Speeding up Elliptic Curve Scalar Multiplication without Precomputation *

Kwang Ho Kim[1], Junyop Choe[1], Song Yun Kim[1], and Namsu Kim[2] and Sekung Hong[2]

[1] Institute of Mathematics, State Academy of Sciences, Pyongyang, DPR Korea
math.kwangho@star-co.net.kp
[2] PGItech Corp., Pyongyang, DPR Korea
pgitech.namsukim@aliyun.com

Abstract. This paper presents a series of Montgomery scalar multiplication algorithms on general short Weierstrass curves over odd characteristic fields, which need only 12 field multiplications plus $12 \sim 20$ field additions per scalar bit using $8 \sim 10$ field registers, thus significantly outperform the binary NAF method on average. Over binary fields, the Montgomery scalar multiplication algorithm which was presented at the first CHES workshop by López and Dahab has been a favorite of ECC implementors, due to its nice properties such as high efficiency outperforming the binary NAF, natural SPA-resistance, generality coping with all ordinary curves and implementation easiness. Over odd characteristic fields, the new scalar multiplication algorithms are the first ones featuring all these properties. Building-blocks of our contribution are new efficient differential addition-and-doubling formulae and a novel conception of *on-the-fly adaptive coordinates* which softly represent points occurring during a scalar multiplication not only in accordance with the base point but also bits of the given scalar. Importantly, the new algorithms are equipped with built-in countermeasures against known side-channel attacks, while it is shown that previous Montgomery ladder algorithms with the randomized addressing countermeasure fail to thwart attacks exploiting address-dependent leakage.

Keywords: Elliptic curve cryptography · Scalar multiplication · Montgomery ladder · Countermeasures against SCA

1 Introduction

The main burden of elliptic curve cryptosystems (ECC), especially in the Internet of Things (IoT) applications, lies in operation called elliptic curve scalar

* This title follows the ones of two previous breakthrough papers: "Speeding the Pollard and elliptic curve methods of factorization" (Math. Comp. 48, 243-264, 1987) by P.L. Montgomery and "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation" in CHES 1999 by J. López and R. Dahab. This paper along with the additional data has been submitted to CHES 2017.

multiplication (ECSM) that is to compute kP where P is a curve point and k is an integer (also called scalar) served usually as a secret key of the system. In the past decades, the importance of ubiquitous PKI has built up a huge treasury of academic literatures claiming more efficient ECSM implementation methods.

Besides low computational cost and memory usage, there are two main requests for an ECC deployed on resource-constrained devices: scalability to random general curves and resistance to side-channel attacks (SCA). Despite the prominent works that compete ECC speed records by using special curve forms other than general Weierstrass form (e.g., Montgomery form [71, 11], DIK form [25], (twisted) Edwards form [7, 46, 22, 52, 5], (twisted) Hessian form [45], (extended) Jacobi Form [9], GLS curves [33] and so on, see [6] for a comprehensive reference), the reason why ECSM algorithms specific to a special curve family must be avoided for ECC's hardware deployment is multi-fold: 1) for backwards compatibility with already adopted random-curves standards (e.g. [15, 27]); 2) for coping with potential needs of curve exchange in future, in regard that hardware reconfiguration is costly [61, 40]; 3) for the fact that IP cores as generic solutions for all possible clients are preferred for industry, as mentioned in [68]; 4) for the curve isomorphisms countermeasure against differential power analysis (DPA) needs general ECSM [21, 34]; 5) incidentally, for a doubt that the special curves may encounter attacks exploiting their specific structure, as shown in such examples as supersingular curves and anomalous curves [42, 10] (in fact, there also exist some non-trivial security issues related to them [41, 79]). On the other hand, SCA having its origin by Kocher [59] became today the most serious practical threat to ECC through an explosive growth of succeeding works (e.g., [32, 47, 28, 19, 44, 30, 3, 24, 73]). The first rudimentary condition for an ECSM implementation to withstand SCA is the so called SPA-resistance: it must behave regularly in power consumption, independently of scalars. ECSM algorithms with precomputation are not only subject to the zero-value SPA attacks [80, 23, 1], but also unsuitable for implementation on memory-constrained devices such as smart card.

Thus, general-curve-applicable, SPA-resistant, precomputation-less ECSM algorithms have been intensively studied by the ECC community for last decades and also become a subject of this paper. The fastest algorithm of general-curve-applicable and precomputation-less ECSM ones is the binary NAF method [42] and costs on average $6.66M + 7S + 13.33A + E$ per scalar bit using 8 registers. (Throughout this paper, M, S, A, N and E mean the costs of a multiplication, a squaring, an addition, a negation in the base field and the cost to on-the-fly compute a NAF-bit of scalar, respectively. As for the number of field registers, we take into account the in-place register as in [48]. For easy understanding, reader can think of $S = M$ and $A = 0.1M$ for the present.) Then, how much extra should an algorithm with added SPA-resistance cost? There are many previous works to address this problem.

Previous Works. Thirty years ago, as a mean to speed up the elliptic curve method for integer factoring, Montgomery [71] suggested to use a special elliptic curve form over prime fields and developed an addition chain to compute ECSM,

which nowadays are famous as Montgomery form and Montgomery ladder. At the first CHES workshop in 1999, López and Dahab [67] presented a Montgomery ladder based ECSM algorithm for general ECC over binary fields, which is faster than the binary NAF method on average and resists against SPA which started to be extensively explored about that time. The Montgomery ladder as a regular chain attracted ECC researchers [55, 74, 4, 13, 78]. In 2002, Montgomery ladder based ECSM algorithms for general elliptic curves over large characteristic fields have been appeared in the literatures [12, 51, 31, 50]. They outperformed the double-and-add-always method [21] costing $12M+9S+18A$ by 12 field registers, but were not competitive with the binary NAF method or the window-based algorithms [72, 75]: Brier-Joye algorithm and Izu-Takagi algorithm [12, 51] at PKC'02 cost $14M + 5S + 15A$, Fischer-Giraud-Knudsen-Seifert algorithm [31] costs $14M+5S+14A$, and Izu-Möller-Takagi algorithm [50] at INDOCRYPT'02 costs $13M + 4S + 18A$, using 8∼9 field registers.

