Main text appears in A. H. Chan, V. Gligor (Eds.): *Information Security – ISC 2002,* Springer-Verlag LNCS 2433, pp. 402–413, ISBN 3-540-44270-7.

Notes added 2002-10-16.

Parallelizable Elliptic Curve Point Multiplication Method with Resistance against Side-Channel Attacks

Bodo Möller

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

Abstract. We present a new 2^w-ary elliptic curve point multiplication method with resistance against side-channel attacks. This method provides two advantages compared with previous similar side-channel attack countermeasures: It avoids a fixed table, thus reducing potential information leakage available to adversaries; and it is easily parallelizable on two-processor systems, where it provides much improved performance.

1 Introduction

Implementations of elliptic curve cryptosystems may be vulnerable to *side-channel attacks* ([12], [13]) where adversaries can use power consumption measurements or similar observations to derive information on secret scalars e in point multiplications eP. One distinguishes between *differential side-channel attacks,* which require correlated measurements from multiple point multiplications, and *simple side-channel attacks,* which directly interpret data obtained during a single point multiplication.

Randomization can be used as a countermeasure against differential side-channel attacks. In particular, for elliptic curve cryptography, *projective randomization* is a simple and effective tool [5]: If (X, Y, Z) represents the point whose affine coordinates are $(X/Z^2, Y/Z^3)$, another representation of the same point that cannot be predicted by the adversary is obtained by substituting (r^2X, r^3Y, rZ) with a randomly chosen secret non-zero field element r. (When starting from an affine representation (X, Y), this simplifies to (r^2X, r^3Y, r).)

Simple side-channel attacks can be easy to perform because usually the attacker can tell apart point doublings from general point additions. Thus point multiplication should be implemented using a fixed sequence of point operations that does not depend on the particular scalar. Note that it is reasonable to assume that point addition and point subtraction are uniform to the attacker as point inversion is nearly immediate (dummy inversions can be inserted to obtain the same sequence of operations for point additions as for point subtractions).

Various point multiplication methods have been proposed that use an alternating sequence of doublings and additions: The simplest approach is to use a binary point multiplication method with dummy additions inserted to avoid

dependencies on scalar bits [5]; however as noted in [15] it may be easy for adversaries to determine which additions are dummy operations, so it is not clear that this method provides sufficient security. For odd scalars, a variant of binary point multiplication can be used where the scalar is represented in balanced binary representation (digits -1 and $+1$) [22]. Also Montgomery's binary point multiplication method [17], which maintains an invariant $Q_1 - Q_0 = P$ while computing eP using two variables Q_0, Q_1, can be adapted for implementing point multiplication with a fixed sequence of point operations ([23], [18], [1], [9], [6]). With this approach, specific techniques can be used to speed up point arithmetic: The doubling and addition steps can be combined; y-coordinates of points may be omitted during the computation ([17], [1], [9], [6]); and on suitable hardware, parallel execution can be conveniently used for improved efficiency ([9], [6]).

All of the above point multiplication methods are binary. Given sufficient memory, efficiency can be improved by using 2^w-ary point multiplication methods. Here, the scalar e is represented in base 2^w using digits b_i from some digit set B:

$$e = \sum_{0 \le i \le \ell} b_i 2^{wi}$$

A simple way to obtain a uniform sequence of doublings and additions (namely, one addition after w doublings in the main loop of the point multiplication algorithm) is to use 2^w-ary point multiplication as usual (first compute and store bP for each $b \in B$, then compute eP using this precomputed table), but to insert a dummy addition whenever a zero digit is encountered. However, as noted above for the binary case, the dummy addition approach may not be secure. This problem can be avoided (given $w \ge 2$) by using a representation of e without digit value 0, such as

$$B = \{-2^w, 1, 2, \ldots, 2^w - 1\}$$

as proposed in [15], or

$$B = \left\{-2^w, \pm 1, \pm 2, \ldots, \pm(2^w - 2), 2^w - 1\right\}$$

for improved efficiency as proposed in [16]. A remaining problem in the method of [15] and [16] is that the use of a fixed table may allow for statistical attacks: If the same point from the table is used in a point addition whenever the same digit value occurs, this may help adversaries to find out which of the digits b_i have the same value (cf. the attacks on modular exponentiation using fixed tables in [24] and [21]). This problem can be countered by performing, whenever the table is accessed, a projective randomization of the table value that has been used. This will avoid a fixed table, but at the price of reduced efficiency.

