Appears in A. Menezes, P. Sarkar (Eds.): *Progress in Cryptology – INDOCRYPT 2002,* Springer-Verlag LNCS 2551, pp. 296–313, ISBN 3-540-00263-4.

Errata added 2003-01-30 and 2005-08-14.

Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks

Tetsuya Izu[1], Bodo Möller[2], and Tsuyoshi Takagi[2]

[1] FUJITSU LABORATORIES Ltd.
4-1-1, Kamikodanaka, Nakahara-ku, Kawasaki, 211-8588, Japan
izu@flab.fujitsu.co.jp
[2] TU Darmstadt, Fachbereich Informatik
Alexanderstr.10, D-64283 Darmstadt, Germany
{moeller,ttakagi}@cdc.informatik.tu-darmstadt.de

Abstract. We improve several elliptic curve multiplication algorithms secure against side channel attacks (SCA). While some efficient SCA-resistant algorithms were developed that apply only to special classes of curves, we are interested in algorithms that are suitable for general elliptic curves and can be applied to the recommended curves found in various standards. We compare the running time and memory usage of the improved schemes.

Keywords: elliptic curve cryptosystems, scalar multiplication, side channel attacks, memory constraints, window method

1 Introduction

Side channel attacks (SCA) [Koc96,KJJ99] allow adversaries to obtain the secret key in a cryptographic device, or partial information on it, by observing information such as computing time and power consumption traces if the implementation is naive or careless. This is a serious threat especially to mobile devices such as smart cards. Thus, implementers need algorithms that are not only efficient, but also SCA-resistant. Power analysis attacks subsume timing attacks, so we will focus on the former. *Simple power analysis* (SPA) utilizes information from a single computation, while *differential power analysis* (DPA) uses statistical tools to evaluate information from multiple computations.

Elliptic curve based cryptosystems (ECC) have gained popularity for cryptographic applications because of the short key length compared with earlier public key cryptosystems such as RSA. They are considered particularly suitable for implementation on smart cards or mobile devices. Because of the physical characteristics of such devices and their use in potentially hostile environments, the power consumption trace or the timing of computations using the secret key can be clearly observed. Thus, side channel attacks are a serious threat against these devices. The main target for side channel attacks against ECC implementation is the algorithm used for scalar multiplication on the elliptic curve. Therefore,

various elliptic curve multiplication algorithms designed to resist side channel attacks have been proposed.

In this paper we deal with DPA-resistant methods that do not require specifically selected elliptic curves and thus can be used for the recommended curves found in [NIST], [ANSI] and [SEC2]. We focus on curves over finite fields of characteristic greater than 3. (While all the methods can be similarly applied to curves over binary fields, our efficiency analysis does not cover this case.) We examine Coron's dummy addition method [Cor99], a method using a non-standard addition chain [Möl01], and a method using the Montgomery ladder [IT02a]. We investigate their security against DPA and analyze their efficiency.

2 Elliptic Curve Arithmetic

Let $K = \mathbb{F}_q$ be the finite field with q elements where q is a power of a prime $p > 3$. Elliptic curves over K are certain subsets of $K^2 \cup \{\mathcal{O}\}$ equipped with an additive group structure; $\mathcal{O}$ denotes the *point at infinity,* the neutral element of addition. Every elliptic curve over K is isomorphic to a curve described in the form

$$E(K) := \{(x, y) \mid y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}, \tag{1}$$

with $a, b \in K$, $4a^3 + 27b^2 \neq 0$, which we call the *Weierstrass form.* Let $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ be two elements of $E(K)$ that are different from $\mathcal{O}$. We have $-P_1 = (x_1, -y_1)$. If $-P_1 \neq P_2$, then the sum $P_1 + P_2 = (x_3, y_3)$ is given by

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1 \tag{2}$$

where $\lambda = (y_2 - y_1)/(x_2 - x_1)$ for $P_1 \neq P_2$, and $\lambda = (3x_1^2 + a)/(2y_1)$ for $P_1 = P_2$. We call computing $P_1 + P_2$ an *elliptic curve addition* (ECADD) if $P_1 \neq \pm P_2$; otherwise if $P_1 = P_2$ we speak of an *elliptic curve doubling* (ECDBL); the remaining case $P_1 = -P_2$ (where $P_1 + P_2 = \mathcal{O}$) should usually be avoided when SCA-resistance is intended. The algorithms for ECADD and ECDBL are usually not same (a general addition algorithm typically will have to detect the special cases when the ECDBL algorithm is called for, or when $P_1 = -P_2$).

Elliptic curve cryptography usually employs curves whose order is the product of a large prime and a very small integer h, the so-called cofactor. In this paper, we assume that this standard scenario is fulfilled. Cryptographic protocols avoid points of small order, i.e. points P such that $hP = \mathcal{O}$. The recommendations of [SEC1] require $h \leq 4$; in practice, the cofactor often is 1.

2.1 Efficiency of Addition and Doubling Algorithms

We estimate the efficiency of ECDBL and ECADD when using Jacobian coordinates, a variant of projective coordinates where triples $(X : Y : Z)$ represent points $(X/Z^2, Y/Z^3)$ on the elliptic curve. This type of coordinates yields the best performance for our purposes. Denote by M, S, and A the time needed for a multiplication, a squaring, and an addition, respectively, in the base field $\mathbb{F}_q$.

(The effort for a subtraction may be considered equivalent to that for an addition.) For the total efficiency of elliptic curve operations, we are interested in the time depending on the individual times M, S, A, and in the amount of auxiliary storage used by the computation.

The coefficient a in the defining polynomial of the curve can be an arbitrary field element. However, many curves recommended by specifications such as [NIST,ANSI,SEC2] use $a = -3$, allowing for a more efficient ECDBL implementation. We assume that a is stored in memory as part of the system parameters. In appendix A.1, we show algorithms for both cases: the general algorithm $\texttt{ECDBL}^{\mathcal{J}}$ requires time $4M + 6S + 11A$ using 6 auxiliary variables; the optimized algorithm $\texttt{ECDBL}^{\mathcal{J},a=-3}$ requires time $4M+4S+13A$ using 5 auxiliary variables.

For ECADD, we consider two cases: the general case of addition of points given in Jacobian coordinates, and the special case where one of the input points has a Z-coordinate of 1, i.e. is represented in affine coordinates. The latter case allows for faster addition [CMO98]; this is known as addition with *mixed coordinates.* Algorithms for both cases are shown in appendix A.2: the general algorithm $\texttt{ECADD}^{\mathcal{J}}$ requires time $12M + 4S + 7A$ using 7 auxiliary variables, and algorithm $\texttt{ECADD}^{\mathcal{J},Z=1}$ for mixed coordinates requires time $8M + 3S + 7A$ using 7 auxiliary variables.

In the scalar multiplication algorithm in section 4, we will have to compute $2^w P$ where P is a point and w is a positive integer. If $\texttt{ECDBL}^{\mathcal{J}}$ is repeatedly applied to compute $2^w P$, we need $4wM + 6wM + 11wA$ operations. Itoh et al. [ITTTK99] proposed a faster algorithm for directly computing $2^w P$, which can be found in appendix A.3. This algorithm $\texttt{wECDBL}_w^{\mathcal{J}}$ requires time $4wM + (4w+2)S + (12w-1)A$ using 7 auxiliary variables. However, in the case $a = -3$, it is more efficient to iterate w times algorithm $\texttt{ECDBL}^{\mathcal{J},a=-3}$, which requires time $4wM + 4wS + 13wA$.

We summarize these results on the efficiency of algorithms for elliptic curve arithmetic in table 1.

Table 1. Computing time and number of auxiliary variables for several algorithms

Algorithm	Time	# of auxiliary variables
$\texttt{ECDBL}^{\mathcal{J}}$	$4M + 6S + 11A$	6
$\texttt{ECDBL}^{\mathcal{J},a=-3}$	$4M + 4S + 13A$	5
$\texttt{ECADD}^{\mathcal{J}}$	$12M + 4S + 7A$	7
$\texttt{ECADD}^{\mathcal{J},Z=1}$	$8M + 3S + 7A$	7
$\texttt{wECDBL}_w^{\mathcal{J}}$	$4wM + (4w+2)S + (12w-1)A$	7

3 Scalar Multiplication and Side Channel Attacks

Let d be a positive integer and P be a point on an elliptic curve $E(K)$. Computing $dP = \sum_{1 \le i \le d} P$ is called a scalar multiplication. Scalar multiplications are used in encryption and decryption or signature generation and verification of elliptic curve cryptosystems. These computations are relatively expensive when implemented on low-power devices.

