n-i)}$, i.e. the number formed by the $n-i$ most significant bits of the binary expansion of e. Then f_{n-i} can be obtained from f_{n-i-1} and e_i via the relation $f_{n-i} = 2f_{n-i-1} + e_i$. In terms of scalar multiplication, this becomes: $[f_{n-i}]D = [2]([f_{n-i-1}]D) + [e_i]D$. The left-to-right double-and-add algorithm follows easily from this relation and its general form proceeds as in Algorithm 1. This algorithm is written in its most general form to cover most of the cases encountered in this paper. If we consider only the "classical" double-and-add algorithm on the binary representation, there is no recoding step, no precomputation ($[1]D$ is already known) and the addition step when $r_j \neq 0$ is simply $D_0 + D_1$.

Algorithm 1: Generic double-and-add algorithm

```
Input: $D$, $e$
Output: $[e]D$
recode $e$ as $\sum_{j=0}^{m} r_j 2^j$                 recoding
precompute $[r]D$ for every digit $r \neq 0$     precomputations
$D_0 \leftarrow [r_m]D$
for $j = m-1$ down to 0 do
  $D_0 \leftarrow [2]D_0$                              doubling
  if $r_j \neq 0$ then
    $D_0 \leftarrow D_0 + [r_j]D$                      addition
return $D_0$
```

3 Simple side channel analysis attacks

Power traces [13] and electromagnetic emissions of processors [1] can be used as sources of information for simple side channel analysis (we will refer to both as SPA attacks for simplicity) SPA attacks may exploit even small differences between the addition and the doubling operations on group elements to discover the sequence in which they are used in the double-and-add algorithm. If successful, this gives the binary expansion of e, hence the secret key. It is therefore essential to secure implementations of public key cryptosystems against this type of attack.

As a general rule, countermeasures against simple side channel attacks do not secure the encryption against differential power analysis (DPA). If differential side channel attacks are potential a threat, i.e. if the scalar is used more than once, this problem can be resolved by combining SPA and DPA countermeasure when possible (see [3] for details). On the other hand, DPA countermeasures are useless if the encryption is insecure against simple side channel attacks, so SPA countermeasures should always be used.

3.1 Standard countermeasures

There are two standard countermeasures against SPA attacks: Dummy operations and unified formulas. Both approach attempt to make the power traces of the two group operations (addition and doubling) indistinguishable.

The first approach consists in adding extra or “dummy” operations in the addition and doubling algorithms where the sequences of operations differ [9]. The result is an addition and a doubling formula which use the same sequence of operations, so they will appear identical to SPA attacks. Obviously this will increase the cost of the group operations (or at least the cheapest of the two), having a negative impact on the efficiency of the encryption algorithm.

Although this is a very simple countermeasure to implement, it is not always safe: If the secret key is used multiple times, dummy operations can be revealed by adaptive fault analysis [24, 25], and further countermeasures are required to prevent this attack.

The second approach consist in rewriting the two group operations into a unified formula. Since both operations will then use the same set of operations, the two operations will have the same power trace.

Unified formulas tend to be more costly to use than dummy operations, but they prevent adaptive fault analysis (but not DPA). The main disadvantage of unified formulas is that they are group specific and so far they have only been developed for elliptic curves [11, 15, 5, 4, 6].

Moreover, some of these formulas have been shown to be weak because some field multiplications are performed twice with the same inputs in the doubling formula but not in the addition formula, making the system potentially vulnerable [23].

3.2 Montgomery ladders

Another countermeasure against SPA is the use of a Montgomery ladder [12]. The algorithm proceeds from left-to-right, computing two elements at each step: $[f_j]D$ and $[f_j+1]D$, where f_j is the partial sum of the $n-j$ most significant bits of e, i.e. $f_j = \sum_{i=j}^{n} e_i 2^{i-j}$.

Since $f_j = 2f_{j+1} + e_j$, the pair (f_j, f_j+1) can be obtained from the pair $(f_{j+1}, f_{j+1}+1)$ (computed at the previous step) using the rules:

e_j	f_j	f_j+1
0	$2f_{j+1}$	$f_{j+1} + (f_{j+1}+1)$
1	$f_{j+1} + (f_{j+1}+1)$	$2(f_{j+1}+1)$

which gives Algorithm 2 for scalar multiplication (where $D_0 = [f_j]D$ and $D_1 = [f_j+1]D$). Since all the steps use the same set of operations the two group operations do not have to be secured against SPA attacks. As no dummy operations are introduced, the risk posed by adaptive fault analysis is minimal.