In this paper, we present a new variant of 2^w-ary point multiplication with resistance against side-channel attacks that avoids a fixed table without requiring frequently repeated projective randomization. An additional advantage of the new method is that it is easily parallelizable on two-processor systems. The essential change in strategy compared with earlier methods for side-channel attack resistant point multiplication is that we use a right-to-left method (the scalar

is processed starting at the least significant digit, cf. [25]) whereas the conventional methods work in a left-to-right fashion. Section 2 describes the new point multiplication method. Section 3 examines its efficiency in comparison with the left-to-right method of [15] and [16]. In section 4, we describe some possible variants. Section 5 summarizes our conclusions.

2 Description of the Point Multiplication Method

Our method for computing eP is parameterized by an integer $w \geq 2$ and a digit set B consisting of 2^w integers of small absolute value such that every positive scalar e can be represented in the form

$$e = \sum_{0 \leq i \leq \ell} b_i 2^{wi}$$

using digits $b_i \in B$; for example

$$B = \{0, 1, \ldots, 2^w - 1\}$$

or

$$B = \{-2^{w-1}, \ldots, 2^{w-1} - 1\}.$$

A representation of e using the latter digit set can be easily determined on the fly when scanning the binary digits of e in right-to-left direction. If e is at most n bits long (i.e. $0 < e < 2^n$), $\ell = \lfloor n/w \rfloor$ is sufficient.

Let B' denote the set $\{|b| \mid b \in B\}$ of absolute values of digits, which has at least $2^{w-1} + 1$ and at most 2^w elements. The point multiplication method uses $\#(B') + 1$ variables for storing points on the elliptic curve in projective representation: Namely, one variable A_b for each $b \in B'$, and one additional variable Q.

The method works in three stages, which we call *initialization stage, right-to-left stage,* and *result stage.* We will first give a high-level view of these stages before discussing the details. Let A_b^{init} denote the value of A_b at the end of the initialization stage, and let A_b^{sum} denote the value of A_b at the end of the right-to-left stage.

The *initialization stage* sets up the variables A_b ($b \in B'$) in a randomized way such that $A_b^{\mathsf{init}} \neq \mathcal{O}$ for each b, but

$$\sum_{b \in B'} b A_b^{\mathsf{init}} = \mathcal{O}.$$

($\mathcal{O}$ denotes the point at infinity, the neutral element of the elliptic curve group.) Then the *right-to-left stage* performs computations depending on P and the digits b_i, yielding new values A_b^{sum} of the variables A_b satisfying

$$A_b^{\mathsf{sum}} \;=\; A_b^{\mathsf{init}} + \sum_{\substack{0 \leq i \leq \ell \\ b_i = b}} 2^{wi} P - \sum_{\substack{0 \leq i \leq \ell \\ b_i = -b}} 2^{wi} P$$

for each $b \in B'$. Finally, the *result stage* computes

$$\sum_{b \in B' \setminus \{0\}} bA_b^{\mathsf{sum}},$$

which yields the final result eP because

$$\begin{aligned}\sum_{b \in B' \setminus \{0\}} bA_b^{\mathsf{sum}} &= \underbrace{\sum_{b \in B' \setminus \{0\}} bA_b^{\mathsf{init}}}_{\mathcal{O}} + \sum_{b \in B' \setminus \{0\}} b\left(\sum_{\substack{0 \le i \le \ell \\ b_i = b}} 2^{wi} P - \sum_{\substack{0 \le i \le \ell \\ b_i = -b}} 2^{wi} P \right) \\ &= \sum_{0 \le i \le \ell} b_i 2^{wi} P \;=\; eP.\end{aligned}$$

Our point multiplication method is a signed-digit variant of Yao's right-to-left method [25] (see also Knuth [10, exercise 4.6.3-9] and [11, exercise 4.6.3-9] and Brickell et al. [3]) with two essential modifications for achieving resistance against side-channel attacks: The randomized initialization stage is new; and in the right-to-left stage, we treat digit 0 like any other digit.