A standard method for performing scalar multiplications is the left-to-right binary method. We show how to compute dP with this method. Let

$$d = d[k-1]2^{k-1} + \ldots + d[1]2^1 + d[0]2^0$$

be the binary representation of d where k is chosen minimal so that $d[k-1] = 1$. Then, given $d[0], d[1], \ldots, d[k-1]$ and P, we can compute dP as follows.

```
INPUT d, P, (d[0],d[1],..,d[k-1])
OUTPUT d*P
 1: Q = P
 2: for i = k-2 down to 0
 3:   Q = ECDBL(Q)
 4:   if d[i]==1
 5:     Q = ECADD(Q,P)
 6: return Q
```

With the usual algorithms for ECDBL in step 3 and for ECADD in step 5, bit information can be detected by SPA [Cor99]: the power consumption traces of ECDBL and ECADD are not same, so an attacker can easily distinguish between these operations and derive the values $d[i]$.

3.1 SPA-Resistant Scalar Multiplication Methods

We describe several SPA-resistant methods for computing dP. If point P might be chosen by the attacker, we assume that it is rejected if $hP = \mathcal{O}$ where h is the cofactor, as otherwise SCA-resistance may be voided due to the special cases of elliptic curve arithmetic (see section 2). Note that h is usually very small so that hP can be computed with a short fixed sequence of operations.

At first we describe Coron's dummy addition method [Cor99], which is one of standard countermeasures against SPA. We will compare it with other efficient methods in this paper. The algorithm is as follows:

```
INPUT d, P, (d[0],d[1],...,d[k-1])
OUTPUT d*P
 1: Q[0] = P
 2: for i = k-2 down to 0
 3:   Q[0] = ECDBL(Q[0])
 4:   Q[1] = ECADD(Q[0],P)
 5:   Q[0] = Q[d[i]]
 6: return Q[0]
```

(We note that there is a potential security problem with this method [Möl01]. If $d[i] = 0$, the point that is one of the inputs to ECADD in the current iteration will be the input to ECDBL in the next iteration. When using projective coordinates, both ECADD and ECDBL involve squaring the Z coordinate, so the same Z value will be squared again if $d[i] = 0$. Side channels may provide hints that the same squaring is performed again, thus leaking information on $d[i]$.)

As Coron's dummy addition method requires $(k-1)$ ECDBL operations and $(k-1)$ ECADD operations, it is slower than the standard binary method. When we use algorithms $\texttt{ECADD}^{\mathcal{J}}$ and $\texttt{ECDBL}^{\mathcal{J}}$ or $\texttt{ECDBL}^{\mathcal{J},a=-3}$, Coron's dummy addition method requires $12(k-1)M + 9(k-1)S + 18(k-1)A$ for $a \neq -3$ and $12(k-1)M + 7(k-1)S + 20(k-1)A$ for $a = -3$ (not counting conversion of the final result from Jacobian into affine coordinates).

Several SPA-resistant algorithms have been proposed that are faster than Coron's dummy addition method. Three basic approaches are known to achieve SPA resistance:

- The first one is to use indistinguishable addition and doubling algorithms in the scalar multiplication (cf. [CJ01]). Jacobi form and Hesse form elliptic curves achieve this as they allow using the same algorithm for both additions and doublings [LS01,JQ01]. However, this requires specifically chosen elliptic curves and does not work for the standardized curves recommended by specifications such as [NIST], [ANSI] and [SEC2]. Brier and Joye proposed an indistinguishable addition and doubling algorithm applicable to Weierstrass form curves [BJ02], but it fails on certain inputs, making it vulnerable to attacks [IT02b].
- The second one is the so-called double-and-always-add approach. Coron's dummy addition method is the simplest algorithm of this type. Okeya and Sakurai proposed to use Montgomery form elliptic curves to achieve a double-and-always-add method [OS00], but this is not applicable to the standardized curves. This method was recently extended to general curves [BJ02,IT02a].
- The third approach is to use a special addition chain with a sequence of additions and doublings that does not depend on the bit information of the secret key, as proposed by Möller [Möl01]. (Recently, Seysen has proposed a different addition chain secure against the SPA [Sey01].)

In this paper, we are interested in scalar multiplication algorithms that do not require specifically chosen curves. Therefore we will examine Möller's method [Möl01] and Izu and Takagi's method [IT02a].

3.2 Countermeasures against DPA

Even if a scalar multiplication implementation is secure against SPA, it may be possible to break it by using DPA, i.e. by employing statistical tools to analyze the information observed in many executions of the algorithm. However, it is easy to enhance SPA-resistant methods to be DPA-resistant. We describe two approaches, one due to Coron and one due to Joye and Tymen.

One of the countermeasures described by Coron in [Cor99] is *projective randomization:* Let $P = (X : Y : Z)$ be a base point given in Jacobian coordinates; then for all $r \in K \backslash \{0\}$, $(r^2X : r^3Y : rZ)$ represents the same point. If we transform a base point $(X : Y : Z)$ into $(r^2X : r^3Y : rZ)$ with a random r before starting the scalar multiplication, the side channel information available to the statistic analysis will be randomized. The additional computational cost is only $4M + 1S$ at the beginning of the scalar multiplication.

Joye and Tymen proposed a related countermeasure [JT01]. It is based on randomly selected isomorphisms between elliptic curves. The base point $P = (x, y)$ and the defining coefficients a, b of an elliptic curve can be randomized into $P' = (r^2x, r^3y)$ and $a' = r^4a, b' = r^6b$, yielding the corresponding point on an isomorphic curve defined by a', b'. This randomization allows us to keep a Z-coordinate of 1 and thus benefit from mixed coordinates in the scalar multiplication. Joye-Tymen randomization requires $5M + 3S$ in the beginning of the scalar multiplication. At the end of the scalar multiplication, we have to transform the point to the original curve using r. For Jacobian coordinates, transformation from $(X : Y : Z)$ back into affine coordinates $\left(X/(rZ)^2, Y/(rZ)^3\right)$ for the original curve requires $5M + 1S + 1I$. Joye-Tymen randomization requires additional storage: during the scalar multiplication, implementations must store the random field element r; and elliptic curve operations have to be performed using modified coefficients a', b'. Thus, three field elements have to be stored.

Actually the b coefficient is usually not needed for elliptic curve operations. For Joye-Tymen randomization without b, the initial transformations require only $4M + 3S$, and additional storage is needed only for two field elements.

The other countermeasure against DPA is to randomize the computation process of the addition chain [IYTT02,OA01,LS01]. However, the Oswald-Aigner scheme was broken by an SPA proposed by Okeya and Sakurai [OS02a]. Walter showed how to attack the Liardet-Smart method [Wal02].

3.3 Computing Architecture

We discuss the relevant properties of smart card computing architectures (for a more comprehensive description, see [VW98]).

The main components are a central processing unit (CPU), read only memory (ROM), electrically erasable programmable read only memory (EEPROM), random access memory (RAM), and the arithmetic unit (AU). Typical smart card CPUs are variants of the Motorola 6805 or Intel 8051 processor.

The ROM contains the smart card operating system and additional software including the scalar multiplication algorithm. Fixed system parameters can be stored in EEPROM. Writing into EEPROM is very slow (usually on the order of 1 000 times slower than writing or reading RAM). The RAM of smart cards is usually limited to 4 Kbits. The AU includes a coprocessor; this is used for implementing field operations (addition, subtraction, and multiplication).

Assuming that the underlying field $K = \mathbb{F}_q$ is a prime field (which is typically the case), system parameters for most elliptic curve cryptosystems are of the form $(q, a, b, G, \#G, h)$ where (a, b) are the coefficients defining the elliptic curve, G is

a base point generating a prime-order subgroup of the elliptic curve, $\#G$ is the order of G, and $h = \#E(K)/\#G$ is the cofactor. Coefficient b may be omitted if it is not needed for elliptic curve arithmetic or for verifying that externally supplied points actually lie on the curve. These system parameters are fixed, so they can be stored in EEPROM. A fixed secret key may also be stored in EEPROM.