Algorithm 2: Montgomery ladder

```
Input: $D$, $e = \sum_{i=0}^{n} e_i 2^i$
Output: $[e]D$
$D_0 \leftarrow 0$; $D_1 \leftarrow D$
for $j = n$ down to 0 do
  if $e_j = 0$ then
    $D_1 \leftarrow D_0 + D_1$; $D_0 \leftarrow 2D_0$                    $e_j = 0$
  else
    $D_0 \leftarrow D_0 + D_1$; $D_1 \leftarrow 2D_1$                    $e_j = 1$
return $D_0$
```

One drawback of the Montgomery ladder is the high count of group operations since every step requires one doubling and one addition. Since at any given step the two group operations are independent from each other, it is sometimes possible to offset part of the high operation count by combining the them. For example, with elliptic curves in affine coordinates it is possible to combine the field inversions of the group addition and doubling into one field inversion and three field multiplication. Unfortunately, for most groups used in cryptographic applications this approach is unlikely to give enough savings to justify using a Montgomery ladder instead of other SPA countermeasures.

4 Integer recoding

A common approach to improve the efficiency of the scalar multiplication is to use integer recoding to reduce the number of operations required in the double-and-add algorithm. By allowing integers other than 0 or 1 to be used in the expansion of e, it becomes possible to *recode* $e = \sum_{i=0}^{n} e_i 2^i$ as $e = \sum_{i=0}^{n'} r_i 2^i$.

The double-and-add algorithm will still work as in Algorithm 1, but the term added may now be different from D. If the *weight* (number of non-zero digits) of the recoding of e is smaller than the weight of its binary expansion, then Algorithm 1 will require fewer additions to compute $[e]D$ (and possibly fewer doublings if $n' < n$).

The main difference is that unlike the double-and-add algorithm on the binary representation, the elements $[s]D$ must be precomputed for all the possible digits s.

Most recodings can be divided into two categories depending on the order in which the bits are processed. Right-to-left recodings, i.e. from the least significant bit to the most significant one, are more natural but they must be computed before the double-and-add algorithm and the storage required for the recoding is usually greater than that of the integer itself. Left-to-right recodings are computed from the most significant bits down to the least significant bit, hence the recoding can be done at the same time as the left-to-right double-and-add multiplication, avoiding the need to record the new representation. This makes left-to-right recodings somewhat more interesting for implementations in restricted environments, especially if the group element is fixed (so the precomputed values $[s]D$ for the digits s can be reused for a number of scalar multiplications).

For a number of groups used in cryptography, and in particular for Elliptic and Jacobians of hyperelliptic curves, recodings can take advantage of symmetries. For these groups, the group subtraction is almost identical to the group addition, up to a few sign changes in the field operations. Since field additions and subtractions are indistinguishable under SPA attacks, the performance and security of the cryptosystem are unaffected if we subtract $[s]D$ instead of adding $[-s]D$, but the storage requirement for the double and add algorithm can be reduced. This makes it very interesting to use digit sets which are symmetric around 0 since only half of the points must be precomputed (for example those corresponding to the positive digits).

4.1 Recodings and SPA attacks

In general, SPA attacks are much less effective on double-and-add algorithms using integer recodings than those using the binary representation directly. From the power trace of the double-and-add algorithm, it is possible to know which digits in the recoding are non-zero, but not their values.

If the recoding has a *density* (weight of the recoding divided by its length) which is too low or if it contains long sequences of zero digits, the attacker may be able to restrict the portion of the keyspace the secret key could be in. The size of the keyspace to consider may then become small enough for the attacker to find the key using other methods (for example Shanks' Baby-step Giant-step algorithm, Pollard's Rho algorithm, etc). When this is the case, the implementation of the double-and-add algorithm must also include a countermeasure against SPA attacks (see Section 3).

On the other hand, if the weight of the representation is high enough and the non-zero digits are distributed uniformly enough, the recoding is inherently secure and act as a SPA countermeasures. This is the idea behind the fixed recodings in Subsection 4.4 and Section 5.