Subsequently, the atomicity principle introduced by Chevallier-Mames, Ciet and Joye [16] has been found attractive, which consists in rewriting all the operations carried out through NAF-based ECSM into a sequence of identical atomic patterns. While the unified addition formulae by Brier and Joye [12] compute either of a point addition or a point doubling by $13M + 5S$ plus several additions, Chevallier-Mames, Ciet and Joye [16] proposed to compute a curve point doubling (mixed addition) by 10 (11) identical patterns costing $1M + 2A + 1N$. Longa [62, 65] suggested more efficient atomic patterns for $a = -3$ curves. At CARDIS'10, Giraud and Verneuil [34] proposed to write a doubling (mixed addition) as 5 (6) identical patterns costing $1M + 1S + 3A + 2N$ for general curves and better atomicity improvement methods for $a = -3$ curves. By practical implementation results, they also depicted that A could not be neglected in cost evaluation of ECSM algorithms, since A/M ratios on smart cards with crypto-coprocessor lie between $0.36 \sim 0.09$ for 160∼521 bit field sizes.

In 2010-2011 years, many works which were motivated by Meloni's co-Z formulae presentation [69, 70] have been devoted to regular ECSM algorithms based on Montgomery ladder and the Joye double-and-add ([53]). Venelli and Dassande algorithm [83] based on $(X; Y)$-only co-Z point additions reaches a computational cost of $9M + 5S$ plus several additions per scalar bit. At CHES'10, independently of this work, Goundar, Joye and Miyaji [35] applied co-Z arithmetic to the Montgomery ladder and the Joye double-and-add, resulting in an algorithm costing $11M+5S+23A$ ($9M+7S+27A$) per scalar bit using 8 (9) field registers. Bajard, Duquesne, and Ercegovac [2] developed Montgomery scalar multiplication combined with RNS. At AFRICACRYPT'11, Hutter, Joye and Sierra [48] proposed Montgomery ladder (out-of-place) algorithm with $(X; Z)$-only projective co-Z arithmetic which costs $10M + 5S + 13A$ per scalar bit using 10 field registers. In [80], Rivain developed ladder based ECSM algorithms with costs $9M+5S+23A$ ($9M+5S+18A$) and $8M+6S+31A$ ($8M+6S+26A$) per scalar bit using 7 (8) field registers, proposing several trade-offs and saving a few field additions per scalar bit as well as a few memory registers over previous works. A right-to-left window algorithm with a cost $\frac{(4M+4S+11A)\omega+12M+4S+8A}{\omega} + E$ per

bit using $3 \cdot 2^{\omega-1} + 5$ field registers is also presented therein, pointing out its vulnerability to the zero-value point attacks.

At CARDIS'13 [81] (and its extended version [82]), Rondepierre improved atomicity algorithms. His algorithms cost $\frac{4}{3}(8M + 3S + 9A + E)$ per bit for general curves and $\frac{4}{3}(8M + 2S + 10A + E)$ per bit for $a = -3$ curves, where E is the cost to treat (one bit of) the scalar. Their implementation on a smart card shows that the ratio E/M varies between 0.95 and 0.19 for 160∼521 bit sizes and so E should be inevitably considered in cost evaluations for ECSM.

In 2014 Fay [29] and in 2015 Chung, Costello and Smith [18] also worked on this subject, but they have no achieved efficiency improvement. In [11], Bos, Costello, Longa and Naehrig suggested a usage of the regular window method based on the Joye-Tunstall recoding [54] for $a = -3$ curves. At Eurocrypt'16, Renes, Costello and Batina [79] presented complete addition formulae for curves of odd order. All these results are summarized in Table 1 and 2 of Sect. 5, together with our new results. Noticeably, also there have been many works ([77] as the most recent work only) which studied hardware or software implementations of the algorithms enumerated above. In addition, neither of these algorithms is equipped with built-in countermeasures against various SCA developed in [1, 19, 23, 24, 26, 28, 30, 32, 43, 44, 47, 49, 73].

Contributions of this work. We present a series of new fast Montgomery scalar multiplication algorithms which work for general short Weierstrass elliptic curves over arbitrary finite field with characteristic greater than 3. New algorithms cost $8M + 4S + 15.5A$, $8M + 4S + 12.5A$ and $6M + 6S + 20.5A$ per scalar bit using 8, 9 and 10 field registers, respectively, with countermeasures against known Vertical and Horizontal analyses, thus significantly outperform the binary NAF method which costs on average $6.66M + 7S + 13.33A + E$ per scalar bit with 8 registers as well as all previous SPA-resistant algorithms and even be very competitive with the binary NAF method ([80]) on $a = -3$ curves which needs $6.66M + 5S + 14.33A + E$ per scalar bit. It also allows parallelization with high efficiency, for example, costing $4M + 2S + 14.5A$ on two processors.

