Main text appears in G. I. Davida, Y. Frankel (Eds.): *Information Security – ISC 2001,* Springer-Verlag LNCS 2200, pp. 324–334, ISBN 3-540-42662-0.

Addendum "Efficiency Improvement" added 2001-08-27/2001-08-29.
Errata added 2001-12-21.

Securing Elliptic Curve Point Multiplication against Side-Channel Attacks

Bodo Möller

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

Abstract. For making elliptic curve point multiplication secure against side-channel attacks, various methods have been proposed using special point representations for specifically chosen elliptic curves. We show that the same goal can be achieved based on conventional elliptic curve arithmetic implementations. Our point multiplication method is much more general than the proposals requiring non-standard point representations; in particular, it can be used with the curves recommended by NIST and SECG. It also provides efficiency advantages over most earlier proposals.

1 Introduction

Side-channel attacks on implementations of cryptosystems use observations such as timings [9] or power consumption measurements [10] in order to obtain information that is supposed to be kept secret. In elliptic curve cryptosystems, a particular target for side-channel attacks are algorithms used for point multiplication. The computational task is to compute a product $[e]P$ where P is a point on an elliptic curve $E(\mathbb{F}_q)$ over a finite field and e is a secret positive integer. Point P is usually not secret, and indeed may often be chosen by the attacker. Multiplier e may either be ephemeral or serve as a long-time key.

Elliptic curves used for cryptography are usually required to have a large prime-order subgroup. We denote its order by p and assume that only one order-p subgroup exists. In this setting, cryptographic protocols typically employ only points of order p.

Various countermeasures against power analysis have been proposed that randomise the computation to make the attacker's task harder:

- If projective coordinates are used for representing points (which is usually the case for efficiency reasons, see [1] and [7]), then the representation of the input point can easily be transformed into a random equivalent representation [6].
- $[e]P$ can be expressed as $[e](P - Q) + [e]Q$ where Q is a random point.
- $[e]P$ can be expressed as $[e + np]P$ or $[e - n]P + [n]P$ where n is a random integer.

However, while these countermeasures may provide protection against differential side-channel analysis (i.e. attacks requiring correlated measurements from

multiple computations), they are not likely to provide sufficient protection if simple side-channel analysis is possible (i.e. the direct interpretation of measurements from a single computation). Note that straightforward implementations of elliptic curve systems are particularly vulnerable to simple side-channel analysis because the usual point addition algorithm cannot be used for point doubling: As adding and doubling require different algorithms, the bits of e may be plainly visible in a power consumption trace if the double-and-add algorithm is used for point multiplication. Improved point multiplication algorithms such as the m-ary or sliding window method or their counterparts using signed digits or signed windows [12] will obscure e to some degree, but may still reveal plenty of information.

For these reasons, Liardet and Smart [11] and Joye and Quisquater [8] have proposed reducing information leakage by using special representations of points of certain elliptic curves over prime fields such that a single formula can be used both for addition and doubling. This approach induces some performance penalty, however (with the usual algorithms, point doubling is much faster than addition). Okeya and Sakurai [16] take a different approach by using a variant of Montgomery's point multiplication algorithm [13] for suitable curves over odd-characteristic fields, where the usual group operations are replaced by certain special operations working with triplets of points with y-coordinates omitted, allowing to implement a quite efficient variant of the binary point multiplication method that does not readily reveal the bits of the multiplier.

We will not discuss the details of these methods. A common disadvantage is that each of them requires specifically selected elliptic curves: All curves suitable for [11] (curves with Jacobi form) have group order order divisible by 4; curves suitable for [8] (curves with Hesse form) have group order divisible by 3; and curves suitable for [16] (curves with Montgomery form) again have group order divisible by 4. None of these methods is applicable to the NIST and SECG recommended curves given in [14] and [4], whose use is often encouraged in order to ease interoperability.

We propose an alternative approach without a limitation to specifically chosen curves (and indeed without a limitation to curves over fields of odd characteristic): We accept that doublings are more efficient than additions, but we use these two in a uniform pattern. This makes our method largely independent of the particular point representation employed. We will see that in terms of field multiplications our method with the usual Jacobian projective coordinates is typically more efficient than the Liardet-Smart and Joye-Quisquater proposals.