4.2 w-NAF

The most commonly used recodings are the Non Adjacent Form (NAF) [21] and its extension the w-NAF [7, 22]. For this paper, we will denote the w-NAF as using the digit set $\{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\} \cup \{0\}$ and such that any pairs of non-zero digits are separated by at least w zeros. This is also called the $(w-1)$-NAF and sometimes denoted NAF_{w-1}. The w-NAF recoding is computed from left to right and has average density $1/(w+2)$.

To use the negative digits, we use consider sequences of up to w bits and a *carry* c_j (just as in a base 2 addition). To a sequence of w bits starting from the j-th bit of e, we associate the integer $s_j = \sum_{i=0}^{w} e_{i+j} 2^i$. Starting with $j = 0$ and $c_0 = 0$, each step of the recoding follows the rules

$e_j + c_j$	$s_j + c_j$	k	c_{j+k}	r_j	$r_{j+1}, \ldots, r_{j+k-1}$
0	–	1	0	0	–
2	–	1	1	0	–
1	$< 2^w$	$w+1$	0	$s_j + c_j$	0
1	$> 2^w$	$w+1$	1	$s_j + c_j - 2^{w+1}$	0

where the next bit to be encoded is the $(j + k)$-th bit of the binary representation.

Although the w-NAF gives a recoding of the smallest possible weight for the given digit set (see [2]), which is advantageous for the performance of the encryption, the key is weakened by the low density and by the knowledge of the variable positions of the non-zero digits. Since there are 2^w possible values for the non-zero digits and the recodings have an average density of $1/(w + 2)$, there are (on average) $2^{wn/(w+2)}$ keys of n bits with a given sequence of doublings and additions. Compared to the 2^n possible keys of length n, we get a reduction by a factor of $2^{2n/(w+2)}$ in the number of possible keys. We can see that unless SPA countermeasures are used, the w-NAF is not intended for applications such as restricted environments which are susceptible of side channel attacks.

4.3 Minimal weight left-to-right recoding

Avanzi [2] and Muir and Stinson [17] developed left-to-right equivalents of the w-NAF. This recoding gives the same advantage as the w-NAF, i.e. it gives a recoding of minimal weight for the digit set, but with the added bonus that it proceeds from the most significant bit downward so it can be interleaved with the left-to-right scalar multiplication.

Let $v_{j,k} = s_{j-k} + e_{j-k} - e_j 2^k$ and let $t_{j,k}$ be the highest power of 2 dividing $v_{j,k}$, then the recoding step follows the rule:

$e_j - e_{j-1}$	k	$r_j, \ldots, r_{j-k+1}$	r_{j-k+t}
0	1	0	–
± 1	$\min\{w, j+1\}$	0	$v_{j,k}/2^t_{j,k}$

where the next recoding step is for the bit $j - k$ (and lower). We refer to [2] and [17] for the proof of correctness of the recoding process.

As was the case with the w-NAF, the group operations will also have to be secured against SPA attacks.

4.4 Fixed right-to-left recoding

In [16], Möller introduced a new fixed right-to-left recoding. The idea consists in computing a 2^w-ary expansion of e, but in such a way that none of the digits are 0 (hence producing a “regular” or “fixed” expansion).

Since a 2^w-ary recoding requires a set of at least 2^w digits in order to represent every possible integer, the digit 0 is replaced by -2^w, i.e. the digit set is $\{1, 2, 3, \ldots, 2^w - 1\} \cup \{-2^w\}$. As with the w-NAF, we need to introduce a carry in order to do the recoding but, in order to cover all the possible situations, it can take the values 0, 1 and 2. The recoding goes from right to left by blocks of w bits, starting with a carry of 0.

Given $s_j = \sum_{i=0}^{w-1} e_{i+wj} 2^i$, the recoding steps follows the rule:

$s_j + c_j$	r_j	c_{j+1}
0	-2^w	1
2^w	-2^w	2
$2^w + 1$	1	1
other values	$s_j + c_j$	0

Once the scalar is recoded (and stored), the scalar multiplication works very much like a left-to-right "2^w and add" algorithm on the recoding. Rather than computing $[2]D_0$ (where D_0 is the partial sum at the previous step of the scalar multiplication) and then adding $[e_j]D$, the algorithm computes $[2^w]D_0$ (by doubling w times) and then adds $[r_j]D$.

Since the sequence of doublings and additions is fixed and is the same for all integers of the same size, this recoding is resistant against SPA attacks and the fastest implementations of the group operations can be used even if they are very unbalanced.