– Our strategy for the development of Montgomery scalar multiplication is to apply new differential addition-and-doubling (LADD) formulae and novel coordinate systems called *on-the-fly adaptive coordinates* to Montgomery ladder. These two new components have been tailored by following considerations: a Montgomery ladder iterates LADD, which computes either $(P + Q, 2Q)$ or $(2P, P + Q)$ on input of a point pair (P, Q) with the difference $Q - P = (x_0, y_0)$ which is fixed through the ladder. All known LADD formulae do not fully reflect the difference information (x_0, y_0): something involve only x_0 and others involve neither x_0 nor y_0. Since the speciality of LADD executed lies in the fixed difference, new formulae are designed to involve both x_0 and y_0 in an effective way (Sect. 2). Then, we should look for projective coordinates appropriate to the formulae. Today the ECC community knows, besides standard projective coordinates, such a lot of coordinate systems as (Extended, Modified, Relative) Jacobian [17, 29, 20], López-Dahab [66], Inverted Edwards [8], ML [56], XZ [58],

Lambda [76] and so on. While their final goal is to speed up ECSM, they reflect neither the scalar nor the base point, focusing on point arithmetic only. In this sense, all known coordinates are thought to be "hard". In regard to the fact that the fixed difference point is just the base point of ECSM, we conceive *on-the-fly adaptive coordinates* which softly represent the points occurring during a ECSM execution with the given scalar and base point (Sect. 3).

– We show that previous Montgomery ladder algorithms with the randomized addressing countermeasure still fail to thwart attacks exploiting address-dependent leakage. Sequentially, We propose an efficient countermeasure against attacks exploiting address-dependent leakage and an effective method that performs refreshment of blinding-points for free, while making the comparative SCA impossible. Together with these results, the whole code of ECSM equipped with built-in countermeasures against SCA is presented in Sect. 4 so that the readers could easily assimilate it. Section 5 shows performance comparison of this work with previous ones.

2 New Affine LADD Formulae on Short Weierstrass Curves

We consider the short Weierstrass elliptic curve $y^2 = x^3 + \mathfrak{a}x + \mathfrak{b}$ (where $\mathfrak{a}, \mathfrak{b} \in \mathbb{F}_q$ satisfy $4\mathfrak{a}^3 + 27\mathfrak{b}^2 \neq 0$) over finite field $\mathbb{F}_q$ with characteristic greater than 3. We note that all ideas described here can be also applied to elliptic curves ([57]) over ternary fields.

2.1 Affine Differential Addition-and-Doubling Formulae

Lemma 1. *Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be non-zero affine points with a difference $P_2 - P_1 = (x_0, y_0)$ on the elliptic curve $y^2 = x^3 + \mathfrak{a}x + \mathfrak{b}$ over fields of characteristic greater than 3. Let $P_1 \neq \pm P_2$ and neither of the orders of P_1 and P_2 be 2. Then the following holds.*

1) Let $P_3 = P_1 + P_2 = (x_3, y_3)$, $P_5 = 2P_2 = (x_5, y_5)$. Then,

$$\begin{cases} x_3 = x_0 - \frac{16y_1y_2^3}{4y_2^2(x_2-x_1)^2} \quad (x_3 = x_0 - \frac{4y_1y_2}{(x_2-x_1)^2}) \\ y_3 = y_0 + \frac{16y_1y_2^3u}{8y_2^3(x_2-x_1)^3} \quad (y_3 = y_0 + \frac{2y_1u}{(x_2-x_1)^3}) \\ x_5 = \frac{u^2-16y_1y_2^3}{4y_2^2(x_2-x_1)^2} - 2x_3 \\ y_5 = \frac{u(x_5-x_3)}{2y_2(x_2-x_1)} - y_3 \ , \end{cases} \tag{1}$$

where $u = (x_0 - x_2)(x_2 - x_1)^2 + 2y_2(y_2 - y_1)$.

2) Let $P_3 = P_1 + P_2 = (x_3, y_3)$, $P_5 = 2P_1 = (x_5, y_5)$. Then,

$$\begin{cases} x_3 = x_0 - \frac{16y_2y_1^3}{4y_1^2(x_1-x_2)^2} \quad (x_3 = x_0 - \frac{4y_1y_2}{(x_1-x_2)^2}) \\ y_3 = -y_0 + \frac{16y_2y_1^3u}{8y_1^3(x_1-x_2)^3} \quad (y_3 = -y_0 + \frac{2y_2u}{(x_1-x_2)^3}) \\ x_5 = \frac{u^2-16y_2y_1^3}{4y_1^2(x_1-x_2)^2} - 2x_3 \\ y_5 = \frac{u(x_5-x_3)}{2y_1(x_1-x_2)} - y_3 \ , \end{cases} \tag{2}$$

where $u = (x_0 - x_1)(x_1 - x_2)^2 + 2y_1(y_1 - y_2)$.

Proof. 1) By the point addition formulae (see pp. 80 of [42]), we have

$$\begin{cases} x_3 = \left(\frac{y_2-y_1}{x_2-x_1}\right)^2 - x_1 - x_2 \\ y_3 = \frac{y_2-y_1}{x_2-x_1}(x_2 - x_3) - y_2 \end{cases}, \tag{3}$$

$$\begin{cases} x_0 = \left(\frac{y_2+y_1}{x_2-x_1}\right)^2 - x_1 - x_2 \\ y_0 = \frac{y_2+y_1}{x_2-x_1}(x_2 - x_0) - y_2 \end{cases}. \tag{4}$$

Therefore, we get

$$x_3 = x_0 - \frac{4y_1y_2}{(x_2 - x_1)^2} = x_0 - \frac{16y_1y_2^3}{4y_2^2(x_2 - x_1)^2} \ . \tag{5}$$

From (3), (4) and (5), it follows that

$$\begin{aligned} y_3 &= y_0 + \tfrac{(y_2-y_1)(x_2-x_3)-(y_2+y_1)(x_2-x_0)}{x_2-x_1} \\ &= y_0 + \tfrac{y_2(x_0-x_3)+y_1(-2x_2+x_0+x_3)}{x_2-x_1} \\ &= y_0 + \tfrac{y_2\cdot\frac{4y_1y_2}{(x_2-x_1)^2}+y_1(-2x_2+2x_0-\frac{4y_1y_2}{(x_2-x_1)^2})}{x_2-x_1} \\ &= y_0 + \tfrac{2y_1[(x_0-x_2)(x_2-x_1)^2+2y_2(y_2-y_1)]}{(x_2-x_1)^3} \ , \end{aligned}$$

i.e.,

$$y_3 = y_0 + \frac{2y_1u}{(x_2 - x_1)^3} = y_0 + \frac{16y_1y_2^3u}{8y_2^3(x_2 - x_1)^3} \ , \tag{6}$$

where $u = (x_0 - x_2)(x_2 - x_1)^2 + 2y_2(y_2 - y_1)$.