Depending on the cryptographic application, scalar multiplication dP may involve the fixed base point ($P = G$) or ephemeral points. For scalar multiplication methods involving a precomputed table of points, we assume that this table is stored in RAM; as we will explain in section 4.1, using a fixed table might make the implementation vulnerable to DPA.

Finally, we need a random number generator (see section 3.2). Smart cards often provide random number generation through their operation system.

4 Window-Based Method

To minimize the exposure to side channel attacks, elliptic curve scalar multiplication should be implemented using a fixed sequence of operations. We describe Möller's method that achieves this for general elliptic curves [Möl01,Möl01a].[3] We first describe it in a general form; choice of point representations will be discussed in the following security analysis.

This method represents the multiplier d in base 2^w for some window size $w \geq 2$ such that digit value 0 is avoided (except for leading zeros): in the original method [Möl01], digits are from the set $\{-2^w, 1, 2, \ldots, 2^w - 1\}$; in the improved method [Möl01a], digits are from the set $b_i \in \{-2^w, \pm 1, \pm 2, \ldots, \pm(2^{w-1} - 1), 2^{w-1}\}$. Both sets have cardinality 2^w, and for both methods the binary representation of d can easily be transformed into a representation $d = \sum_{i=0}^{k} b_i \cdot 2^{wi}$ using digits b_i from the respective set. (To ensure that k does not depend on the specific multiplier, a small multiple of the group order can be added to d, assuming that the original value of d is bounded above by the order. This way it is easy to achieve $k = \lceil (n+2)/w \rceil$ where n denotes the bit length of the group order.) Then the following algorithm can be used for computing dP if precomputed values $P[b] = bP$ are available:

```
INPUT k, b[], P[]
OUTPUT d*P
 1: A = P[b[k]]
 2: for i = k-1 down to 0
 3:   A = ECADD(A, P[b[i]])
 4:   for j = 1 to w
```

[3] A new window-based algorithm for elliptic curve multiplication with resistance against side channel attacks has recently been described in [Möl02]. It can provide better efficiency than the window-based algorithm from [Möl01,Möl01a] if fixed precomputation for the elliptic curve in question can be used. In the present paper, we assume that we have to work without such precomputation.

```
 5:     A = ECDBL(A)
 4: return A
```
Algorithm 1: Compute dP where $d = \sum_0^k b_i 2^{wi}$ and $P[b] = bP$

Note that $-bP$ can be computed from bP at almost no cost, so it is possible to implement the improved method using a $P[]$ array containing only $2^{w-1} + 1$ elements. Precomputation can be performed as follows (`exp2(w-1)` denotes 2^{w-1}):

```
INPUT w, P
OUTPUT P[]
 1: m = exp2(w-1)
 2: P[1] = P
 3: for i = 2 to m-2 step 2
 4:   P[i] = ECDBL(P[i/2])
 5:   P[i+1] = ECADD(P[i], P)
 6: P[m] = ECDBL(P[m/2])
 7: P[2*m] = ECDBL(P[m])
 5: return P[]
```
Algorithm 2: Precomputation for Algorithm 1

If $w \geq 4$, it is possible to compute array $P[]$ more efficiently than this by exploiting that a combined computation of $(b + b')P$ and $(b - b')P$ from bP and $b'P$ is faster than two separate point additions; see [OK02].

4.1 Security Analysis

If points in the table are represented using affine coordinates, the security of window-based methods against DPA is questionable: because the points in affine coordinate system can be uniquely presented for the given system parameter (p, a, b), we can analyze the power consumption trace of the `ECADD` for a fixed point and we can guess which points are used for the `ECADD`. We show a general attack strategy against table lookup based methods using the DPA if the same scalar is used in many elliptic curve multiplications. In applications such as elliptic curve Diffie-Hellman, the attacker may be able to submit the same point P many times. The feasibility of the attack depends on an implementation of the method. We call the attack a *fixed table attack*.

We briefly describe the attack strategy. Let $(A_1 : B_1 : 1), (A_2 : B_2 : 1)$, $(A_3 : B_3 : 1)$ be the values of points in the table using affine coordinates. The ECADD implementation consists of several base field operations, and we can know the power consumption traces of each base field operation by the help of SPA [Sey01]. When using algorithm $\texttt{ECADD}^{\mathcal{J}}$ or $\texttt{ECADD}^{\mathcal{J},Z=1}$, we always compute A_iX and B_iY for some integers X, Y. During the scalar multiplication, the integers X and Y may be considered random assuming that point A is projectively randomized at the beginning of Algorithm 1. An attacker can gather measurements $Power(A_iX)$ and $Power(B_iY)$ for $i = 1, 2, 3$ and random X, Y from many computations. The length of A_i and B_i is fixed, and we can find the mean value of A_iX and B_iY:

$$Exp(A_i) = \frac{1}{\#S} \sum_{X \in S} Power(A_iX), \qquad Exp(B_i) = \frac{1}{\#S} \sum_{X \in S} Power(A_iX)$$

where S is the set of all sampled points and $\#S$ is the cardinality of set S. Then we can guess which point of the table is used, or we can classify them into three classes based on $Exp(A_i), Exp(B_i)$ for $i = 1, 2, 3$. For example, if A_1 has smaller Hamming weight than A_2, A_3 and $Power(A_iX)$ is positively correlated with the Hamming weight of A_i, the relationship $Exp(A_1) < Exp(A_2), Exp(A_3)$ will hold for large S.

A countermeasure against the fixed table attack is to randomize the points in the table by using Coron's projective randomization method: when using Jacobian coordinates, replace $(X : Y : Z)$ by $(r^2X : r^3Y : rZ)$ where r is a random non-zero field element.

More sophisticated fixed table attacks may apply even if the table is fixed only during each single point multiplication: observations from individual `ECADD` operations performed within each point multiplication may show correlations that indicate whether the same table value is used or not (cf. [WT01,Sch02] for related attacks against RSA).[4] Thus, a projective randomization should be done for each `ECADD`: after each use of a table value, the table should be updated by substituting the randomized point $(r^2X : r^3Y : rZ)$ for the old point $(X : Y : Z)$. This requires an additional $4M + 1S$ for each `ECADD`.

We now analyze the security of randomization in the initial phase. We assume that the above countermeasure against the fixed table attack is used. In the first step of Algorithm 1, a point is assigned to A depending on the digit $b[k]$. In step 3 of Algorithm 1, the addition $\texttt{ECADD}(A, R)$ is carried out for a randomized point R. If the point A is not randomized before the scalar multiplication, the attacker has a statistical advantage for guessing the digit $b[k]$. Therefore point A should be randomized.

When using Algorithm 1 in the improved method with precomputed points $-2^wP, P, 2P, \ldots, (2^{w-1}-1)P, 2^{w-1}P$ and digit set $\{-2^w, \pm 1, \pm 2P, \ldots, \pm(2^{w-1}-1), 2^{w-1}\}$, then when $b[i]$ is a negative digit for which no table entry exists, an addition in the underlying field K must be carried out to invert the Y coordinate of $P\big[-b[i]\big]$ in order to compute the inverted point for use by the `ECADD` in step 3 of Algorithm 1. This may provide the attacker with partial information on digit $b[i]$ if this point inversion can be detected by DPA. A countermeasure is to perform this inversion unconditionally and use either its result or the original value (dummy point inversion).

4.2 Efficiency

We now estimate the efficiency of Möller's method for $w = 2, 3$. We use the algorithms and efficiency estimations discussed in section 2.1. Both the cases $a = -3$ and $a \neq -3$ are considered. As input to the scalar multiplication algorithm, we assume that the following values are given: the definition of the curve (the definition of field $K = \mathbb{F}_q$ and coefficients $a, b \in K$), the base point $P = (x, y)$ represented in affine coordinates, and the scalar d in Möller's representation.

[4] Recently, Okeya and Sakurai proposed a fixed table attack against Möller's scheme using a second order DPA [OS02b]

The points in the precomputed table for $w = 2$ are $-4P, P, 2P$. In order to generate the points, we first compute $2P = \texttt{ECDBL}^{\mathcal{J}}(P), 4P = \texttt{ECDBL}^{\mathcal{J}}(2P)$, and reverse the sign of the Y-coordinate of $4P$, which requires $2(4M + 6S + 11A) + 1A = 8M + 12S + 23A$ for $a \neq -3$ and $2(4M + 4S + 13A) + 1A = 8M + 8S + 27A$ for $a = -3$.