A side effect of this approach is that even leading zero digits can (and will) be recoded as non-zero. The length of the recoding must then be decided beforehand – usually to fit the longest possible key – with the added bonus that short scalars are indistinguishable from longer ones.

5 Fixed left-to-right recodings

The two main disadvantages of Möller's recoding algorithm are that it is right-to-left (so it must be computed and stored before the scalar multiplication) and the digit set does not really take advantage of symmetries. In this section, we introduce two new fixed recodings which

In order to have a left-to-right recoding (which can be interleaved with the scalar multiplication) and to use symmetries to save space, we must introduce new digit sets which are symmetric around 0.

Since the recoding goes from the highest powers of 2^w down to the lowest, the carry will not behave as usual: Instead of delaying the addition of 2^w and replacing it by the addition of 1 at the next (higher) power of 2^w, the carry (if different from 0) will delay the subtraction of 1 and replace it by the subtraction of 2^w at the next (lower) power of 2^w. For simplicity, the values of the carry will still be denoted 0 and 1 as in the w-NAF, but with the understanding that it has the new meaning.

To simplify the notation, we define s_j as $\sum_{i=0}^{w-1} e_{i+wj}2^i$: The coefficient of 2^{wj} in the 2^w-ary expansion (using the digit set $\{0, 1, \ldots, 2^w - 1\}$). As was the case with the fixed right-to-left recoding, the length of the representation must be decided beforehand.

5.1 General case

We first consider which digit set could be used for the recoding. The argument used here is by no means the only one possible and there are indeed other digit sets that will give a valid recoding.

As the carry is done downward, we must be able to recode all the possible values of s_j and $s_j - 2^w$, i.e. all the integers in $[-2^w, 2^w - 1]$. Since the introduction of a carry of one to the next (lower) power of 2^w will increase the current coefficient by 1, the possible values (after the carry) are $-2^w, -(2^w - 1), \ldots, 2^w - 1, 2^w$, so the set of even integers $0, \pm 2, \pm 4, \ldots, \pm 2^w$ seems like a reasonable choice. However, we want to remove the possibility of a zero digit in the 2^w-ary expansion, and since the carry is either 0 or 1, the only possible choice for the recoding of 0 is 1 (with a new carry of 1), so ± 1 must also be allowed as digits instead of 0 (-1 is also necessary since it cannot be recoded as 0 with a new carry of 1). Bringing all this together, we obtain the digit set $\{\pm 1\} \cup \{\pm 2, \pm 4, \pm 6, \ldots, \pm 2^w\}$.

If with start with a carry of 0 for the leftmost bit recoded, the general recoding rule can be written as follows:

$s_j - c_j 2^w$	r_j	c_{j-1}
even, $\neq 0$	$s_j - c_j 2^w$	0
0	1	1
odd, $\neq -1$	$(s_j - c_j 2^w) + 1$	1
-1	-1	0

It is easy to verify that at every step of the recoding $r_j = s_j - c_j 2^w + c_{j-1}$, so that $\sum_{j=0}^{m} r_j 2^{wj} = \sum_{j=0}^{m} s_j 2^{wj} - 2^{(m+1)w} c_m + c_{-1} = e + c_{-1}$.

Remark: With the residue system $\{\pm 1\} \cup \{\pm 2, \pm 4, \ldots, \pm 2^w\}$, there are multiple choices for the recodings of -2 and 1:

- -2 can be recoded as -2 without a carry, or as -1 with a carry;
- 1 can be recoded as 1 without a carry, or as 2 with a carry.

The recodings rules given above were chosen for simplicity.

Since the recoding goes from left to right, the final recoding step takes place at the w least significant bits. We could use the same recoding system for the final step, but one must then decide what to do if there is a carry after that step (a "rightward" carry at the unit level would require a fractionnal expansion, which is incompatible with the scalar multiplication). One solution consists in taking the result obtained at the final step and apply a "$-c_{-1}$" directly to it (without any extra doubling), essentially applying the carry directly on the integer instead of delaying it. But in the case of a SPA attack, this would essentially reveal the final bit (with the possible exception of a recoding of 0 or -1).

A better alternative consists in replacing the final recoding step so that the final step of the encryption always consists of *two* additions (with $r_0 + r'_0 = s_0 - c_0 2^w$):

$s_0 - c_0 2^w$	r_0	r'_0
even, $\neq -2$	$(s_0 - c_0 2^w) + 2$	-2
-2	-1	-1
odd, $\neq -1$	$(s_0 - c_0 2^w) + 1$	-1
-1	1	-2

Remark: With the exceptions of ± 1 (and -4 if $w = 2$), there exists multiple choices for the recodings of all the possible values of $s_0 - c_0 2^w$. The recodings rules given above were chosen for simplicity.