Our approach requires more additions than the standard 2^w-ary point multiplication algorithm, of which it is a variant, but it does not involve dummy additions in the sense of throwing away results of point operations. Dummy additions provide an immediate way to achieve a fixed pattern of point doublings and point additions [6, section 3.1]; however, as we will explain in section 2, it appears prudent to avoid them.

The method may fail in some cases in that an addition step may turn out to be a point doubling or may involve the point at infinity (which both requires special

treatment and is potentially clearly visible through side channels). However, we will show that the probability of this happening is negligible if multipliers are appropriately selected: Randomly chosen e is safe.

Multiple side-channel attack countermeasures can be combined; the point multiplication method proposed in the current paper can be used together with one or multiple of the randomisation methods mentioned above. In particular, randomised projective coordinates should be used (cf. [16]).

In section 2, we discuss assumptions of the security model underlying our side-channel attack countermeasure and look into certain implementation details. Section 3 presents our approach for implementing elliptic curve point multiplication with security against side-channel attacks. Section 4 provides an efficiency comparison between our method and the methods of [11], [8], and [16].

2 Security against Side-Channel Attacks

The goal to achieve security against side-channel attacks warrants some notes on the model to be used for the security analysis. While a complete model of all side-channel information is hardly ever attainable (and, in any case, would be closely tied to a specific implementation), it is possible to describe the workings of the point multiplication algorithm in enough detail to obtain reasonable assurance that information leakage will not be useful to an attacker. Before presenting the actual point multiplication method in section 3, we discuss desired features and how to achieve them. The most prominent item to consider are elliptic curve point operations. We enumerate special cases of point operations that should be avoided (section 2.1). At a lower level, we can focus on the individual field operations used to implement point operations. In section 2.2, we show that point multiplication implementations intended to be secure against side-channel attacks should use randomised projective coordinates and should use certain extended point representations; we also explain why inserting dummy point additions to achieve uniform behaviour would be questionable.

2.1 Elliptic Curve Point Operations

As different algorithms are usually required for point doubling and point addition, we assume that side-channel data reveals the sequence in which these operations take place. Thus the point multiplication algorithm should use doublings and additions in a uniform pattern independent of the specific multiplier. We also have to take into account certain special cases that must be expected to be visible through side channels when they occur:

- Point doubling $[2]A$ requires conditional statements for the case that A is the point at infinity or that A is a point of order two. If these cases are avoided, then, expressed in field operations, point doubling runs as a fixed routine.
- Point addition $A + B$ requires conditional statements for the case that one of the points is the point at infinity, or that A coincides with B, or that one

point is the inverse of the other. For other cases, it too can be implemented as a fixed routine.

Details of the sequences of field operations used for point doubling and point addition depend on the underlying field (odd characteristic vs. characteristic 2) and the choice of point representations (e.g. either affine coordinates or one of multiple styles of projective coordinates, cf. [5]), so implementations may vary widely. The essential observation is that the respective algorithm always behaves the same as long as the above special cases are avoided.

2.2 Field Operations

While it is reasonable to assume that side-channel observations for, say, a field multiplication do not immediately reveal the factors involved, it would not be prudent to assume that all multiplications look the same to an attacker. This is why randomised projective coordinates are useful ([6], [16]): If, e.g., Jacobian projective coordinates are used, triplets (X, Y, Z) with $Z \neq 0$ represent affine points $(X/Z^2, Y/Z^3)$; then for any field element $\lambda \neq 0$, $(\lambda^2 X, \lambda^3 Y, \lambda Z)$ is a representation of the same point on the curve. If λ is secretly chosen at random, none of the coordinates of the new representation will be predictable to the attacker. Point doubling or point addition using projective coordinates results in a point represented with a Z-coordinate that is the product of the Z-coordinate(s) of the input point(s) and a short polynomial involving one or more other coordinates of the input points; thus the output point is again in a randomised representation.