For making Möller's method DPA-resistant, we apply Coron's projective randomization method (see section 4.1). In the beginning of the scalar multiplication, we randomize all the points in the table, namely $-4P, P, 2P$. Because the Z coordinate of P is 1, we require $2(4M + S) + 1(3M + S) = 11M + 3S$. Before each multiplication, we randomize the point $P\big[b[i]\big]$ in the table, which requires $k(4M + 1S)$ in total, where k is the number of $\texttt{ECADD}^{\mathcal{J}}$ operations performed by Algorithm 1.

In the main loop of the scalar multiplication in the case $a \neq -3$, we perform k point inversions and compute k times $\texttt{ECADD}^{\mathcal{J}}$ and k times $\texttt{wECDBL}_w^{\mathcal{J}}$, which requires $kA + k(12M + 4S + 7A) + k(4wM + (4w + 2)S + 12w - 1)A = (12 + 4w)kM + (6 + 4w)kS + (6 + 12w)kA = 20kM + 14kS + 31kA$. In the case $a = -3$, we perform k point inversions and compute k times $\texttt{ECADD}^{\mathcal{J}}$ and kw times $\texttt{ECDBL}^{\mathcal{J},a=-3}$, which requires $kA+k(12M+4S+7A)+kw(4M+4S+13A) = (12 + 4w)kM + (4 + 4w)kS + (7 + 13w)kA = 20kM + 12kS + 34kA$.

After the scalar multiplication, we pull back the point $(X : Y : Z)$ to affine coordinates by computing $(X/Z^2, Y/Z^3)$. This requires $2M + 1S + 1I$.

Consequently, we need $(24k + 21)M + (15k + 16)S + (31k + 23)A + 1I$ for $a \neq -3$ and $(24k+21)M+(13k+12)S+(34k+27)A+1I$ for $a = -3$. For scalars d up to 160 bits, k becomes 81. In this case, we conclude that Möller's method with Coron's projective randomization requires $3005.1M$ for $a \neq 3$ and $2874.8M$ for $a = -3$, assuming that $1S = 0.8M, 1A = 0.01M, 1I = 30M$ [OS01] [MvOV97].

We estimate the memory requirements for the method excluding the system parameters. Assume that n bits are needed to store one element of the underlying field K. The table of the algorithm consists of 3 points represented in Jacobian coordinates, using a total of $(3n) \cdot 3 = 9n$ bits of storage. This is 1440 bits for 160-bit elliptic curve cryptography. The point arithmetic algorithms need 7 auxiliary variables (i.e., $7n$ bits). These also suffice to store point A during the algorithm and to perform projective randomization of table elements.

Similarly, we can estimate the efficiency and memory for $w = 3$. In this case, there are 5 precomputed points: $-8P, P, 2P, 3P, 4P$. Compared with $w = 2$, preparing the table requires one more $\texttt{ECADD}^{\mathcal{J},Z_1=1}$ operation for computing $3P$ (additional cost $8M + 3S + 7A$), one more $\texttt{ECDBL}^{\mathcal{J}}$ or $\texttt{ECDBL}^{\mathcal{J},a=-3}$ operation for computing $4P$ (additional cost $4M + 6S + 11A$ or $4M + 4S + 13A$, respectively), and two more projective randomizations (additional cost $2(4M + S) = 8M + 2S$). The loop body in Algorithm 1 requires one more point doubling (additional cost $4kM + 4kS + 12kA$ for $a \neq -3$ with $\texttt{wECDBL}_w^{\mathcal{J}}$, $4kM + 4kS + 13kA$ for $a = -3$ with $\texttt{ECDBL}^{\mathcal{J},\dashv=-\ni}$). The total computational cost of the scalar multiplications becomes $(28k + 41)M + (19k + 27)S + (43k + 41)A + 1I$ for $a \neq -3$ and $(28k + 41)M + (17k + 21)S + (47k + 47)A + 1I$ for $a = -3$. For scalars up to 160 bits, here we have $k = 54$, so this is $2449.0M$ for $a \neq -3$ and $2360.0M$ for $a = -3$

with the same assumption above. The size of the precomputed table is $15n$ bits (2400 bits for 160-bit ECC).

5 Montgomery-Type Method

Another approach to compute an SCA-resistant scalar multiplication is to use Montgomery's ladder, which was originally proposed in [Mon87] for Montgomery form elliptic curves. Recently, the method also has been applied to Weierstrass form curves [BJ02,IT02a] in order to resist side channel attacks. With this approach, the x-coordinate-only point addition algorithm is employed to minimize the computation time. In this paper, we use the following addition algorithm [IT02a] for efficiency. Let x_1, x_2 be x-coordinate values of two points P_1, P_2 of an elliptic curve $E : y^2 = x^3 + ax + b$. Then the x-coordinate value x_3 of the sum $P_3 = P_1 + P_2$ is given by

$$x_3 = \frac{2(x_1 + x_2)(x_1x_2 + a) + 4b}{(x_1 - x_2)^2} - x_3'$$

where x_3' is the x-coordinate value of $P_3' = P_1 - P_2$. On the other hand, the x-coordinate value x_4 of the doubled point $P_4 = 2P_1$ is given by

$$x_4 = \frac{(x_1^2 - a)^2 - 8bx_1}{4(x_1^3 + ax_1 + b)}.$$

These relations enable us to compute x_d, the x-coordinate value of dP, using only the x-coordinates of points. These formulae are called the (additive) x-coordinate-only addition formulae.

In the original ladder, ECADD and ECDBL are computed separately. For performing SCA-resistant scalar multiplication efficiently, Izu and Takagi [IT02b]. encapsulated these formulae into one formula `xECADDDBL`, which outputs x-coordinate values of $P_3 = P_1 + P_2$ and $P_4 = 2P_1$ on inputs P_1, P_2. In fact, with a projective version of the x-coordinate-only formulae, we can compute X_3, Z_3, X_4, Z_4 with $13M + 4S + 18A$ for $a \neq -3$ and $11M + 4S + 23A$ for $a = -3$. The number of auxiliary variables for the formulae is 7. The concrete algorithms are in appendix A.4 (`xECADDDBL`, `xECADDDBL`$^{a=-3}$).

Algorithm 3 shows an improved Montgomery's ladder. Note that we need $P_3' = P_1 - P_2$ to compute $P_3 = P_1 - P_2$. The following ladder keeps P_3' constant (equal to P).

In the scalar multiplication with the x-coordinate-only formula, an output is the x-coordinate of dP. We need to extra computation to obtain the y-coordinate (y-recovering). $dP = (X_d' : Y_d' : Z_d')$ is computed on input $X_d, Z_d, X_{d+1}, Z_{d+1}$, $P = (x, y)$ as in appendix A.5 (`YRecovering`), which requires $11M + 2S + 7A$ and 7 auxiliary variables.

```
INPUT d, P, (n)
OUTPUT d*P
 1: Q[0] = P, Q[1] = ECDBL(P)
```

```
2: for i = n-2 down to 0
3:   (Q[d[i] XOR 1], Q[d[i]]) = ECADDDBL(Q[d[i]], Q[d[i] XOR 1])
4: return Q[0]
```

Algorithm 3: Improved Montgomery ladder

5.1 Security Analysis

We discuss the security of the improved Montgomery ladder (Algorithm 3). For each bit of Algorithm 3, we always compute ECADDDBL. As a sequence of operations in K, the computation is a fixed pattern unrelated to the bit information `d[]`. Thus the side information becomes a fixed pattern and we conclude that the ladder is secure against the SPA. Note that the security of the ladder is independent of which particular formula is used within the ladder.

In order to enhance the method to be DPA-resistant, we have the Coron and Joye-Tymen countermeasures described in section 3.2. With Coron's projective randomization countermeasure transferred to the x-coordinate-only setting, base point $P = (x : 1)$ is randomized to $(rx : r)$, giving us a DPA-resistant algorithm as side information is randomized. In this situation, Okeya et al. observed that a constant difference P_3' need not be randomized [OMS01]; they claimed that it is secure enough if only the base point is randomized. This approach provides good efficiency. With Joye-Tymen's countermeasure, base point (x, y) would be transformed into (r^2x, y^3y) in order to randomize side channel information. Efficiency is a little worse than Coron's countermeasure. From now on, we assume that Coron's countermeasure is used with Algorithm 3.