The computation of the scalar multiplication proceeds as in Algorithm 1 (with w doublings between every two additions since we have a fixed 2^w-ary expansion, as in Subsection 4.4) except for the final step which becomes: $D_0 \leftarrow D_0 + [r_0]D + [r'_0]D$.

5.2 Groups of odd order

The digit set $\{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$ might be a more natural choice: Since the carry produces a shift of $+1$ on s_j, this is the smallest symmetric set of integer not containing 0 for which all possible values of s_j and $s_j - 2^w$ (to take into account the previous carry) can be recoded using a carry of either 0 or 1. With this digit set, the general recoding step is described by the following rule:

s_j	r_j	c_{j-1}
even	$s_j + 1 - c_j 2^w$	1
odd	$s_j - c_j 2^w$	0

Once again, it is easy to verify that at every step $r_j = s_j - c_j 2^w + c_{j-1}$, so that $\sum_{j=0}^{m} r_j 2^{wj} = e + c_{-1}$. A nice aspect of this rule is that for $j < m$ it can be rewritten as: $r_j = 1 - 2^w + \sum_{i=1}^{w} e_{i+wj} 2^i$ (with $c_{j-1} = 1 - e_{wj}$), making it very straightforward to implement and requires no conditional statement.

If $e_0 = 1$ (e is odd), we get $c_{-1} = 0$ so there are no carries into negative powers of 2 (or 2^w) and the the recoding terminates correctly, but if $e_0 = 0$ (e is even), we get $c_{-1} = 1$ and the recoding is no longer an integer. Unlike the previous recoding, it is not possible to replace the final step and carry by a fixed number of operations: The parity of a sum of digits depends only on the number of additions, not which digits are added. Although this problem cannot be fixed in general, it can be avoided in most cryptographic applications.

From a cryptographic point of view, there is no disadvantage to consider that the order of the group used is a large prime. This is because the discrete logarithm problem in a group can be reduced to the discrete log problem in its subgroups using the Chinese Remainder Theorem [20]. We can therefore make the assumption that the group in which the scalar multiplication is done has odd order.

Under this condition, it is always possible to force the secret key to be an odd integer: If e is even, it can be replaced by $e' = e + \#G$ (since $[e']D = [e]D$). Since can ensure the scalar is odd, the left-to-right recoding using digits $\pm 1, \pm 3, \ldots, \pm(2^w - 1)$ will have a final carry (c_{-1}) equal to zero and the recoding will always terminate correctly. Interleaving the recoding and the scalar multiplication gives us Algorithm 3.

Algorithm 3: fixed left-to-right scalar multiplication (interleaved)

```
Input: $D$, $w$, $e = 1 + \sum_{i=1}^{wm} e_i 2^i$ (odd)
Output: $[e]D$
precompute $[1]D, [3]D, \ldots, [2^w - 1]D$
$r_j \leftarrow 1 + \sum_{i=1}^{w-1} e_{i+wm} 2^i$          recoding
$D_0 \leftarrow [r_m]D$
for $j = m - 1$ down to 0 do
  for $k = 0$ to $w - 1$ do
    $D_0 \leftarrow [2]D_0$                                  $w$ doublings
  $r_j \leftarrow 1 - 2^w + \sum_{i=1}^{w} e_{i+wj} 2^i$      recoding
  $D_0 \leftarrow D_0 + [r_j]D$                              addition
return $D_0$
```

6 Performance comparison

We can now summarize and compare the efficiency of the different scalar multiplication and recoding algorithms to get a better idea of which ones are more interesting depending on the situation. To compare equivalent security levels, we assume that SPA countermeasures (for example unified formulas) are used on the group operations in the cases where SPA attacks could reveal even partial information on the secret key.

6.1 Unrestricted environment

We first consider the case of applications where there is no restriction on the memory used by the algorithm and where the group element is assumed fixed for every scalar multiplication while the scalar varies. Under these conditions, we can assume that the precomputations are already done when the double-and-add algorithm is used, so their cost does not have to be taken into account. To have a common basis for the comparison, we assume that the recodings all have the same (average) density of $1/t$, with the exception of the double-and-add algorithm on the binary representation (average density of $1/2$) and the Montgomery ladders (density of 1).