When use of randomised projective coordinates prevents the attacker from guessing the specific inputs to field operations (and thus from directly verifying, for example, which of several known points are currently being added), the attacker may still be able to recognise if the same field operation with the same input values is performed multiple times. Even if these values are not known, this may leak essential information, so we want to avoid it. We now show what should be taken care of.

Many algorithms for point multiplication $[e]P$, including the one that we will examine in section 3, can be roughly outlined as follows:

[Precomputation stage.] First, independently of the specific multiplier e, certain small multiples of P are computed and stored in a table.

[Evaluation stage.] Second, the product $[e]P$ is evaluated as follows: A variable A is initialized to one of the table values; then, many times, A either is doubled or a table value is added to A, replacing the previous value of A. Finally, A contains the result $[e]P$.

For algorithms using this structure, each entry of the precomputed table should contain not only the representation of the point, but additionally any field elements that would have to be computed each time the point is added to A. If Jacobian projective coordinates are used, this means that (X, Y, Z, Z^2) should be stored rather than just (X, Y, Z) (cf. the addition formulas in [1, Chapter IV.2] or [7]). If such *extended point representations* are not used for the precomputed

table, the attacker may be able to tell which point additions use the same table value.

Most point multiplication algorithms of the above form can achieve a fixed pattern of doublings and additions only if dummy additions are inserted into the evaluation stage. That is, sometimes a table value is added to A, but the result is thrown away and A is left unchanged. The problem with dummy additions is the the attacker may be able to tell that their result is ignored: For Jacobian projective coordinates, each point operation requires squaring the Z-coordinate of A; if A is not changed, then two consecutive point operations involve the very same squaring. The point multiplication algorithm presented in section 3 achieves uniform behaviour without resorting to dummy additions.

Finally, we show how to use randomised projective coordinates in practice. It is possible, but inefficient, to randomise the point representation after each point operation. (If this is done, dummy additions are no longer a problem.) For point multiplication algorithms of the above form, it is more practical to use randomisation just twice or once: If the precomputed table of multiples of P is stored in projective coordinates, then the representation of P should be randomised before the table is computed. Also at the beginning of the second stage, after the initial value has been assigned to A, the representation of A should be randomised. If the table of multiples of P is stored in affine coordinates (to speed up the evaluation stage by using mixed addition of affine and projective points [5]), then the first randomisation obviously is not necessary.

3 Multiplier Recoding Providing Resistance against Side-Channel Attacks

We show how to perform point multiplication $[e]P$ in a way such that doublings and additions occur in a fixed pattern in order to provide resistance against side-channel attacks. Section 3.1 describes an algorithm for recoding the multiplier e into a special signed-digit encoding. In section 3.2, we discuss the multiplication algorithm implied by this encoding. Section 3.3 shows that, unless e is ill-chosen, this point multiplication algorithm indeed limits information leakage as intended.

3.1 Recoding Algorithm

Let the positive integer e be given in 2^w-ary digits where $w \geq 2$ is a small integer: That is,

$$e = \sum_{i=0}^{k'} b'_i \cdot 2^{wi}$$

with $b'_i \in \{0, 1, \ldots, 2^w - 1\}$. We demand that k' be chosen minimal, i.e. $b'_{k'} \neq 0$. For $i > k'$, we define that $b'_i = 0$.

We will show how to convert this into a representation

$$e = \sum_{i=0}^{k} b_i \cdot 2^{wi}$$

such that $b_i \in \{-2^w, 1, 2, \ldots, 2^w - 1\}$. This means that we disallow digit value 0 and instead introduce the new value -2^w. Intuitively, the recoding algorithm replaces 0 digits by -2^w and increments the next more significant digit to adjust the value. It is easy to see that b_k must be positive (otherwise e would be negative) and that the representation of e needs to grow by at most one digit, i.e. $k = k'$ or $k = k' + 1$.