2.1 Initialization Stage

The initialization stage can be implemented as follows:

1. For each $b \in B' \setminus \{1\}$, generate a random point on the elliptic curve and store it in variable A_b.
2. Compute the point $-\sum_{b \in B' \setminus \{0,1\}} bA_b$ and store it in variable A_1.
3. For each $b \in B'$, perform a projective randomization of variable A_b.

The resulting values of the variables A_b are denoted by A_b^{init}.

If the elliptic curve is fixed, precomputation can be used to speed up the initialization stage: Run steps 1 and 2 just once, e.g. during personalization of a smart card, and store the resulting intermediate values A_b for future use. We denote these values by A_b^{fix}. Then only step 3 (projective randomization of the values A_b^{fix} to obtain new representations A_b^{init}) has to be performed anew each time the initialization stage is called for. The points A_b^{fix} must not be revealed; they should be protected like secret keys.

Generating a random point on an elliptic curve is straightforward. For each element X of the underlying field, there are zero, one or two values Y such that (X, Y) is the affine representation of a point on the elliptic curve. Given a random candidate value X, it is possible to compute an appropriate Y if one exists; the probability for this is approximately $1/2$ by Hasse's theorem. If there is no appropriate Y, one can simply start again with a new X.

Computing an appropriate Y given X involves solving a quadratic equation, which usually (depending on the underlying field) is computationally expensive. This makes it worthwile to use precomputation as explained above. It is also possible to reuse the values that have remained in the variables A_b, $b \neq 1$, after a previous computation, and start at step 2 of the initialization stage.

To determine $-\sum_{b \in B' \setminus \{0,1\}} bA_b$ in step 2, it is not necessary to compute all the individual products bA_b. Algorithm 1 can be used instead to set up A_1 appropriately if $B' = \{0, 1, \ldots, \beta\}$, $\beta \geq 2$. (Note that both loops will be skipped in

Algorithm 1 Compute $A_1 \leftarrow -\sum_{b\in\{2,\ldots,\beta\}} bA_b$ in the initialization stage

for $i = \beta - 1$ down to 2 **do**
  $A_i \leftarrow A_i + A_{i+1}$
$A_1 \leftarrow 2A_2$
for $i = 2$ to $\beta - 1$ **do**
  $A_i \leftarrow A_i - A_{i+1}$
  $A_1 \leftarrow A_1 + A_{i+1}$
$A_1 \leftarrow -A_1$

the case $\beta = 2$.) This algorithm takes one point doubling and $3\beta - 6$ point additions. When it has finished, the variables A_b for $1 < b < \beta$ will contain modified values, but these are representations of the points originally stored in the respective variables. If sufficient memory is available, a faster algorithm can be used to compute A_1 without intermediate modification of the variables A_b for $b > 1$ (use additional variables Q_b instead; in this case, see section 2.3 for a possible additional improvement if point doublings are faster than point additions).

The projective randomization of the variables A_b ($b \in B'$) in step 3 has the purpose to prevent adversaries from correlating observations from the computation of A_1 in the initialization stage with observations from the following right-to-left stage. If algorithm 1 has been used to compute A_1 and the points are not reused for multiple invocations of the initialization stage, then no explicit projective randomization of the variables A_b for $1 < b < \beta$ is necessary; and if $\beta > 2$, no explicit projective randomization of A_1 is necessary: The variables have automatically been converted into new representations by the point additions used to determine their final values.