5.2 Efficiency

We estimate the efficiency of the improved Montgomery-ladder with Coron's project randomization method. As input to the scalar multiplication algorithm, we assume that the following values are given: the definition of the curve (the definition of field $K = \mathbb{F}_q$ and coefficients $a, b \in K$), the base point $P = (x, y)$ in affine coordinates, and the scalar d in binary representation.

From base point (x, y), we compute two points $P = (rx : r)$ (randomized base point) and $P_3' = (x : 1)$ (constant difference) before applying Algorithm 3, which requires $1M$. We also compute a ECDBL in step 1, which requires $6M + 3S + 9A$ using 5 auxiliary variables. In the main loop of Algorithm 3, we compute $n - 1$ times `xECADDDBL` (or `xECADDDBL`$^{a=-3}$), where n is the bit length of the scalar d. This requires $(13n - 13)M + (4n - 4)S + (18n - 18)A$ for $a \neq -3$ and $(11n - 11)M + (4n - 4)S + (23n - 23)A$ for $a = -3$. After that, y-recovering requires $13M + 4S + 16A$ using 7 auxiliary variables, and the conversion from projective to affine coordinates requires $2M + 1I$.

The total efficiency of the improved Montgomery ladder combined with the x-coordinate-only formula and Coron's projective randomization method is $(13n + 7)M + (4n + 1)S + (18n - 2)A + 1I$ for $a \neq -3$, and $(11n + 9)M + (4n + 1)M + (23n - 7)A + 1I$ for $a = -3$. If we choose a 160-bit scalar, the method requires

Table 2. Computing times of the scalar multiplications

Method	Computing Time	(160-bit ECC)
Coron's dummy	$(12n-7)M+(9n-8)S+(18n-18)A+1I$	$3117M$
Coron's dummy ($a=-3$)	$(12n-7)M+(7n-6)S+(20n-20)A+1I$	$2866M$
Improved Möller ($w=2$)	$(24k_2+21)M+(15k_2+16)S+(31k_2+23)A+1I$	$3005M$
Improved Möller ($w=2, a=-3$)	$(24k_2+21)M+(13k_2+12)S+(34k_2+27)A+1I$	$2875M$
Improved Möller ($w=3$)	$(28k_3+41)M+(19k_3+27)S+(43k_3+41)A+1I$	$2449M$
Improved Möller ($w=3, a=-3$)	$(28k_3+41)M+(17k_3+21)S+(47k_3+47)A+1I$	$2360M$
Improved Izu-Takagi	$(13n+7)M+(4n+1)S+(18n-2)A+1I$	$2659M$
Improved Izu-Takagi ($a=-3$)	$(11n+9)M+(4n+1)S+(23n-7)A+1I$	$2349M$

$2659M$ for $a \neq 3$, and $2349M$ for $a=-3$, under assumptions $1S=0.8M, 1A=0.01M, 1I=30M$ [OS01] [MvOV97].

Let us estimate the required memory excluding the system parameters. Let n be the bit size of the definition field. The registers used by the algorithm are $(X_1:Z_1)$ and $(X_2:Z_2)$. The total bit size of these registers is $4n$, which is 640 bits for $n=160$. The number of auxiliary variables used during the computation is 7, which amounts to 1120 bits for $n=160$.

6 Comparison

In this section, we compare the computing times of a scalar multiplication resistant against the SCA. The Coron dummy addition method with Joye-Tymen randomization in section 3, the improved Möller method with Coron's projective randomization method in section 4.2, and the improved Izu-Takagi method with Coron's projective randomization method in section 5.2 are compared.

In table 2 we summarize the computing time of these methods depending on the group size n (in bits). Both cases of $a \neq -3$ and $a=-3$ are estimated. The last numbers of the rows are estimated for 160-bit ECC, where we assume that $1S=0.8M, 1A=0.01M, 1I=30M$ [OS01] [MvOV97]. Here we use $k_w = \lceil (n+2)/w \rceil$.

In table 3, we summarize the RAM usage of the improved Möller method and the improved Izu-Takagi method.

We see that at 160 bits the improved Izu-Takagi method is more efficient than the improved Möller method with $w=2$ both in terms of computing time and in terms of storage requirements. In the case $a=-3$, it remains faster than the latter method with $w=3$. If sufficient storage is available for the improved Möller method with $w=3$, then in the case $a \neq -3$ this method is the fastest.

Table 3. Comparison of memory usage

Method	RAM usage
Improved Möller ($w=2$)	$7n$ plus $9n$ for the table (2560 bits)
Improved Möller ($w=3$)	$7n$ plus $15n$ for the table (3520 bits)
Improved Izu-Takagi	$7n$ (1120 bits)

References

[ANSI] ANSI X9.62 - 1998, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), 1998.

[BJ02] E. Brier and M. Joye, "Weierstraß Elliptic Curves and Side-Channel Attacks", *PKC 2002,* LNCS 2274, pp. 335–345, Springer-Verlag, 2002.

[CJ01] C. Clavier and M. Joye, "Universal exponentiation algorithm – A first step towards provable SPA-resistance –", *CHES 2001,* LNCS 2162, pp. 300–308, 2001.

[CMO98] H. Cohen, A. Miyaji and T. Ono, "Efficient elliptic curve exponentiation using mixed coordinates", *ASIACRYPT '98,* LNCS 1514, pp. 51–65, 1998.

[Cor99] J. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems", *CHES '99,* LNCS 1717, pp. 292–302, 1999.

[ITTTK99] K. Itoh, et al. "Fast Implementation of Public-Key Cryptography on a DSP TMS320C6201", *CHES '99,* LNCS 1717, pp. 61–72, 1999.

[IYTT02] K. Itoh, J. Yajima, M. Takenaka, and N. Torii, "DPA Countermeasures by improving the Window Method", to appear in *CHES 2002,*, 2002

[IT02] T. Izu and T. Takagi, "A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks", *PKC 2002,* LNCS 2274, pp. 280–296, 2002.

[IT02a] T. Izu and T. Takagi, "A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks", Technical Report CORR 2002-03, University of Waterloo, 2002. Available from `http://www.cacr.math.uwaterloo.ca/`.

[IT02b] T. Izu and T. Takagi, "On the Security of Brier-Joye's Addition Formula for Weierstrass-form Elliptic Curves", TR No. TI-3/02, Technische Universität Darmstadt, 2002. Available from `http://www.informatik.tu-darmstadt.de/TI/`.

[JQ01] M. Joye and J. Quisquater, "Hessian elliptic curves and side-channel attacks", *CHES 2001,* LNCS 2162, pp. 402–410, 2001.

[JT01] M. Joye and C. Tymen, "Protections against differential analysis for elliptic curve cryptography", *CHES 2001,* LNCS 2162, pp. 377–390, 2001.

[Koc96] C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", *CRYPTO '96,* LNCS 1109, pp. 104–113, 1996.

[KJJ99] C. Kocher, J. Jaffe and B. Jun, "Differential power analysis", *CRYPTO '99,* LNCS 1666, pp. 388–397, 1999.

[LS01] P. Liardet and N. Smart, "Preventing SPA/DPA in ECC systems using the Jacobi form", *CHES 2001,* LNCS 2162, pp. 391–401, 2001.

[MvOV97] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*, CRC Press, 1997.

[Möl01] B. Möller, "Securing elliptic curve point multiplication against side-channel attacks", *ISC 2001,* LNCS 2200. pp. 324-334, Springer-Verlag, 2001.

[Möl01a] B. Möller, "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.

[Möl02] B. Möller, "Parallelizable elliptic curve point multiplication method with resistance against side-channel attacks", *ISC 2002,* LNCS 2433. pp. 402-413, 2002.

[Mon87] P. Montgomery, "Speeding the Pollard and elliptic curve methods for factorizations", *Math. Comp.,* vol. 48, pp. 243–264, 1987.

[NIST] National Institute of Standards and Technology, Recommended Elliptic Curves for Federal Government Use, Appendix to FIPS 186-2, 2000.

[OA01] E. Oswald, M. Aigner, "Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks", *CHES 2001*, LNCS2162, pp. 39-50, 2001.