We express the recoding algorithm recursively, using auxiliary values c_i and t_i such that $0 \le c_i \le 2$ and $0 \le t_i \le 2^w + 1$: Let

$$c_0 = 0,$$

and for $i = 0, \ldots, k' + 1$, let

$$t_i = b'_i + c_i$$

and

$$(c_{i+1}, b_i) = \begin{cases} (1, -2^w) & \text{if } t_i = 0 \\ (0, t_i) & \text{if } 0 < t_i < 2^w \\ (2, -2^w) & \text{if } t_i = 2^w \\ (1, 1) & \text{if } t_i = 2^w + 1. \end{cases}$$

Note that we always have $c_{i+1} \cdot 2^w + b_i = t_i$.

If $b_{k'+1} \neq -2^w$, then $e = \sum_{i=0}^{k'+1} b_i \cdot 2^{wi}$; in this case, we set $k = k' + 1$. Otherwise, we have $e = \sum_{i=0}^{k'} b_i \cdot 2^{wi}$ and $b'_k \neq -2^w$, and we set $k = k'$.

Observe that if $b_k = 1$ (and $k > 0$), then $b_{k-1} \neq -2^w$.

As a measure against side-channel attacks on the recoding algorithm itself, assignments can be implemented by table lookups instead of using conditional statements.

Storing the converted representation of e requires almost no additional memory compared with the original representation: Digits with value -2^w can be encoded as a pattern of bits all set to zero. If w is understood, then this new representation can be stored as an ordinary integer. (Leading 0 digits can simply be ignored because b_k is never -2^w.) This integer is at most two bits longer than e; the maximum possible length growth occurs if $k = k' + 1$ and $b_k = 2$.

It may be desirable to ensure that the encoded form has a predetermined maximum index k. For our point multiplication application, if kw is large enough compared with the binary length of p, this is easy to do: If necessary, adjust e by adding p or a multiple of p.

If a random multiplier is required and a uniform distribution is not essential, then random bits can be used directly to generate the recoded form of e. In this case, too, it should be ensured that $b_{k-1} \neq -2^w$ if $b_k = 1$.

3.2 Point Multiplication Algorithm

Remember that we want to compute $[e]P$ where point P should have order p, p being a large prime that divides the order of the elliptic curve $E(\mathbb{F}_q)$. For our point multiplication algorithm to work as intended, $\mathrm{ord}(P)$ may be any positive multiple of p.

If point P can be chosen by the attacker, we must be aware that it might have incorrect order. In case that $\#E(\mathbb{F}_q) = p$, then (besides testing that P is indeed a point on the curve) we just have to verify that P is not the point at infinity, $\mathcal{O}$. Otherwise, we need an additional sanity check. Let h be the cofactor of the elliptic curve, i.e. the integer $\#E(\mathbb{F}_q)/p$. Curves used for cryptography are usually selected such that h is small, e.g. $h \leq 4$ as required by [3]. Thus $[h]P$ can be computed quickly; if it is not $\mathcal{O}$, then we know that p divides $\mathrm{ord}(P)$.

If P has incorrect order, computing $[e]P$ must be rejected. Otherwise, given a representation

$$e = \sum_{i=0}^{k} b_i \cdot 2^{wi}$$

of e as determined by the recoding algorithm of 3.1, $[e]P$ can be computed as follows:

Assume that points P, $[2]P$, $[3]P$, ..., $[2^w - 1]P$, $[-2^w]P$ have been stored in a table. Let $e_j = \sum_{i=j}^{k} b_i \cdot 2^{w(i-j)}$ for $j = 0, \ldots, k$. The goal is to compute $[e_0]P = [e]P$. We can determine $[e_k]P = [b_k]P$ simply by table lookup. Then, to compute $[e_j]P$ for j going down to 0, we can use that $e_j = 2^w \cdot e_{j+1} + b_j$ and thus $[e_j]P = [2^w]\big([e_{j+1}]P\big) + [b_j]P$, where $[b_j]P$ again is available by table lookup. This is essentially the 2^w-ary algorithm for computing $[e]P$, except that we use an unusual set of digits.

The procedure is given in algorithm 1. We show a high-level description of the algorithm; as explained in section 2.2, implementations should use randomised projective coordinates, and values computed in the precomputation stage should be stored in extended point representation. The algorithm requires exactly $2^{w-1} + kw$ point doublings and $2^{w-1} - 1 + k$ point additions. (Inverting $[2^w]P$ to obtain $[-2^w]P$ is essentially free in elliptic curves.)