We express the costs as "group operations (on average) per bit of the scalar". We denote by r the cost (in normal group addition) of an optimized group doubling, and by c (≥ 1) the cost of indistinguishable group operations (either using uniform formulas or dummy operations).

By memory, we mean the number of precomputed elements which must be in memory for the double-and-add algorithms, including $[1]D$. Since montgomery ladders do not require any precomputations but compute two group elements instead of one, we write its memory requirement as 1.

We get the following table:

method	section	w	cost	memory	direction
double-and-add	2.1	1	$\frac{3}{2}c$	1	left-to-right
Montgomery Ladder	3.2	1	$r+1$	1	left-to-right
w-NAF	4.2	$t-2$	$c(1+\frac{1}{t})$	2^{t-3}	right-to-left
minimal LtoR	4.3	$t-2$	$c(1+\frac{1}{t})$	2^{t-3}	left-to-right
Möller	4.4	t	$r+\frac{1}{t}$	2^t	right-to-left
fixed LtoR (general)	5.1	t	$r+\frac{1}{t}$	$2^{t-1}+1$	left-to-right
fixed LtoR (odd order)	5.2	t	$r+\frac{1}{t}$	2^{t-1}	left-to-right

We can see that for the same density, the fixed left-to-right recoding requires four times as much memory and precomputations than the w-NAF (a little more in the case of general group orders), while Möller's fixed right-to-left recoding requires eight times as much as the w-NAF. If $r < (ct + c - 1)/t$, the three fixed recodings are more efficient, but if $r > (ct + c - 1)/t$, the w-NAF and the minimal weight left-to-right recoding become more efficient.

6.2 Restricted memory

In some applications (such as restricted environments and implementations where the secret key is used more than once but on different group elements), it is really unfair to compare recodings which require different number of precomputations. The easiest way to compare the different recodings in these situations is to assume that a fixed number of precomputations are done (here we use 2^t) and compare the cost of the multiplications without taking into account the precomputation cost (which is the same for all the recodings, even thought they use different digit sets).

To make the comparisons uniform, we do not consider the double-and-add on the binary expansion and Montgomery ladders. We also remove the general case of the fixed left-to-right recoding (which requires one more precomputation), restricting ourselves to the case of groups of odd order. Using the same notation as in the previous subsection, we find:

method	section	w	average density	cost	direction
w-NAF	4.2	$t+1$	$\frac{1}{t+3}$	$c(1+\frac{1}{t+3})$	right-to-left
minimal LtoR	4.3	$t+1$	$\frac{1}{t+3}$	$c(1+\frac{1}{t+3})$	left-to-right
Möller	4.4	t	$\frac{1}{t}$	$r+\frac{1}{t}$	right-to-left
fixed LtoR	5.2	$t+1$	$\frac{1}{t+1}$	$r+\frac{1}{t+1}$	left-to-right

This time the comparisons are much more clearly delimited. If we let $\gamma = c\left(1+\frac{1}{t+3}\right) - \frac{1}{t+1}$, we get the following rules:

- The fixed left-to-right recoding always gives a saving compared to Möller's fixed right-to-left recoding.
- If $r < \gamma$, the fixed left-to-right recoding is faster than the w-NAF or the minimal weight left-to-right recoding.
- If $r > \gamma$, the w-NAF and the minimal weight left-to-right recoding are faster than the fixed left-to-right recoding, even though SPA countermeasures must be added in the implementation of these algorithms.

7 Conclusion

We presented two new integers recodings which are resistant to SPA attacks. These recodings are left-to-right so they can be interleaved with a left-to-right scalar multiplication, removing the need to store both the scalar and its recoding. In groups where the doubling operations can be implemented with significant savings compared to a group addition, the new algorithms become faster than a w-NAF (or its left-to-right equivalent) which has been secured against SPA attacks. It should be kept in mind that these implementation do not ensure in any way the security against differential side channel analysis, so countermeasures against these attacks should also be used if the secret key is used more than once.

Acknowledgements

The author would like to thank Roberto Avanzi for his useful comments and remarks.