2.2 Right-to-Left Stage

Algorithm 2 implements the right-to-left stage using a uniform pattern of point doublings and point additions. Initially, for each b, variable A_b contains the value A_b^{init}; the final value is denoted by A_b^{sum}. Due to special cases that must be

Algorithm 2 Right-to-left stage

$Q \leftarrow P$
for $i = 0$ to ℓ **do**
  if $b_i \geq 0$ **then**
    $A_{b_i} \leftarrow A_{b_i} + Q$
  else
    $A_{|b_i|} \leftarrow A_{|b_i|} - Q$
  $Q \leftarrow 2^w Q$

handled in the point addition algorithm (see [7]), uniformity of this algorithm is violated if $A_{|b_i|}$ is a projective representation of $\pm Q$; the randomization in the

initialization stage ensures that the probability of this is negligible. (This is why in section 2.1 we required that precomputed values A_b^{fix} be kept secret.)

If B contains no negative digits, the corresponding branch in the algorithm can be omitted.

The obvious way to implement $Q \leftarrow 2^w Q$ in this algorithm is w-fold iteration of the statement $Q \leftarrow 2Q$, but depending on the elliptic curve, more efficient specific algorithms for w-fold point doubling may be available (see [8]).

In the final iteration of the loop, the assignment to Q may be skipped (the value Q is not used after the right-to-left stage has finished). With this modification, the algorithm uses ℓw point doublings and $\ell + 1$ point additions.

Observe that on two-processor systems the point addition and the w-fold point doubling in the body of the loop may be performed in parallel: Neither operations depends on the other's result.

2.3 Result Stage

Similarly to the computation of A_1 in the initialization stage, the result stage computation

$$\sum_{b \in B' \setminus \{0\}} b A_b^{\mathsf{sum}}$$

can be performed without computing all the individual products bA_b^{sum}. In the result stage, it is not necessary to preserve the original values of the variables A_b, so algorithm 3 (from [10, answer to exercise 4.6.3-9]) can be used if $B' = \{0, 1, \ldots, \beta\}$ when initially each variable A_b contains the value A_b^{sum}. This algorithm uses $2\beta - 2$ point additions.

Algorithm 3 Compute $\sum_{b \in \{1,\ldots,\beta\}} b A_b^{\mathsf{sum}}$ when initially $A_b = A_b^{\mathsf{sum}}$

for $i = \beta - 1$ down to 1 **do**
  $A_i \leftarrow A_i + A_{i+1}$
for $i = 2$ to β **do**
  $A_1 \leftarrow A_1 + A_i$
return A_1

Elliptic curve point arithmetic usually has the property that point doublings are faster than point additions. Then the variant given in algorithm 4 is advantageous. This algorithm uses $\lfloor \beta/2 \rfloor$ point doublings and $2\beta - 2 - \lfloor \beta/2 \rfloor$ point additions.

3 Efficiency

We first examine the efficiency of our algorithm for performing a point multiplication eP in a small configuration with $w = 2$ and $B = \{-2, -1, 0, 1\}$ (i.e. with four variables A_0, A_1, A_2, Q).

Algorithm 4 Compute $\sum_{b\in\{1,\ldots,\beta\}} bA_b^{\mathsf{sum}}$ when initially $A_b = A_b^{\mathsf{sum}}$ (variant)

```
for i = β down to 1 do
  if 2i ≤ β then
    A_i ← A_i + A_{2i}
  if i is even then
    if i < β then
      A_i ← A_i + A_{i+1}
    A_i ← 2A_i
  else
    if i > 1 then
      A_1 ← A_1 + A_i
return A_1
```

For elliptic curve cryptography over prime fields using Jacobian projective coordinates, a point addition can be done in 16 field multiplications, and curves are usually chosen such that a point doubling can be done in 8 field multiplications [7]. The cost for a projective randomization is 5 field multiplications. Generating a random point on the curve would be rather expensive, so we assume that points $A_0^{\mathsf{fix}}, A_1^{\mathsf{fix}}, A_2^{\mathsf{fix}}$ such that $A_1^{\mathsf{fix}} = -2A_2^{\mathsf{fix}}$ have been precomputed.

In this scenario, the initialization stage has to perform three projective randomizations; the right-to-left stage uses 2ℓ point doublings and $\ell + 1$ point additions; and the result stage can be implemented in one point doubling and one point additions. The total cost is $(\ell+2)\cdot 16 + (2\ell+1)\cdot 8 + 3\cdot 5 = 32\ell + 55$ field multiplications; assuming 160-bit scalars ($\ell = 80$), we have 2615 field multiplications.