3.3 Uniformity of the Point Multiplication Algorithm

It is apparent that algorithm 1 uses doublings and additions in a regular pattern. As discussed in section 2, to show that the algorithm has uniform behaviour and thus is suitable for limiting information leakage during the computation of $[e]P$, we also have to show that the following special cases of point doubling and point addition can be avoided:

- $[2]\mathcal{O}$
- $[2]A$ where $\mathrm{ord}(A) = 2$

- $A + \mathcal{O}$ or $\mathcal{O} + B$
- $A + A$
- $A + (-A)$

We have required that $\mathrm{ord}(P)$ be a positive multiple of p. Without loss of generality, here we may assume that $\mathrm{ord}(P) = p$. (Otherwise, multiply P by the cofactor $h = \#E(\mathbb{F}_q)/p$ to obtain a point of order p. In the algorithm performed for P, the above special cases can occur only when one of them would occur

Algorithm 1 Compute $\left[\sum_{i=0}^{k} b_i \cdot 2^{wi}\right] P$

{*Precomputation stage*}
$P_1 \leftarrow P$
for $n = 2$ to $2^w - 2$ step 2 **do**
  $P_n \leftarrow [2]P_{n/2}$
  $P_{n+1} \leftarrow P_n + P$
  $\{P_n = [n]P,\ P_{n+1} = [n+1]P\}$
end for
$P_{-2^w} \leftarrow -[2]P_{2^{w-1}}$
$\{P_{-2^w} = [-2^w]P\}$

{*Evaluation stage*}
$A \leftarrow P_{b_k}$
for $j = k - 1$ down to 0 **do**
  $A \leftarrow [2^w]A$
  $A \leftarrow A + P_{b_j}$
end for
return A

in the algorithm performed for $[h]P$; in particular, $\mathrm{ord}(A) = 2$ would imply $[h]A = \mathcal{O}$, so points of order 2 are not an issue if we can show that the point at infinity can be avoided.) Then, as 2^w can be expected to be much smaller than p, all precomputed points $P_{b_j} = [b_j]P$ in algorithm 1 will be of order p. Thus, initially we have $A \neq \mathcal{O}$; and as long as we avoid additions of the form $A + (-A)$, it will stay like this. So what remains to be checked is whether in the evaluation stage of algorithm 1 we can avoid that $A = [b_j]P$ or $A = [-b_j]P$ before an addition takes place.

With e_j defined as in section 3.2, the point addition step in the second loop of algorithm 1 performs the point addition $[2^w \cdot e_{j+1}]P + [b_j]P$. Thus we are save if

$$2^w \cdot e_{j+1} \not\equiv \pm b_j \pmod{p}.$$

Since $|e_j| \leq 2^{(1+k-j)w}$ and $|b_j| \leq 2^w$, for many indices j we have $|2^w \cdot e_{j+1}| + |b_j| < p$, meaning that reduction modulo p does not matter and it suffices to check whether $2^w \cdot e_{j+1} \neq \pm b_j$.

The largest index to consider is $j = k-1$: Can we be sure that $2^w \cdot e_k \neq \pm b_{k-1}$? Indeed we can, as $e_k = b_k$ and if $b_k = 1$, then $b_{k-1} \neq -2^w$ (see section 3.1). It follows that $e_j \geq 2^{(k-j)w}$, which shows that the incongruence is satisfied for indices $j < k - 1$ too as long as we do not have to consider reduction modulo p.

For indices j so small such that this modular reduction is relevant, we argue that if e is sufficiently random, then the probability of picking an e such that the above incongruence is not satisfied can be neglected: It is comparable to the probability that e mod p can be guessed in a modest number of attempts, which can be presumed to be practically impossible.

4 Efficiency Comparison

We show that using the point multiplication method of section 3 with the usual Jacobian projective representation of points is typically more time-efficient than the method of Joye and Quisquater [8] and the method of Liardet and Smart [11]. The method of Okeya and Sakurai [16], however, has performance advantages (the downside being that it requires specific curves).