References

1. D. Agrawal, B. Archambeault, J.R. Rao, and P. Rohatgi. The em side-channel(s). In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 29–45. Springer–Verlag, 2003.
2. R.M. Avanzi. A note on the signed sliding window integer recoding and a left-to-right analogue. In H. Handschuh and M.A. Hasan, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *LNCS*, pages 130–143. Springer–Verlag, 2005.
3. R.M. Avanzi. Side channel attacks on implementations of curve-based cryptographic primitives. Cryptology ePrint Archive, Report 2005/017, 2005. Available at: `<http://eprint.iacr.org/>`.
4. O. Billet and M. Joye. The jacobi model of an elliptic curve and side-channel analysis. In M. Fossorier, T. Høholdt, and A. Poli, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes – AAECC-15*, volume 2643 of *LNCS*, pages 34–42. Springer–Verlag, 2003.
5. É. Brier and M. Joye. Weierstraßelliptic curves and side-channel attacks. In D. Naccache and P. Paillier, editors, *Public Key Cryptography – PKC 2002*, volume 2274 of *LNCS*, pages 335–345. Springer–Verlag, 2002.
6. É. Brier, M. Joye, and I. Déchène. Unified point addition formulæfor elliptic curve cryptosystems. In N. Nedjah and L. de Macedo Mourelle, editors, *Embedded Cryptographic Hardware: Methodologies & Architectures.* Nova Science Publishers, 2004.
7. H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation. In *ICICS'97*, volume 1334 of *LNCS*, pages 282–290. Springer–Verlag, 1997.
8. H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In K. Ohta and D. Pei, editors, *Advances in Cryptology - ASIACRYPT'98*, volume 1514 of *LNCS*, pages 51–65. Springer–Verlag, 1998.
9. J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99*, volume 1717 of *LNCS*, pages 292–302. Springer–Verlag, 1999.
10. C. Guyot, K. Kaveh, and V. Patankar. Explicit algorithm for the arithmetic on the hyperelliptic jacobians of genus 3. *J. Ramanujan Math. Soc.*, 19(2):75–115, 2004.

11. M. Joye and J.-J. Quisquater. Hessian elliptic curves and side-channel attacks. In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *LNCS*, pages 402–410. Springer–Verlag, 2001.
12. M. Joye and S.-M. Yen. The montgomery powering ladder. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 291–302. Springer–Verlag, 2003.
13. P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer–Verlag, 1999.
14. T. Lange and M. Stevens. Efficient doubling on genus two curves over binary fields. In H. Handschuh and M.A. Hasan, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *LNCS*, pages 170–181. Springer–Verlag, 2005.
15. P.-Y. Liardet and N.P. Smart. Preventing spa/dpa in ecc systems using the jacobi form. In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *LNCS*, pages 391–401. Springer–Verlag, 2001.
16. B. Möller. Securing elliptic curve point multiplication against side-channel attacks. In G.I. Davida and Y. Frankel, editors, *Information Security: 4th International Conference – ISC 2001*, volume 2200 of *LNCS*, pages 324–334. Springer–Verlag, 2001.
17. J. Muir and D. Stinson. New minimal weight representations for left-to-right window methods. CACR Technical Report, CORR 2004-19, 2004. Available at: `<http://www.cacr.math.uwaterloo.ca/techreports/2004/corr2004-19.pdf>`.
18. J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar. Hyperelliptic curve cryptosystems: Closing the performance gap to elliptic curves. In *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *LNCS*, pages 351–365. Springer–Verlag, 2003.
19. J. Pelzl, T. Wollinger, and C. Paar. Low cost security: Explicit formulae for genus-4 hyperelliptic curves. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2003*, volume 3006 of *LNCS*, pages 1–16. Springer–Verlag, 2004.
20. S.C. Pohlig and M.E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Trans. Information Theory*, 24(1):106–110, 1978.
21. G.W. Reitwiesner. Binary arithmetic. In *Advances in computers*, volume 1, pages 231–308. Academic Press, New York, 1960.
22. J.A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In B.S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *LNCS*, pages 357–371. Springer–Verlag, 1997.
23. C.D. Walter. Simple power analysis of unified code for ecc double and add. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *LNCS*, pages 191–204. Springer–Verlag, 2004.
24. S.-M. Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. on Computers*, 49(9):967–970, Sept. 2000.
25. S.-M. Yen, S. Kim, S. Lim, and S. Moon. A countermeasure against one physical cryptanalysis may benefit another attack. In K. Kim, editor, *Information Security and Cryptology – ICISC 2001*, volume 2288 of *LNCS*, pages 414–427. Springer–Verlag, 2002.