From equations (5) and (6), it follows

$$\frac{y_0-y_3}{x_0-x_3} = \frac{\frac{-2y_1u}{(x_2-x_1)^3}}{\frac{4y_1y_2}{(x_2-x_1)^2}} = \frac{-u}{2y_2(x_2-x_1)} \ . \tag{7}$$

Since $2P_2 = (x_5, y_5) = (P_1 + P_2) + (P_2 - P_1) = P_3 + P_0$, the point addition formulae, (7), (5) and (6) in turn yield

$$\begin{aligned} x_5 &= \left(\tfrac{y_0-y_3}{x_0-x_3}\right)^2 - x_0 - x_3 \\ &= \tfrac{u^2}{4y_2^2(x_2-x_1)^2} - 2x_3 - \tfrac{4y_1y_2}{(x_2-x_1)^2} \\ &= \tfrac{u^2-16y_1y_2^3}{4y_2^2(x_2-x_1)^2} - 2x_3 \end{aligned}$$

and

$$y_5 = \left(\frac{y_0 - y_3}{x_0 - x_3}\right)(x_3 - x_5) - y_3 = \frac{u(x_5 - x_3)}{2y_2(x_2 - x_1)} - y_3 \ .$$

2) In consideration of the commutativity of elliptic curve group, we can exchange P_1 and P_2 in the formulae of case 1), but at this time $P_1 - P_2 = -(P_2 - P_1) = -(x_0, y_0) = (x_0, -y_0)$. □

2.2 Motivations for On-the-fly Adaptive Coordinate Representation

It is common to use projective coordinates in order to trade expensive field inversions for relatively cheap field operations in ECSM. Every loop iteration of Montgomery ladder (Algorithm 1) computes a pair of new curve points from a pair of curve points by LADD. Co-Z Jacobian coordinates [69] which represents a pair of affine points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ as a quintuplet $(X_1 : Y_1 : X_2 : Y_2 : Z)$ where $x_i = \frac{X_i}{Z^2}$ and $y_i = \frac{Y_i}{Z^3}$ for any $i \in \{1, 2\}$ have been recognized to be the best choice for LADD on short Weierstrass elliptic curves over fields of characteristic greater than 3 [37, 35, 36, 80, 29]. In the co-Z Jacobian coordinates, the new LADD formulae can be represented as follows:
(1) Let $P_3 = P_1 + P_2$, $P_5 = 2P_2$ and $P_2 - P_1 = (x_0, y_0)$. Then $(P_3, P_5) = (X_3 : Y_3 : X_5 : Y_5 : Z')$ is computed by $A = 2Y_2 \cdot (X_2 - X_1)$, $Z' = A \cdot Z$, $S = x_0 \cdot Z'^2$, $T = y_0 \cdot Z'^3$, $U = (x_0 Z^2 - X_2) \cdot (X_2 - X_1)^2 + 2Y_2^2 - 2Y_1 \cdot Y_2$, $B = 16 Y_1 Y_2 \cdot Y_2^2$, $X_3 = S - B$, $Y_3 = B \cdot U + T$, $X_5 = U^2 - B - 2X_3$, $Y_5 = U \cdot (X_5 - X_3) - Y_3$.
(2) Let $P_3 = P_1 + P_2$, $P_5 = 2P_1$ and $P_2 - P_1 = (x_0, y_0)$. Then $(P_3, P_5) = (X_3 : Y_3 : X_5 : Y_5 : Z')$ is computed by $A = 2Y_1 \cdot (X_1 - X_2)$, $Z' = A \cdot Z$, $S = x_0 \cdot Z'^2$, $T = y_0 \cdot Z'^3$, $U = (x_0 Z^2 - X_1) \cdot (X_2 - X_1)^2 + 2Y_1^2 - 2Y_1 \cdot Y_2$, $B = 16 Y_1 Y_2 \cdot Y_1^2$, $X_3 = S - B$, $Y_3 = B \cdot U - T$, $X_5 = U^2 - B - 2X_3$, $Y_5 = U \cdot (X_5 - X_3) - Y_3$.

It is noted that the cases (1) and (2) have a similar operation structure, but they differ in the computation of Y_3: for the case (1) BU plus T, and for the case (2) BU minus T.

To decrease the computational cost, first we come up with new coordinates called (x_0, y_0)-*Jacobian coordinates* which be presented below. With $S = x_0 Z^2$ and $T = y_0 Z^3$ in mind, we propose to represent a curve point as a quadruplet $(X : Y : S : T)$. Thus,

$$\begin{array}{l}(X_1 : Y_1 : S_1 : T_1) = (X_2 : Y_2 : S_2 : T_2) \Leftrightarrow \\ \exists \lambda \in \mathbb{F}_q^*, X_1 = \lambda^2 X_2, Y_1 = \lambda^3 Y_2, S_1 = \lambda^2 S_2, T_1 = \lambda^3 T_2 \ .\end{array} \tag{8}$$

More detail, when a curve point (x_0, y_0) with $x_0 y_0 \neq 0$ is given, (x_0, y_0)-*Jacobian coordinates* represent a point on the elliptic curve by quadruplet $(X : Y : S : T)$ satisfying a relation $y_0^2 S^3 = x_0^3 T^2$, which corresponds to the affine point $(\frac{X}{S} \cdot x_0, \frac{Y}{T} \cdot y_0)$. Conversely an affine point (x, y) on the curve has (x_0, y_0)-*Jacobian representation* $(\lambda^2 x : \lambda^3 y : \lambda^2 x_0 : \lambda^3 y_0)$ for any $\lambda \in \mathbb{F}_q^*$.