With two processors, in the loop of the right-to-left stage, the two point doublings ($2 \cdot 8 = 16$ field multiplications) can be performed in parallel with the one point addition (also 16 field multiplications), and so we can remove 16ℓ field multiplications from the tally. (We ignore the small additional savings that can be achieved through parallelization in the other stages.) Only $16\ell + 55$ field multiplications remain; for 160-bit scalars, this is 1335 field multiplications.

The 2^w-ary left-to-right method from [15] with the improvement from [16] in a similar configuration (digit set $\{-4, -1, 1, 2\}$, three points P, $2P$, $4P$ to precompute) uses one projective randomization followed be two point doublings for precomputation and then one projective randomization, 2ℓ point doublings, and ℓ point additions for determining the result. If we additionally perform a projective randomization after each except the very last point addition to avoid a fixed table, the total cost becomes $\ell \cdot 16 + (2\ell+2)\cdot 8 + (\ell+1)\cdot 5 = 37\ell + 21$ field multiplications. Assuming 160-bit scalars ($\ell = 80$), we have 2981 field multiplications. This is about 14 % more than with the new method on a single processor.

Now we consider similar scenarios with arbitrary window sizes $w \geq 2$ and arbitrary scalar bit lengths n. The new method ($\ell = \lfloor n/w \rfloor$, $\beta = 2^{w-1}$) performs $2^{w-1} + 1$ projective randomizations in the initialization stage; $\lfloor n/w \rfloor \cdot w$ point doublings and $\lfloor n/w \rfloor + 1$ point additions in the right-to-left stage; and 2^{w-2}

point doublings and $3 \cdot 2^{w-2} - 2$ point additions in the result stage. The total cost is

$$\left\lfloor \frac{n}{w} \right\rfloor \cdot (w \cdot 8 + 16) + 2^{w-2} \cdot 66 - 11$$

field multiplications.

Note that in the case with parallelization, $w = 2$ provides better performance than larger values of w (the right-to-left stage provides essentially the same amount of work to both processors if $w = 2$). Compared with the one-processor variant, we always save $\lfloor n/w \rfloor \cdot 16$ field multiplications, and

$$\left\lfloor \frac{n}{w} \right\rfloor \cdot w \cdot 8 + 2^{w-2} \cdot 66 - 11$$

field multiplications remain.

For arbitrary window size $w \geq 2$ and scalar bit length n, the left-to-right method from [15] as improved in [16] with additional projective randomizations to avoid a fixed table uses one projective randomization, $2^{w-2} + 1$ point doublings, and $2^{w-2} - 1$ point additions for precomputation and $\lfloor n/w \rfloor$ projective randomizations, $\lfloor n/w \rfloor \cdot w$ point doublings, and $\lfloor n/w \rfloor$ point additions for computing the result. The total cost of this is

$$\left\lfloor \frac{n}{w} \right\rfloor \cdot (w \cdot 8 + 21) + 2^{w-2} \cdot 24 - 3$$

field multiplications.

Table 1 compares the efficiency of the methods for various window sizes in the case of 160-bit scalars. Table 2 provides a similar comparison for 256-bit scalars. (Note that these efficiency comparisons do not take into account the additional cost of generating random field elements for repeated projective randomization in the left-to-right method from [15] and [16].) Table entries are printed in bold if the respective window size w provides better efficiency than smaller values of w, i.e. if w is optimal given certain bounds on memory usage.

The new method needs read-write memory for the same number of points as the left-to-right method from [16] with the same window size. (The new method needs additional read-only memory for the precomputed points A_b^{fix}.) The tables show that, when using a single processor, the left-to-right method with additional projective randomizations can be faster than the new method, but will need more read-write memory to achieve this: In the example scenarios, the left-to-right method needs $w = 5$ (17 table values) to outperform the new method with $w = 4$. (Whether the left-to-right method is actually faster in such cases depends on the speed of random number generation for projective randomization.)