With the Joye-Quisquater proposal, 12 multiplications in the underlying field are needed for each addition or doubling [8]. (The Liardet-Smart proposal requires 16 field multiplications for each addition or doubling [11] and thus is computationally more expensive, so we omit in the remainder of this comparison.) This means that for computing $[e]P$, the expected number of field multiplications per bit of e is more than 12. For a more precise estimate, assume that the point multiplication algorithm based on signed windows [12] is used (one of the most efficient general point multiplication algorithms) and that $2^w - 1$ points are precomputed. Then the number of field multiplications per bit is about $(1 + \frac{1}{w+3}) \cdot 12$, not including the precomputation effort. For multipliers whose length ℓ is between 120 and 336 bits, the expected number of operations required by the signed window method is minimised when $2^3 - 1 = 7$ points are precomputed; the expected number of field operations then is about

$$\left(7 + \frac{7}{6} \cdot \ell\right) \cdot 12 = 84 + 14 \cdot \ell$$

including precomputation.

For elliptic curve cryptography over fields of odd characteristic, curves are usually chosen such that a doubling can be done in 8 field multiplications and an addition can be done in 16 field multiplications using Jacobian projective coordinates [7]. (Further speed-ups are possible by using mixed coordinates for faster point addition; see [5] and [2].) With our method of section 3, using an extended point representation for precomputed points as explained in section 2.2, one additional field multiplication is required for each precomputed point, but only 15 field multiplications are needed for each evaluation stage point addition. Thus, if we use $w = 4$, we need $8 + \frac{1}{4} \cdot 15 = 11.75$ field multiplications for each bit of e (not including the effort to precompute 15 points). This is less than 12, so for sufficiently long e this method will be faster than the Joye-Quisquater method. Choosing $w = 4$ minimises the number of field multiplications for scalars of length ℓ between 84 and 277 bits; it is about

$$192 + 11.75 \cdot \ell,$$

including $2^3 \cdot 8 + (2^3 - 1) \cdot 16 + 2^4 = 192$ multiplications for precomputation using extended point representation (but neglecting the small additional cost for randomising projective coordinates). Compared with the Joye-Quisquater approach, this is nearly 10 % faster even for multipliers as short as 120 bits.

The method of [16] for curves having a Montgomery form is more efficient than any of the other methods: It needs only 11 field multiplications per bit [15], and as it is directly based on the binary point multiplication algorithm, it does not involve any precomputation.

5 Conclusion

We have presented a method for elliptic curve point multiplication that can be shown to provide security against side-channel attacks. The algorithm uses a special signed-digit encoding to ensure that point doublings and point additions occur in a uniform pattern. No dummy additions are required; implementing the method using randomised projective coordinates and storing precomputed points in extended point representation limits information leakage to a minimum. While the method of Okeya and Sakurai [16] for curves with Montgomery form is more efficient, the approach of the current paper is much more general: Unlike various previous proposals, it is applicable to the recommended curves of [14] and [4].

References

1. Blake, I. F., Seroussi, G., and Smart, N. P. *Elliptic Curves in Cryptography*, vol. 265 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1999.
2. Brown, M., Hankerson, D., López, J., and Menezes, A. Software implementation of the NIST elliptic curves over prime fields. In *Progress in Cryptology – CT-RSA 2001* (2001), D. Naccache, Ed., vol. 2020 of *Lecture Notes in Computer Science*, pp. 250–265.
3. Certicom Research. Standards for efficient cryptography – SEC 1: Elliptic curve cryptography. Version 1.0, 2000. Available from http://www.secg.org/.
4. Certicom Research. Standards for efficient cryptography – SEC 2: Recommended elliptic curve cryptography domain parameters. Version 1.0, 2000. Available from http://www.secg.org/.
5. Cohen, H., Ono, T., and Miyaji, A. Efficient elliptic curve exponentiation using mixed coordinates. In *Advances in Cryptology – ASIACRYPT '98* (1998), K. Ohta and D. Pei, Eds., vol. 1514 of *Lecture Notes in Computer Science*, pp. 51–65.
6. Coron, J.-S. Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems – CHES '99* (1999), C. K. Koç and C. Paar, Eds., vol. 1717 of *Lecture Notes in Computer Science*, pp. 292–302.
7. Institute of Electrical and Electronics Engineers (IEEE). IEEE standard specifications for public-key cryptography. IEEE Std 1363-2000, 2000.
8. Joye, M., and Quisquater, J.-J. Hessian elliptic curves and side-channel attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2001 [Pre-]Proceedings* (2001), C. K. Koç, D. Naccache, and C. Paar, Eds., pp. 412–420.
9. 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.
10. 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.
11. Liardet, P.-Y., and Smart, N. P. Preventing SPA/DPA in ECC systems using the Jacobi form. In *Cryptographic Hardware and Embedded Systems – CHES 2001 [Pre-]Proceedings* (2001), C. K. Koç, D. Naccache, and C. Paar, Eds., pp. 401–411.