We can also define so called (x_0, y_0)-*simultaneous Jacobian representation* (*PSJ*) for point pairs. A pair (P_1, P_2) of the projective points $P_1 = (X_1 : Y_1 : S : T)$ and $P_2 = (X_2 : Y_2 : S : T)$ in (x_0, y_0)-*Jacobian coordinates* is represented as septuplet $(X_1 : Y_1 : X_2 : Y_2 : S : T : R)$ of field elements where $R = (X_2 - X_1)^2$, which corresponds to the pair of affine points $(\frac{X_1}{S} \cdot x_0, \frac{Y_1}{T} \cdot y_0)$ and $(\frac{X_2}{S} \cdot x_0, \frac{Y_2}{T} \cdot y_0)$. The affine point pair $((x_1, y_1), (x_2, y_2))$ can be represented as $(\lambda^2 x_1 : \lambda^3 y_1 : \lambda^2 x_2 : \lambda^3 y_2 : \lambda^2 x_0 : \lambda^3 y_0 : \lambda^4 (x_2 - x_1)^2)$ for any $\lambda \in \mathbb{F}_q^*$ in *PSJ*.

In *PSJ* representation, we get new LADD formulae:
(1) Let $(P_1, P_2) = (X_1 : Y_1 : X_2 : Y_2 : S : T : R)$, $P_3 = P_1 + P_2$, $P_5 = 2P_2$ and $P_2 - P_1 = (x_0, y_0)$. Then $(P_3, P_5) = (X_3 : Y_3 : X_5 : Y_5 : S' : T' : R')$ can be computed by $A = (Y_2 + X_2 - X_1)^2 - Y_2^2 - R$, $S' = S \cdot A^2$, $T' = T \cdot A^3$, $U = (S - X_2) \cdot R + 2Y_2^2 - 2Y_1 \cdot Y_2$, $B = 4 \cdot 2Y_1Y_2 \cdot 2Y_2^2$, $X_3 = S' - B$, $Y_3 = B \cdot U + T'$, $X_5 = U^2 - B - 2X_3$, $R' = (X_5 - X_3)^2$, $Y_5 = \frac{1}{2}((U + X_5 - X_3)^2 - U^2 - R') - Y_3$.
(2) Let $(P_1, P_2) = (X_1 : Y_1 : X_2 : Y_2 : S : T : R)$, $P_3 = P_1 + P_2$, $P_5 = 2P_1$ and $P_2 - P_1 = (x_0, y_0)$. Then $(P_3, P_5) = (X_3 : Y_3 : X_5 : Y_5 : S' : T' : R')$ can be computed by $A = (Y_1 + X_1 - X_2)^2 - Y_1^2 - R$, $S' = S \cdot A^2$, $T' = T \cdot A^3$, $U = (S - X_1) \cdot R + 2Y_1^2 - 2Y_1 \cdot Y_2$, $B = 4 \cdot 2Y_1Y_2 \cdot 2Y_1^2$, $X_3 = S' - B$, $Y_3 = B \cdot U - T'$, $X_5 = U^2 - B - 2X_3$, $R' = (X_5 - X_3)^2$, $Y_5 = \frac{1}{2}((U + X_5 - X_3)^2 - U^2 - R') - Y_3$, which cost $7M + 6S + 22A$ and already improve over all previous results including the record cost $8M + 6S + 26A$ in [80]. In next section, we show how to further reduce this cost by introducing a novel conception of point pair representation which changes not only in accordance with the base point but also bits of the scalar.

3 On-the-Fly Adaptive Coordinates for Pairs of Points

In this section we present three faster formulae to compute a loop iteration of Montgomery ladder - line 3 or line 4 of Algorithm 1. First of all, we note that the assumptions in Lemma 1 are always satisfied at every iteration of Algorithm 1 for practical ECC applications: 1) P_1 and P_2 have no order 2 since they must be in the large prime order subgroup $< P_0 >$; 2) $P_2 - P_1 = P_0 \neq \infty$ holds always during ladder; 3) $P_2 = -P_1$ if and only if $P_1 = [(N-1)/2]P_0$ and $P_2 = [(N+1)/2]P_0$ (see 2)) where $N = \mathtt{Ord}_E(P_0)$. For a scalar $k < N$, this occurs if and only if $k = [(N-1)/2] * 2 = N - 1$. Thus, implementors need only forcing input scalars to be $1 < k < N - 1$. (In fact, $k = \pm 1$ must be avoided in cryptographic applications.) This is also a reason why in Sec. 4.2 we choose point blinding as a countermeasure, refraining from scalar blinding.

Algorithm 1. Montgomery ladder
Input: $k = (k_{n-1} \cdots k_1 k_0)$ and $P_0 \in E(\mathbb{F}_q)$
Output: $Q = kP_0$
1: $P_1 = \infty, P_2 = P_0$
2: **for** $i = n - 1$ **downto** 0 **do**
3: **if** $k_i = 1$ **then** $P_1 \leftarrow P_1 + P_2, P_2 \leftarrow 2P_2$
4: **if** $k_i = 0$ **then** $P_2 \leftarrow P_1 + P_2, P_1 \leftarrow 2P_1$
5: **end for**
6: $Q = P_1$

Algorithm 1 runs n times LADD which on input of (P_1, P_2) computes $(2P_1, P_1 + P_2)$ if the current scalar bit is 0 and $(P_1 + P_2, 2P_2)$ if the current scalar bit is 1. Therefore, we introduce a new terminology: "point pair (P_1, P_2) being at bit k_i", which means that the current scalar bit is k_i.