4 Variants

We show some possibilities to vary the point multiplication method described in section 2.

Table 1. Number of field multiplications for a 160-bit point multiplication

w	2	3	4	5	6
New method	**2615**	**2241**	**2173**	2309	2709
New method, two processors	**1335**	1393	1533	1797	2293
L-to-R method [16], proj. rand. to avoid fixed table	**2981**	**2430**	**2213**	**2141**	2175

Table 2. Number of field multiplications for a 256-bit point multiplication

w	2	3	4	5	6	7
New method	**4151**	**3521**	**3325**	3373	3733	4693
New method, two processors	**2103**	2161	2301	2557	3061	4117
L-to-R method [16], proj. rand. to avoid fixed table	**4757**	**3870**	**3485**	**3300**	**3279**	3537

4.1 Projective Randomization of P

While it does not appear to be strictly necessary, we recommend to perform a projective randomization of P before beginning the right-to-left stage (algorithm 2). At small computational cost, this will further reduce the side-channel information available to potential attackers.

4.2 Scalar Randomization

Okeya and Sakurai [20] describe a second-order power analysis attack on the fixed-table point multiplication method of [15]. The attack requires power consumption traces from many point multiplications using the same scalar e (and thus the same addition chain). The basis of the attack is to detect table-value reuse by observing side-channel data that leaks information on the Hamming weight of representations of points (cf. [14]): To find out whether the i-th and j-th point addition use the same table value, compute for each power consumption trace the difference between (normalized) power consumption measurements at the two points of time when the respective table entries are read from memory; over sufficiently many point multiplications, the sample variance of these power consumption differences should converge to one of two values depending on whether the i-th and j-th point addition use the same table value or different table values.

No experimental results are given in [20]. If this attack is practical, similar attacks may be possible against most point multiplication methods using a constant sequence of operations as it may be possible to trace values based on their Hamming weight (i.e. determine whether the output of the i-th operation is used as input to the j-th operation). A countermeasure is to randomize the addition chain. This can be done by randomizing the scalar e: Compute eP with two point multiplications and one point subtraction as

$$(e + mN + \widetilde{m})P - \widetilde{m}P$$

where N is the order of the elliptic curve group and $m, \widetilde{m}$ are one-time random numbers (e.g. 32 bits long).

(Adding a multiple of the group order was originally proposed in [12], but it leaves some bias in the least significant digits [19]. Scalar splitting in the form $eP = (e+m)P - mP$ as proposed in [4] avoids this bias, but is only sufficient if m is of the same length as e, which would double the cost of a point multiplication. By combining these two ideas, we avoid the bias while keeping the overhead low.)

4.3 Avoiding Digit 0

In the point multiplication method described in section 2, if $0 \in B$, the variable A_0 is essentially a dummy variable: Its value does not affect the final result. Assume that an attacker is performing a fault attack [2] by purposefully inducing computation faults. If these faults occur only during computations with A_0, the result of the point multiplication will still be correct. Thus, verifying the result cannot reveal that a fault attack has taken place. Therefore it may be useful to avoid the dummy variable.

The point multiplication method of section 2 can be used with a digit set B that does not include the value 0, e.g.

$$B = \left\{-2^w, \pm 1, \pm 2, \ldots, \pm(2^w - 2), 2^w - 1\right\}$$

as in [16]. Compared with digit set $\{-2^{w-1}, \ldots, 2^{w-1} - 1\}$, this requires modifications to the algorithms used in step 2 of the initialization stage (section 2.1) and in the result stage (section 2.3). If we assume that the initialization stage uses precomputed points A_b^{fix}, only the changes to the result stage will increase the computational cost of a point multiplication. The result stage for said digit set has to compute the sum

$$\sum_{b \in \{1, \ldots, 2^{w-1}, 2^w\}} b A_b^{\mathsf{sum}};$$

the additional cost is one point doubling and one point addition (set $A_{2^{w-1}} \leftarrow A_{2^{w-1}} + 2A_{2^w}$ before running algorithm 3 or 4).