12. Miyaji, A., Ono, T., and Cohen, H. Efficient elliptic curve exponentiation. In *International Conference on Information and Communications Security – ICICS '97* (1997), Y. Han, T. Okamoto, and S. Qing, Eds., vol. 1334 of *Lecture Notes in Computer Science*, pp. 282–290.
13. Montgomery, P. L. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation 48* (1987), 243–264.
14. National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). FIPS PUB 186-2, 2000.
15. Okeya, K., Kurumatani, H., and Sakurai, K. Elliptic curves with the Montgomery-form and their cryptographic applications. In *Public Key Cryptography – PKC 2000* (2000), H. Imai and Y. Zheng, Eds., vol. 1751 of *Lecture Notes in Computer Science*, pp. 238–257.
16. 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.

Addendum: Efficiency Improvement[1]

The efficiency of the technique described in the main text can be improved by using an alternative multiplier recoding method that makes extended use of the ease of inversion in elliptic curve groups (if $[b]P$ is known for some integer b, then $[-b]P$ is available as well). In the following, we sketch the changes.

While the conversion algorithm given in section 3.1 produces digits $-2^w, 1, 2, \ldots, 2^w - 1$, the alternative conversion algorithm produces digits

$$b_i \in \left\{ -2^w, \pm 1, \pm 2, \ldots, \pm(2^{w-1} - 1), 2^{w-1} \right\}.$$

Note that the set of possible digit values still has 2^w elements. The precomputation stage of algorithm 1 in section 3.2 requires 2^{w-1} point doublings and $2^{w-1} - 1$ general point additions to compute the appropriate set of multiples of the input point. If algorithm 1 is changed according to the new set of digits, the precomputation stage will require only $2^{w-2} + 1$ point doublings and $2^{w-2} - 1$ point additions. (I.e., compared with the original algorithm, w can be increased by one at the cost of a single additional precomputation stage point doubling.)

Given input digits b'_i such than $0 \le b'_i < 2^w$, we again use auxiliary values c_i and t_i such that $0 \le c_i \le 2$ and $0 \le t_i \le 2^w + 1$: Let

$$c_0 = 0,$$

and for $i = 0, \ldots, k' + 1$, let

$$t_i = b'_i + c_i$$

and

$$(c_{i+1}, b_i) = \begin{cases} (1, -2^w) & \text{if } t_i = 0 \\ (0, t_i) & \text{if } 0 < t_i \le 2^{w-1} \\ (1, -2^w + t_i) & \text{if } 2^{w-1} < t_i < 2^w \\ (2, -2^w) & \text{if } t_i = 2^w \\ (1, 1) & \text{if } t_i = 2^w + 1. \end{cases}$$

For all i, we have $c_{i+1} \cdot 2^w + b_i = t_i$.

Errata[2]

Section 2.2: If the precomputed table uses Jacobian projective coordinates, then the extended point representation to use is (X, Y, Z, Z^2, Z^3), not (X, Y, Z, Z^2).

[1] Added 2001-08-27/2001-08-29. Does not appear in *ISC 2001* proceedings.
[2] Added 2001-12-21; not in *ISC 2001* proceedings.