[OK02] H. Oguro and T. Kobayashi, "Efficient Window Method on Elliptic Curve Cryptosystems", *Proceedings of the 2002 Symposium on Cryptography and Information Security, SCIS 2002,* pp. 687–692, 2002 (in Japanese).

[OMS01] K. Okeya, K. Miyazaki, and K. Sakurai, "A Fast Scalar Multiplication Method with Randomized Projective Coordinates on a Montgomery-form Elliptic Curve Secure against Side Channel Attacks", *ICISC 2001*, LNCS 2288, pp.428-439, Springer-Verlag, 2002.

[OS00] K. Okeya and K. Sakurai, "Power analysis breaks elliptic curve cryptosystems even secure against the timing attack", *INDOCRYPT 2000,* LNCS 1977, pp. 178–190, Springer-Verlag, 2000.

[OS01] K. Okeya and K. Sakurai, "Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve", *CHES 2001,* LNCS 2162, pp. 126–141, Springer-Verlag, 2001.

[OS02a] K. Okeya, and K. Sakurai, "On Insecurity of the Side Channel Attack Countermeasure using Addition-Subtraction Chains under Distinguishability between Addition and Doubling", *ACISP 2002*, LNCS2384, pp. 420–435, 2002.

[OS02b] K. Okeya, and K. Sakurai, "A Second-Order DPA Attack Breaks a Window-method based Countermeasure against Side Channel Attacks", *ISC 2002*, LNCS 2433, pp. 389–401, 2002.

[Sch02] W. Schindler, "A Combined Timing and Power Attack", *PKC 2002,* LNCS 2274, pp. 263–279, Springer-Verlag, 2002.

[Sey01] M. Seysen, "DPA-Gegenmaßnahmen bei einer ECDSA-Implementierung auf Chipkarten", presented at DPA Workshop, Bonn (BSI), ECC Brainpool, 2001.

[SEC1] Standards for Efficient Cryptography Group/Certicom Research, SEC 1: Elliptic Curve Cryptography, Version 1.0, 2000. Available from `http://www.secg.org/`.

[SEC2] Standards for Efficient Cryptography Group/Certicom Research, SEC 2: Recommended Elliptic Curve Cryptography Domain Parameters, Version 1.0, 2000.

[VW98] K. Vedder and F. Weikmann, "Smart Cards – Requirements, Properties and Applications –", Chipkarten, Vieweg, pp. 1-23, 1998.

[WT01] C. D. Walter and S. Thompson, "Distinguishing Exponent Digits by Observing Modular Subtractions", *CT-RSA 2001,* LNCS 2020, pp. 192–207, 2001.

[Wal02] C. D. Walter, "Breaking the Liardet-Smart Randomized Exponentiation Algorithm", to appear in CARDIS '02.

A Appendix

We show the concrete algorithms for computing $\texttt{ECDBL}^{\mathcal{J}}$, $\texttt{ECDBL}^{\mathcal{J},a=-3}$, $\texttt{ECADD}^{\mathcal{J}}$, $\texttt{ECADD}^{\mathcal{J},Z=1}$, $\texttt{wECDBL}^{\mathcal{J}}_{w}$, `xECADDDBL`, $\texttt{xECADDDBL}^{a=-3}$, and `YRecovering`, which

are describe in this paper. In order to estimate the efficiency, we use four notations $\times, \cdot^2, +, -$ for the arithmetic of the definition field K. The notation $\times$ is a standard multiplication in K. The notation $\cdot^2$ is a squaring in K. The notations $+$ and $-$ are a multiplication and a subtraction in K, respectively.

A.1 Computing $\texttt{ECDBL}^{\mathcal{J}}$ (left) and $\texttt{ECDBL}^{\mathcal{J},a=-3}$ (right)

$\texttt{ECDBL}^{\mathcal{J}}$, $4M + 6S + 11A$	$\texttt{ECDBL}^{\mathcal{J},a=-3}$, $4M + 4S + 13A$
Input (X_1, Y_1, Z_1, a)	Input (X_1, Y_1, Z_1)
Output (X_2, Y_2, Z_2)	Output (X_2, Y_2, Z_2)
$R_4 \leftarrow X_1, R_5 \leftarrow Y_1, R_6 \leftarrow Z_1$	$R_4 \leftarrow X_1, R_5 \leftarrow Y_1, R_6 \leftarrow Z_1$
$R_1 \leftarrow R_4^2$	$R_2 \leftarrow R_5^2$
$R_2 \leftarrow R_5^2$	$R_2 \leftarrow R_2 + R_2$
$R_2 \leftarrow R_2 + R_2$	$R_3 \leftarrow R_4 \times R_2$
$R_4 \leftarrow R_4 \times R_2$	$R_3 \leftarrow R_3 + R_3$
$R_4 \leftarrow R_4 + R_4$	$R_2 \leftarrow R_2^2$
$R_2 \leftarrow R_2^2$	$R_2 \leftarrow R_2 + R_2$
$R_2 \leftarrow R_2 + R_2$	$R_5 \leftarrow R_5 \times R_6$
$R_3 \leftarrow R_6^2$	$R_5 \leftarrow R_5 + R_5$
$R_3 \leftarrow R_3^2$	$R_6 \leftarrow R_6^2$
$R_6 \leftarrow R_5 \times R_6$	$R_4 \leftarrow R_4 + R_6$
$R_6 \leftarrow R_6 + R_6$	$R_6 \leftarrow R_6 + R_6$
$R_5 \leftarrow R_1 + R_1$	$R_6 \leftarrow R_4 - R_6$
$R_1 \leftarrow R_1 + R_5$	$R_4 \leftarrow R_4 \times R_6$
$R_3 \leftarrow a \times R_3$	$R_6 \leftarrow R_4 + R_4$
$R_1 \leftarrow R_1 + R_3$	$R_4 \leftarrow R_4 + R_6$
$R_3 \leftarrow R_1^2$	$R_6 \leftarrow R_4^2$
$R_5 \leftarrow R_4 + R_4$	$R_6 \leftarrow R_6 - R_3$
$R_5 \leftarrow R_3 - R_5$	$R_6 \leftarrow R_6 - R_3$
$R_4 \leftarrow R_4 - R_5$	$R_3 \leftarrow R_3 - R_6$
$R_1 \leftarrow R_1 \times R_4$	$R_4 \leftarrow R_4 \times R_3$
$R_4 \leftarrow R_1 - R_2$	$R_4 \leftarrow R_4 - R_2$
$X_2 \leftarrow R_5, Y_2 \leftarrow R_4, Z_2 \leftarrow R_6$	$X_2 \leftarrow R_5, Y_2 \leftarrow R_4, Z_2 \leftarrow R_5$

A.2 Computing $\texttt{ECADD}^{\mathcal{J}}$ (left) and $\texttt{ECADD}^{\mathcal{J},Z_1=1}$ (right)