4.4 Variant for $w = 1$

The point multiplication method as described in section 2 works only for $w \geq 2$ because of the requirement that $A_b^{\mathsf{init}} \neq \mathcal{O}$ for each $b \in B'$, but $\sum_{b \in B'} b A_b^{\mathsf{init}} = \mathcal{O}$. The method can be adapted to the case $w = 1$ by relinquishing the latter part of the requirement; instead, save the value A_1^{init} and change the result stage to compute $A_1^{\mathsf{sum}} - A_1^{\mathsf{init}}$.[1] If A_1^{init} is just a projective randomization of a precomputed random point A_1^{fix}, there is no need to save A_1^{init}, as the result stage can simply compute $A_1^{\mathsf{sum}} - A_1^{\mathsf{fix}}$.

4.5 Application to Modular Exponentiation

A variant of the method of section 2 can be used for modular exponentiation. For this purpose, digit set B will only contain non-negative digits.

[1] This variant was suggested by Tsuyoshi Takagi.

5 Conclusion

We have described a 2^w-ary right-to-left method for elliptic curve point multiplication that employs a randomized initialization stage to achieve resistance against side-channel attacks. In contrast to similar left-to-right methods, there is no inherent fixed table; thus the new method is more secure than fixed-table left-to-right methods, and in many cases faster than left-to-right methods that use repeated projective randomization to avoid a fixed table. Also the right-to-left method is easily parallelizable and provides much improved performance on two-processor systems.

References

1. Bier, É., and Joye, M. Weierstraß elliptic curves and side-channel attacks. In *Public Key Cryptography – PKC 2002* (2002), D. Naccache and P. Paillier, Eds., vol. 2274 of *Lecture Notes in Computer Science*, pp. 335–345.
2. Boneh, D., DeMillo, R. A., and Lipton, R. J. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology 14* (2001), 101–119.
3. Brickell, E. F., Gordon, D. M., McCurley, K. S., and Wilson, D. B. Fast exponentiation with precomputation. In *Advances in Cryptology – EUROCRYPT '92* (1993), R. A. Rueppel, Ed., vol. 658 of *Lecture Notes in Computer Science*, pp. 200–207.
4. Clavier, C., and Joye, M. Universal exponentiation algorithm – a first step towards provable SPA-resistance. In *Cryptographic Hardware and Embedded Systems – CHES 2001* (2001), Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162 of *Lecture Notes in Computer Science*, pp. 300–308.
5. Coron, J.-S. Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems – CHES '99* (1999), Ç. K. Koç and C. Paar, Eds., vol. 1717 of *Lecture Notes in Computer Science*, pp. 292–302.
6. Fischer, W., Giraud, C., Knudsen, E. W., and Jean-Pierre, S. Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against non-differential side-channel attacks. Cryptology ePrint Archive Report 2002/007, 2002. Available from `http://eprint.iacr.org/`.
7. Institute of Electrical and Electronics Engineers (IEEE). IEEE standard specifications for public-key cryptography. IEEE Std 1363-2000, 2000.
8. Itoh, K., Takenaka, M., Torii, N., Temma, S., and Kurihara, Y. Fast implementation of public-key cryptography on a DSP TMS320C6201. In *Cryptographic Hardware and Embedded Systems – CHES '99* (1999), Ç. K. Koç and C. Paar, Eds., vol. 1717 of *Lecture Notes in Computer Science*, pp. 61–72.
9. Izu, T., and Takagi, T. A fast parallel elliptic curve multiplication resistant against side channel attacks. In *Public Key Cryptography – PKC 2002* (2002), D. Naccache and P. Paillier, Eds., vol. 2274 of *Lecture Notes in Computer Science*, pp. 280–296.
10. Knuth, D. E. *The Art of Computer Programming – Vol. 2: Seminumerical Algorithms (2nd ed.).* Addison-Wesley, 1981.
11. Knuth, D. E. *The Art of Computer Programming – Vol. 2: Seminumerical Algorithms (3rd ed.).* Addison-Wesley, 1998.