3.1 Definition of On-the-fly Adaptive Simultaneous Coordinates

We propose to represent a pair of (x_0, y_0)-*Jacobian projective* points $P_1 = (X_1 : Y_1 : S : T)$ and $P_2 = (X_2 : Y_2 : S : T)$ as a septuplet $(X_1 : X_2 : K : L : A : S : T)_b$ indexed by b which means bits of scalar considered, where $K = 2Y_1^2, L = -2Y_1Y_2$ and $A = 2Y_1(X_1 - X_2)$ if $b = 0$, and $K = 2Y_2^2, L = -2Y_1Y_2$ and $A = 2Y_2(X_2 - X_1)$ if $b = 1$. We refer to this new point pair representation as *BPSJ* and precisely define as follows:

Definition 2. *(BPSJ) Let (x_0, y_0) be an affine point satisfying $x_0y_0 \neq 0$ on the short Weierstrass curve. The bitwise (x_0, y_0)-simultaneous Jacobian coordinates $(X_1 : X_2 : K : L : A : S : T)_b$ at bit b represent a pair of affine points $(\frac{X_1}{S} \cdot x_0, \frac{K(X_1-X_2)}{AT} \cdot y_0)$ and $(\frac{X_2}{S} \cdot x_0, \frac{L(X_2-X_1)}{AT} \cdot y_0)$ when $b = 0$, and a pair of affine points $(\frac{X_1}{S} \cdot x_0, \frac{L(X_1-X_2)}{AT} \cdot y_0)$ and $(\frac{X_2}{S} \cdot x_0, \frac{K(X_2-X_1)}{AT} \cdot y_0)$ when $b = 1$. It holds that $y_0^2S^3 = x_0^3T^2$ and $K = \frac{A^2}{2(X_1-X_2)^2}$. Conversely, a pair of affine points (x_1, y_1) and (x_2, y_2) has bitwise (x_0, y_0)-simultaneous Jacobian representation $(\lambda^2x_1 : \lambda^2x_2 : 2\lambda^6y_{1+b}^2 : -2\lambda^6y_1y_2 : 2\lambda^5y_{1+b}(x_{1+b} - x_{2-b}) : \lambda^2x_0 : \lambda^3y_0)_b$ for any $\lambda \in \mathbb{F}_q^*$ and any $b \in \{0, 1\}$.*

3.2 Differential Addition-and-Doubling Formulae in *BPSJ*

Theorem 3. *Let $(P_1, P_2) = (X_1 : X_2 : K : L : A : S : T)_b$ be a pair of non-zero points at bit b in BPSJ. Let $P_1 \neq \pm P_2$ and neither of the orders of P_1 and P_2 be 2. Let us set $(P_1', P_2') = (2P_1, P_1 + P_2)$ if $b = 0$ and $(P_1', P_2') = (P_1 + P_2, 2P_2)$ if $b = 1$. Then, we have $(P_1', P_2') = (X_1' : X_2' : K' : L' : A' : S' : T')_{b'}$ at bit b' in BPSJ, which can be computed by:*
1) if $b = 0$ and $b' = 0$, then

$$\begin{array}{l} S' = S \cdot A^2, T' = T \cdot A^3, X_2' = S' + 4K \cdot L, \\ U = (S - X_1) \cdot (X_1 - X_2)^2 + K + L, X_1' = U^2 - S' - X_2', \\ V = U \cdot (S' - X_1') - T', K' = 2V^2, A' = 2V \cdot (X_2' - X_1'), \\ C = U \cdot A', \bar{C} = -C, L' = K' - C \ ; \end{array} \tag{9}$$

2) if $b = 0$ and $b' = 1$, then

$$\begin{array}{l} S' = S \cdot A^2, T' = T \cdot A^3, X_2' = S' + 4K \cdot L, \\ U = (S - X_1) \cdot (X_1 - X_2)^2 + K + L, X_1' = U^2 - S' - X_2', \\ V = U \cdot (S' - X_2') - T', K' = 2V^2, A' = 2V \cdot (X_2' - X_1'), \\ C = U \cdot A', \bar{C} = -C, L' = K' - \bar{C} \ ; \end{array} \tag{10}$$

3) if $b = 1$ and $b' = 0$, then

$$\begin{array}{l} S' = S \cdot A^2, T' = T \cdot A^3, X_1' = S' + 4K \cdot L, \\ U = (S - X_2) \cdot (X_2 - X_1)^2 + K + L, X_2' = U^2 - S' - X_1', \\ V = U \cdot (X_1' - S') - T', K' = 2V^2, A' = 2V \cdot (X_2' - X_1'), \\ C = U \cdot A', \bar{C} = -C, L' = K' - \bar{C} \ ; \end{array} \tag{11}$$

4) if $b = 1$ and $b' = 1$, then

$$\begin{array}{l} S' = S \cdot A^2, T' = T \cdot A^3, X_1' = S' + 4K \cdot L, \\ U = (S - X_2) \cdot (X_2 - X_1)^2 + K + L, X_2' = U^2 - S' - X_1', \\ V = U \cdot (X_2' - S') - T', K' = 2V^2, A' = 2V \cdot (X_2' - X_1'), \\ C = U \cdot A', \bar{C} = -C, L' = K' - C \ . \end{array} \tag{12}$$

Proof. See Appendix A.

As seen in the theorem, the new formulae have been designed to perform a sequence of identical field operations independently of the bit pair (b, b'). The *BPSJ*-LADD based on Theorem 3 costs $8M + 4S + 15A$ plus a field negation using 8 field registers (A detailed algorithm description is given in Sect. 4). We recommend to use this algorithm for utterly resource-constrained hardware implementations where only an adder unit and a multiplier unit are supported. In environments where field halving and squaring units are additionally supported, we will recommend faster formulae as described in the next subsection. Given a field element **a**, halving **a** means to compute an element **b** such that **b**+**b**=**a**.