$\texttt{ECADD}^{\mathcal{J}}$, $12M + 4S + 7A$	$\texttt{ECADD}^{\mathcal{J},Z_1=1}$, $8M + 3S + 7A$
Input $(X_1, Y_1, Z_1, X_2, Y_2, Z_2)$	Input $(X_1, Y_1, X_2, Y_2, Z_2)$
Output (X_3, Y_3, Z_3)	Output (X_3, Y_3, Z_3)
$R_2 \leftarrow X_1, R_3 \leftarrow Y_1, R_4 \leftarrow Z_1$	$R_2 \leftarrow X_1, R_3 \leftarrow Y_1, R_5 \leftarrow X_2$
$R_5 \leftarrow X_2, R_6 \leftarrow Y_2, R_7 \leftarrow Z_2$	$R_6 \leftarrow Y_2, R_7 \leftarrow Z_2$
$R_1 \leftarrow R_7^2$	$R_1 \leftarrow R_7^2$
$R_2 \leftarrow R_2 \times R_1$	$R_2 \leftarrow R_2 \times R_1$
$R_3 \leftarrow R_3 \times R_7$	$R_3 \leftarrow R_3 \times R_7$
$R_3 \leftarrow R_3 \times R_1$	$R_3 \leftarrow R_3 \times R_1$
$R_1 \leftarrow R_4^2$	$R_5 \leftarrow R_5 - R_2$
$R_5 \leftarrow R_5 \times R_1$	$R_7 \leftarrow R_5 \times R_7$
$R_6 \leftarrow R_6 \times R_4$	$R_6 \leftarrow R_6 - R_3$
$R_6 \leftarrow R_6 \times R_1$	$R_1 \leftarrow R_5^2$
$R_5 \leftarrow R_5 - R_2$	$R_4 \leftarrow R_6^2$
$R_7 \leftarrow R_4 \times R_7$	$R_2 \leftarrow R_2 \times R_1$
$R_7 \leftarrow R_5 \times R_7$	$R_5 \leftarrow R_1 \times R_5$
$R_6 \leftarrow R_6 - R_3$	$R_4 \leftarrow R_4 - R_5$
$R_1 \leftarrow R_5^2$	$R_1 \leftarrow R_2 + R_2$
$R_4 \leftarrow R_6^2$	$R_4 \leftarrow R_4 - R_1$
$R_2 \leftarrow R_2 \times R_1$	$R_2 \leftarrow R_2 - R_4$
$R_5 \leftarrow R_1 \times R_5$	$R_6 \leftarrow R_6 \times R_2$
$R_4 \leftarrow R_4 - R_5$	$R_1 \leftarrow R_3 \times R_5$
$R_1 \leftarrow R_2 + R_2$	$R_1 \leftarrow R_6 - R_1$
$R_4 \leftarrow R_4 - R_1$	
$R_2 \leftarrow R_2 - R_4$	
$R_6 \leftarrow R_6 \times R_2$	
$R_1 \leftarrow R_3 \times R_5$	
$R_1 \leftarrow R_6 - R_1$	
$X_3 \leftarrow R_4, Y_3 \leftarrow R_1, Z_3 \leftarrow R_7$	$X_3 \leftarrow R_4, Y_3 \leftarrow R_1, Z_3 \leftarrow R_7$

A.3 Computing $\texttt{wECDBL}_{\boldsymbol{w}}^{\mathcal{J}}$

$\texttt{wECDBL}_w^{\mathcal{J}}$, $4wM + (4w+2)S + (12w-1)A$	
Input (X_1, Y_1, Z_1, a)	
Output (X_2, Y_2, Z_2)	
$R_4 \leftarrow X_1, R_5 \leftarrow Y_1, R_6 \leftarrow Z_1$	
$R_1 \leftarrow R_4^2$	*Repeat the following $w-1$ times:*
$R_2 \leftarrow R_5^2$	$R_7 \leftarrow R_2 \times R_7$
$R_2 \leftarrow R_2 + R_2$	$R_7 \leftarrow R_7 + R_7$
$R_4 \leftarrow R_4 \times R_2$	$R_1 \leftarrow R_5^2$
$R_4 \leftarrow R_4 + R_4$	$R_2 \leftarrow R_4^2$
$R_2 \leftarrow R_2^2$	$R_2 \leftarrow R_2 + R_2$
$R_2 \leftarrow R_2 + R_2$	$R_6 \leftarrow R_4 \times R_6$
$R_3 \leftarrow R_6^2$	$R_6 \leftarrow R_6 + R_6$
$R_3 \leftarrow R_3^2$	$R_4 \leftarrow R_5 \times R_2$
$R_6 \leftarrow R_5 \times R_6$	$R_4 \leftarrow R_4 + R_4$
$R_6 \leftarrow R_6 + R_6$	$R_2 \leftarrow R_2^2$
$R_5 \leftarrow R_1 + R_1$	$R_2 \leftarrow R_2 + R_2$
$R_1 \leftarrow R_1 + R_5$	$R_5 \leftarrow R_1 + R_1$
$R_7 \leftarrow a \times R_3$	$R_1 \leftarrow R_1 + R_5$
$R_1 \leftarrow R_1 + R_7$	$R_1 \leftarrow R_1 + R_7$
$R_3 \leftarrow R_1^2$	$R_3 \leftarrow R_1^2$
$R_5 \leftarrow R_4 + R_4$	$R_5 \leftarrow R_4 + R_4$
$R_5 \leftarrow R_3 - R_5$	$R_5 \leftarrow R_3 - R_5$
$R_4 \leftarrow R_4 - R_5$	$R_4 \leftarrow R_4 - R_5$
$R_1 \leftarrow R_1 \times R_4$	$R_1 \leftarrow R_1 \times R4$
$R_4 \leftarrow R_1 - R_2$	$R_4 \leftarrow R_1 - R_2$
$X_2 \leftarrow R_5, Y_2 \leftarrow R_4, Z_2 \leftarrow R_6$	

A.4 Computing xECADDDBL (left) and xECADDDBL$^{a=-3}$ (right)

xECADDDBL, $13M + 4S + 18A$
Input $(X_1, Z_1, X_2, Z_2, x, a, b)$
Output (X_3, Z_3, X_4, Z_4)
$R_1 \leftarrow X_1, R_2 \leftarrow Z_1, R_3 \leftarrow X_2$
$R_4 \leftarrow Z_2$
$R_6 \leftarrow R_1 \times R_4$
$R_1 \leftarrow R_1 \times R_3$
$R_4 \leftarrow R_2 \times R_4$
$R_2 \leftarrow R_3 \times R_2$
$R_3 \leftarrow R_6 - R_2$
$R_3 \leftarrow R_3^2$
$R_5 \leftarrow x \times R_3$
$R_7 \leftarrow a \times R_4$
$R_1 \leftarrow R_1 + R_7$
$R_2 \leftarrow R_2 + R_6$
$R_1 \leftarrow R_1 \times R_2$
$R_2 \leftarrow R_4^2$
$R_7 \leftarrow b \times R_2$
$R_1 \leftarrow R_1 + R_7$
$R_1 \leftarrow R_1 + R_1$
$R_5 \leftarrow R_1 - R_5$
$R_5 \leftarrow R_7 + R_5$
$R_5 \leftarrow R_7 + R_5$
$R_2 \leftarrow a \times R_2$
$R_1 \leftarrow R_6^2$
$R_1 \leftarrow R_1 + R_2$
$R_2 \leftarrow R_2 + R_2$
$R_2 \leftarrow R_1 - R_2$
$R_2 \leftarrow R_2^2$
$R_1 \leftarrow R_6 \times R_1$
$R_7 \leftarrow R_4 \times R_7$
$R_1 \leftarrow R_1 + R_7$
$R_7 \leftarrow R_6 \times R_7$
$R_7 \leftarrow R_7 + R_7$
$R_7 \leftarrow R_7 + R_7$
$R_7 \leftarrow R_7 + R_7$
$R_7 \leftarrow R_2 - R_7$
$R_6 \leftarrow R_4 \times R_1$
$R_6 \leftarrow R_6 + R_6$
$R_6 \leftarrow R_6 + R_6$
$X_3 \leftarrow R_5, Z_3 \leftarrow R_3$
$X_4 \leftarrow R_7, Z_4 \leftarrow R_6$

xECADDDBL$^{a=-3}$, $11M + 4S + 23A$
Input $(X_1, Z_1, X_2, Z_2, x, b)$
Output (X_3, Z_3, X_4, Z_4)
$R_1 \leftarrow X_1, R_2 \leftarrow Z_1, R_3 \leftarrow X_2$
$R_4 \leftarrow Z_2$
$R_6 \leftarrow R_1 \times R_4$
$R_1 \leftarrow R_1 \times R_3$
$R_4 \leftarrow R_2 \times R_4$
$R_2 \leftarrow R_3 \times R_2$
$R_3 \leftarrow R_6 - R_2$
$R_3 \leftarrow R_3^2$
$R_5 \leftarrow x \times R_3$
$R_1 \leftarrow R_1 - R_4$
$R_1 \leftarrow R_1 - R_4$
$R_1 \leftarrow R_1 - R_4$
$R_2 \leftarrow R_2 + R_6$
$R_1 \leftarrow R_1 \times R_2$
$R_2 \leftarrow R_4^2$
$R_7 \leftarrow b \times R_2$
$R_1 \leftarrow R_1 + R_7$
$R_1 \leftarrow R_1 + R_1$
$R_5 \leftarrow R_1 - R_5$
$R_5 \leftarrow R_7 + R_5$
$R_5 \leftarrow R_7 + R_5$
$R_1 \leftarrow R_2 + R_2$
$R_1 \leftarrow R_1 + R_1$
$R_2 \leftarrow R_2 - R_1$
$R_1 \leftarrow R_6^2$
$R_1 \leftarrow R_1 + R_2$
$R_2 \leftarrow R_2 + R_2$
$R_1 \leftarrow R_1 - R_2$
$R_2 \leftarrow R_2^2$
$R_1 \leftarrow R_6 \times R_1$
$R_7 \leftarrow R_4 \times R_7$
$R_1 \leftarrow R_1 + R_7$
$R_7 \leftarrow R_6 \times R_7$
$R_7 \leftarrow R_7 + R_7$
$R_7 \leftarrow R_7 + R_7$
$R_7 \leftarrow R_7 + R_7$
$R_7 \leftarrow R_2 - R_7$
$R_6 \leftarrow R_4 \times R_1$
$R_6 \leftarrow R_6 + R_6$
$R_6 \leftarrow R_6 + R_6$
$X_3 \leftarrow R_5, Z_3 \leftarrow R_3$
$X_4 \leftarrow R_7, Z_4 \leftarrow R_6$