12. Kocher, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology – CRYPTO '96* (1996), N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, pp. 104–113.
13. Kocher, P. C., Jaffe, J., and Jun, B. Differential power analysis. In *Advances in Cryptology – CRYPTO '99* (1999), M. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, pp. 388–397.
14. Messerges, T. S. Using second-order power analysis to attack DPA resistant software. In *Cryptographic Hardware and Embedded Systems – CHES 2000* (2000), Ç. K. Koç and C. Paar, Eds., vol. 1965 of *Lecture Notes in Computer Science*, pp. 238–251.
15. Möller, B. Securing elliptic curve point multiplication against side-channel attacks. In *Information Security – ISC 2001* (2001), G. I. Davida and Y. Frankel, Eds., vol. 2200 of *Lecture Notes in Computer Science*, pp. 324–334.
16. Möller, B. Securing elliptic curve point multiplication against side-channel attacks, addendum: Efficiency improvement. `http://www.informatik.tu-darmstadt.de/TI/ Mitarbeiter/moeller/ecc-sca-isc01.pdf`, 2001.
17. Montgomery, P. L. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation 48* (1987), 243–264.
18. Okeya, K. Method of calculating multiplication by scalars on an elliptic curve and apparatus using same. European Patent EP1160661, 2001.
19. Okeya, K., and Sakurai, K. Power analysis breaks elliptic curve cryptosystems even secure against the timing attack. In *Progress in Cryptology – INDOCRYPT 2000* (2000), B. K. Roy and E. Okamoto, Eds., vol. 1977 of *Lecture Notes in Computer Science*, pp. 178–190.
20. Okeya, K., and Sakurai, K. A second-order DPA attack breaks a window-method based countermeasure against side channel attacks. In *Information Security – ISC 2002* (these proceedings), A. H. Chan and V. Gligor, Eds.
21. Schindler, W. A combined timing and power attack. In *Public Key Cryptography – PKC 2002* (2002), D. Naccache and P. Paillier, Eds., vol. 2274 of *Lecture Notes in Computer Science*, pp. 263–279.
22. Vadekar, A., and Lambert, R. J. Timing attack resistant cryptographic system. Patent Cooperation Treaty (PCT) Publication WO 00/05837, 2000.
23. Vanstone, S. A., and Gallant, R. P. Power signature attack resistant cryptography. Patent Cooperation Treaty (PCT) Publication WO 00/25204, 2000.
24. Walter, C. D., and Thompson, S. Distinguishing exponent digits by observing modular subtractions. In *Progress in Cryptology – CT-RSA 2001* (2001), D. Naccache, Ed., vol. 2020 of *Lecture Notes in Computer Science*, pp. 192–207.
25. Yao, A. C.-C. On the evaluation of powers. *SIAM Journal on Computing 5* (1976), 100–103.

Notes[2]

Right-to-Left Stage

Katsuyuki Okeya has pointed out a problem with algorithm 2 as described in section 2.2: unless the variant from section 4.1 is used, adversaries are able to predict the representations of Q and $-Q$ and thus may be able to distinguish between the two conditional branches (even if dummy point inversions are used to let point additions look like point subtractions).

To avoid this problem, algorithm 2 can be expressed as follows:

$Q \leftarrow P$
for $i = 0$ to ℓ **do**
  if $b_i \geq 0$ **then**
    $A_{b_i} \leftarrow A_{b_i} + Q$
  else
    $A_{|b_i|} \leftarrow -\big((-A_{|b_i|}) + Q\big)$
  $Q \leftarrow 2^w Q$

Implementations should use dummy point inversions to achieve uniform behaviour for the two conditional branches.

Scalar Randomization

Scalar randomization for computing eP as proposed in section 4.2 can be expressed as follows:

$$(e + mN + \widetilde{m})P + \widetilde{m}(-P)$$

When computing this sum of products with our point multiplication method, the initialization stage and result stage need to be run just once; only the right-to-left stage needs to be run twice. The final result stage will automatically yield the combined result.

References

The list of authors for reference [6] should read Fischer, W., Giraud, C., Knudsen, E. W., and Seifert, J.-P..

[2] Added 2002-10-16. Do not appear in *ISC 2002* proceedings.