3.3 Faster Formulae in Modified Versions of BPSJ

$8M + 4S + 12.5A$ Formulae Using Nine Registers: The first modified version (referred as *BPSJ-v1*) of *BPSJ* has doubled K-, L-, T-coordinates and an additional coordinate D compared to the original *BPSJ*. This version decreases number of field additions by using a field halving unit. In *BPSJ-v1*, a point pair (P_1, P_2) at bit b has a representation $(X_1 : X_2 : K : L : A : S : T : D)_b$, where $D = X_2 - X_1, T = 2y_0 Z^3$, and if $b = 0$ then $K = 4Y_1^2, L = -4Y_1 Y_2, A = -2Y_1 D$, and if $b = 1$ then $K = 4Y_2^2, L = -4Y_1 Y_2, A = 2Y_2 D$, in relation to *PSJ*.

To be more precise, *BPSJ-v1* versus affine correspondence is defined as follows: (with relations $4y_0^2 S^3 = x_0^3 T^2$ and $K = \frac{A^2}{(X_1 - X_2)^2}$)

$(X_1 : X_2 : K : L : A : S : T : D)_0 \longrightarrow \left(\left(\frac{X_1}{S} x_0, \frac{-KD}{AT} y_0\right), \left(\frac{X_2}{S} x_0, \frac{LD}{AT} y_0\right)\right)$;

$(X_1 : X_2 : K : L : A : S : T : D)_1 \longrightarrow \left(\left(\frac{X_1}{S} x_0, \frac{-LD}{AT} y_0\right), \left(\frac{X_2}{S} x_0, \frac{KD}{AT} y_0\right)\right)$;

$\big((x_1, y_1), (x_2, y_2)\big) \longrightarrow (\lambda^2 x_1 : \lambda^2 x_2 : 4\lambda^6 y_{1+b}^2 : -4\lambda^6 y_1 y_2 : 2\lambda^5 y_{1+b}(x_{1+b} - x_{2-b}) : \lambda^2 x_0 : 2\lambda^3 y_0 : \lambda^2(x_2 - x_1))_b$ for any $\lambda \in \mathbb{F}_q^*$ and any $b \in \{0, 1\}$.

Corollary 4. *Let $(P_1, P_2) = (X_1 : X_2 : K : L : A : S : T : D)_b$ be a non-zero point pair at bit b in BPSJ-v1. Let $P_1 \neq \pm P_2$ and neither of the orders of P_1 and P_2 be 2. Let us set $(P_1', P_2') = (2P_1, P_1 + P_2)$ if $b = 0$ and $(P_1', P_2') = (P_1 + P_2, 2P_2)$ if $b = 1$. Then, we have $(P_1', P_2') = (X_1' : X_2' : K' : L' : A' : S' : T' : D')_{b'}$ at bit b' in BPSJ-v1, which can be computed by:*
$U = 2(S - X_{1+b}) \cdot D^2 + K + L, S' = S \cdot A^2, T' = T \cdot A^3, X_{2-b}' = S' + K \cdot L,$
$X_{1+b}' = \left(\frac{1}{2}U\right)^2 - S' - X_{2-b}', V = U \cdot (-1)^b (S' - X_{1+b'}') - T', K' = V^2,$
$D' = X_2' - X_1', A' = V \cdot D', C_1 = U \cdot A', C_2 = -C_1, L' = K' - C_{1+b\oplus b'}$.

In prime field of odd characteristic p, halving a field element consists in an one-bit-right-shift with probability $\frac{p+1}{2p}$, or an addition-with-p plus an one-bit-right-shift with probability $\frac{p-1}{2p}$. On the other hand, negating a field element consists

in subtracting it from p. Therefore, it may be reasonable to price an halving as A and a negation as $0.5A$ [63]. In this view, the *BPSJ-v1* differential addition-and-doubling based on Corollary 4 costs $8M+4S+12.5A$ using 9 registers (The detailed algorithm appears in Appendix B).

$6M + 6S + 20.5A$ Formulae Using Ten Registers: As an application of a standard trick, by extending the *BPSJ* $(X_1 : X_2 : K : L : A : S : T)_b$ to ninefold coordinates $(X_1 : X_2 : K : L : A : S : T : R : W)_b$ with $R = (X_2 - X_1)^2$ and $W = A^2$, we can trade two field multiplications for two field squarings in the preceding formulae. We refer to this extended representation as *BPSJ-v2*. New formulae stated below need a cost $6M + 6S + 20.5A$ by ten field registers. The detailed algorithm along with two-processors version needing $4M + 2S + 14.5A$ is provided in Appendix C.

Corollary 5. *Let* $(P_1, P_2) = (X_1 : X_2 : K : L : A : S : T : R : W)_b$ *be a non-zero point pair at bit* b *in BPSJ-v2. Let* $P_1 \neq \pm P_2$ *and neither of the orders of* P_1 *and* P_2 *be 2. Let us set* $(P'_1, P'_2) = (2P_1, P_1 + P_2)$ *if* $b = 0$ *and* $(P'_1, P'_2) = (P_1 + P_2, 2P_2)$ *if* $b = 1$. *Then, we have* $(P'_1, P'_2) = (X'_1 : X'_2 : K' : L' : A' : S' : T' : R' : W')_{b'}$ *at bit* b' *in BPSJ-v2, which can be computed by:*
$U = (S - X_{1+b}) \cdot R + K + L, S' = S \cdot W, T' = T \cdot A \cdot W, X'_{2-b} = S' + 4K \cdot L,$
$C = U^2, X'_{1+b} = C - S' - X'_{2-b}, V = U \cdot (-1)^b (S' - X'_{1+b'}) - T',$
$B = V^2, K' = 2B, D = X'_1 - X'_2, R' = D^2, A' = (V - D)^2 - B - R',$
$W' = A'^2, L' = K' - \frac{1}{2}((U + (-1)^{b \oplus b'} A')^2 - C - W')$.