A.5 Computing YRecovering

YRecovering, $11M + 2S + 7A$
Input $(X_d, Z_d, X_{d+1}, Z_{d+1}, x, y, a, b)$
Output (X'_d, Y'_d, Z'_d)
$R_1 \leftarrow X_d, R_2 \leftarrow Z_d, R_3 \leftarrow X_{d+1}, R_4 \leftarrow Z_{d+1}$
$R_5 \leftarrow x \times R_2$
$R_6 \leftarrow R_5 - R_1$
$R_6 \leftarrow R_6^2$
$R_6 \leftarrow R_3 \times R_6$
$R_5 \leftarrow R_5 + R_1$
$R_7 \leftarrow x \times R_1$
$R_1 \leftarrow R_1 \times R_2$
$R_3 \leftarrow a \times R_2$
$R_2 \leftarrow R_2^2$
$R_7 \leftarrow R_3 + R_7$
$R_7 \leftarrow R_5 \times R_7$
$R_5 \leftarrow y \times R_4$
$R_5 \leftarrow R_5 + R_5$
$R_3 \leftarrow R_5 \times R_2$
$R_1 \leftarrow R_5 \times R_1$
$R_2 \leftarrow b \times R_2$
$R_2 \leftarrow R_2 + R_2$
$R_7 \leftarrow R_7 + R_2$
$R_7 \leftarrow R_4 \times R_7$
$R_7 \leftarrow R_7 - R_6$
$X'_d \leftarrow R_1, Y'_d \leftarrow R_7, Z'_d \leftarrow R_3$

Erratum[5]

Algorithm 1 in section 4 should be changed as follows:

```
INPUT k, b[], P[]
OUTPUT d*P
 1: A = P[b[k]]
 2: for i = k-1 down to 0
 3:   for j = 1 to w
 4:     A = ECDBL(A)
 5:   A = ECADD(A, P[b[i]])
 6: return A
```

Algorithm 1: Compute dP where $d = \sum_0^k b_i 2^{wi}$ and $P[b] = bP$

Erratum[6]

Appendix A.4 should read as follows. (The only change is in $\texttt{xECADDDBL}^{a=-3}$, where $R_1 \leftarrow R_1 - R_2$ has been replaced by $R_2 \leftarrow R_1 - R_2$; thanks to Darrel Hankerson for pointing out this typo.)

$\texttt{xECADDDBL}$, $13M + 4S + 18A$	$\texttt{xECADDDBL}^{a=-3}$, $11M + 4S + 23A$
Input $(X_1, Z_1, X_2, Z_2, x, a, b)$	Input $(X_1, Z_1, X_2, Z_2, x, b)$
Output (X_3, Z_3, X_4, Z_4)	Output (X_3, Z_3, X_4, Z_4)
$R_1 \leftarrow X_1, R_2 \leftarrow Z_1, R_3 \leftarrow X_2$	$R_1 \leftarrow X_1, R_2 \leftarrow Z_1, R_3 \leftarrow X_2$
$R_4 \leftarrow Z_2$	$R_4 \leftarrow Z_2$
$R_6 \leftarrow R_1 \times R_4$	$R_6 \leftarrow R_1 \times R_4$
$R_1 \leftarrow R_1 \times R_3$	$R_1 \leftarrow R_1 \times R_3$
$R_4 \leftarrow R_2 \times R_4$	$R_4 \leftarrow R_2 \times R_4$
$R_2 \leftarrow R_3 \times R_2$	$R_2 \leftarrow R_3 \times R_2$
$R_3 \leftarrow R_6 - R_2$	$R_3 \leftarrow R_6 - R_2$
$R_3 \leftarrow R_3^2$	$R_3 \leftarrow R_3^2$
$R_5 \leftarrow x \times R_3$	$R_5 \leftarrow x \times R_3$
$R_7 \leftarrow a \times R_4$	$R_1 \leftarrow R_1 - R_4$
$R_1 \leftarrow R_1 + R_7$	$R_1 \leftarrow R_1 - R_4$
	$R_1 \leftarrow R_1 - R_4$
$R_2 \leftarrow R_2 + R_6$	$R_2 \leftarrow R_2 + R_6$
$R_1 \leftarrow R_1 \times R_2$	$R_1 \leftarrow R_1 \times R_2$
$R_2 \leftarrow R_4^2$	$R_2 \leftarrow R_4^2$
$R_7 \leftarrow b \times R_2$	$R_7 \leftarrow b \times R_2$
$R_1 \leftarrow R_1 + R_7$	$R_1 \leftarrow R_1 + R_7$
$R_1 \leftarrow R_1 + R_1$	$R_1 \leftarrow R_1 + R_1$
$R_5 \leftarrow R_1 - R_5$	$R_5 \leftarrow R_1 - R_5$
$R_5 \leftarrow R_7 + R_5$	$R_5 \leftarrow R_7 + R_5$
$R_5 \leftarrow R_7 + R_5$	$R_5 \leftarrow R_7 + R_5$
	$R_1 \leftarrow R_2 + R_2$
$R_2 \leftarrow a \times R_2$	$R_1 \leftarrow R_1 + R_1$
	$R_2 \leftarrow R_2 - R_1$
$R_1 \leftarrow R_6^2$	$R_1 \leftarrow R_6^2$
$R_1 \leftarrow R_1 + R_2$	$R_1 \leftarrow R_1 + R_2$
$R_2 \leftarrow R_2 + R_2$	$R_2 \leftarrow R_2 + R_2$
$R_2 \leftarrow R_1 - R_2$	$R_2 \leftarrow R_1 - R_2$
$R_2 \leftarrow R_2^2$	$R_2 \leftarrow R_2^2$
$R_1 \leftarrow R_6 \times R_1$	$R_1 \leftarrow R_6 \times R_1$
$R_7 \leftarrow R_4 \times R_7$	$R_7 \leftarrow R_4 \times R_7$
$R_1 \leftarrow R_1 + R_7$	$R_1 \leftarrow R_1 + R_7$
$R_7 \leftarrow R_6 \times R_7$	$R_7 \leftarrow R_6 \times R_7$
$R_7 \leftarrow R_7 + R_7$	$R_7 \leftarrow R_7 + R_7$
$R_7 \leftarrow R_7 + R_7$	$R_7 \leftarrow R_7 + R_7$
$R_7 \leftarrow R_7 + R_7$	$R_7 \leftarrow R_7 + R_7$
$R_7 \leftarrow R_2 - R_7$	$R_7 \leftarrow R_2 - R_7$
$R_6 \leftarrow R_4 \times R_1$	$R_6 \leftarrow R_4 \times R_1$
$R_6 \leftarrow R_6 + R_6$	$R_6 \leftarrow R_6 + R_6$
$R_6 \leftarrow R_6 + R_6$	$R_6 \leftarrow R_6 + R_6$
$X_3 \leftarrow R_5, Z_3 \leftarrow R_3$	$X_3 \leftarrow R_5, Z_3 \leftarrow R_3$
$X_4 \leftarrow R_7, Z_4 \leftarrow R_6$	$X_4 \leftarrow R_7, Z_4 \leftarrow R_6$

[5] Added 2003-01-30. Does not appear in *INDOCRYPT 2002* proceedings.

[6] Added 2005-08-14. Does not appear in *INDOCRYPT 2002* proceedings.