4 Implementing Elliptic Curve Scalar Multiplication

Algorithm 2. Montgomery scalar multiplication (original version with PCR only)

Input: A curve point $P = (x_0, y_0) \in E(\mathbb{F}_q)$ with $x_0 \cdot y_0 \neq 0$ and a scalar $k = (1 k_{n-2} \cdots k_1 k_0) \in$ `(1,`$\mathrm{Ord}_E(P) - 1$`)`
Output: $Q = k \cdot P$
1: `T` $\leftarrow$ `Setup`$(x_0, y_0, k_{n-2}$,`T`)
2: **for** $i = n - 2$ **downto** 1 **do**
3: `T` $\leftarrow$ `Update`$(k_i, k_{i-1}$,`T`)
4: **end for**
5: `T` $\leftarrow$ `Update`$(k_0, 1$,`T`)
6: `(T[0],T[1])` $\leftarrow$ `Recovery`$(x_0, y_0$,`T`)
7: **return** Q=`(T[0],T[1])`

Algorithm 2 and Algorithm 3 use a temporary register array T[l] of size l which is 8, 9, or 10 depending on LADD formulae exploited. Note that the condition $x_0 \cdot y_0 \neq 0$ on input point $P = (x_0, y_0)$ is coerced by not only new formulae but also Zero-value Point Attack [28]. The new ECSM algorithm consists of three major functions: `Setup`, `Update` and `Recovery`. `Setup` treats the initial setting with Projective Coordinates Randomization (PCR) countermeasure [21]. Here,

we need formulae that can setup $(P, 2P)$ at bits 0 and 1 with the same sequence of field operations, since $(P, 2P)$ should be at k_{n-2} which is a bit in the secret k. By the point doubling formulae (pp. 80 of [42]), the point pair $(P, 2P)$ in affine representation is given as

$$((x_0, y_0), (\frac{(3x_0^2 + \mathfrak{a})^2 - 8x_0y_0^2}{4y_0^2}, \frac{(3x_0^2 + \mathfrak{a})(12x_0y_0^2 - (3x_0^2 + \mathfrak{a})^2) - 8y_0^4}{8y_0^3})) \ .$$

Thus, by setting $\lambda = 2y_0Z^4$ for a random $Z \in \mathbb{F}_q$ in Definition 1, this can be represented in *BPSJ* as follows: $(P, 2P)_b = (X_1 : X_2 : K : L : A : S : T)_b$, where $X_1 = S = 4x_0Z^2 \cdot (y_0Z^3)^2, T = \bar{N}, \bar{N} = 8(y_0Z^3)^4, X_2 = (3(x_0Z^2)^2 + \mathfrak{a}Z^4)^2 - 2X_1$, $N = \bar{N} - (3(x_0Z^2)^2 + \mathfrak{a}Z^4) \cdot (X_1 - X_2)$, $L = 2N \cdot \bar{N}$, and $K = 2\bar{N}^2$ and $A = 2\bar{N} \cdot (X_1 - X_2)$ if $b = 0$, and $K = 2N^2$ and $A = 2N \cdot (X_1 - X_2)$ if $b = 1$. The formulae cost $8M + 7S + 15A$.

`Update(x,y,T)` for `x,y`$\in \{0, 1\}$ computes a new point pair at the bit `y` from a point pair at the bit `x` by using the formulae introduced in Sect. 3. Given a point pair $(Q, Q + P)_1$ at bit 1 in *BPSJ* representation, `Recovery` computes Q's affine coordinates, which are returned as the output of scalar multiplication.

To help readers to check the action of Algorithm 2, below we place `Setup`, `Update` and `Recovery` routines written in C code by which we implemented. These codes should also be useful for ECC hardware practitioners.

```
void Setup(xx,yy,b,T) { Mov(T[0],xx); Mov(T[1],yy); T[2] = rand();
Sqr(T[3],T[2]);Mul(T[4],T[0],T[3]);Mul(T[5],T[1],T[3]);Mul(T[1],T[5],T[2]);
Sqr(T[7],T[1]);Mul(T[0],T[4],T[7]);Add(T[0],T[0],T[0]);Add(T[0],T[0],T[0]);
Sqr(T[6],T[7]);Add(T[6],T[6],T[6]);Add(T[6],T[6],T[6]);Add(T[6],T[6],T[6]);
Sqr(T[5],T[4]);Add(T[4],T[5],T[5]);Add(T[4],T[4],T[5]);Sqr(T[5],T[3]);
Mul(T[7],T[5],cu_a);Add(T[7],T[4],T[7]);Sqr(T[1],T[7]);Sub(T[1],T[1],T[0]);
Sub(T[1],T[1],T[0]);Sub(T[2],T[0],T[1]);Mul(T[4],T[7],T[2]);
Sub(T[5],T[6],T[4]);Mul(T[3],T[5],T[6]);Add(T[3],T[3],T[3]);
Mul(T[4],T[6-b],T[2]);Add(T[4],T[4],T[4]);
Sqr(T[2],T[6-b]);Add(T[2],T[2],T[2]);Mov(T[5],T[0]);}

void Update(u,v,T){Sub(T[1-u],T[u],T[1-u]);Sub(T[u],T[5],T[u]);
Sqr(T[7],T[1-u]);Mul(T[1-u],T[u],T[7]);Sqr(T[u],T[4]);Mul(T[7],T[u],T[5]);
Mov(T[5],T[7]);Mul(T[7],T[4],T[6]);Mul(T[6],T[u],T[7]);Mul(T[4],T[2],T[3]);
Add(T[2],T[2],T[3]);Add(T[3],T[1-u],T[2]);Add(T[4],T[4],T[4]);
Add(T[4],T[4],T[4]);Add(T[1-u],T[4],T[5]);Sqr(T[4],T[3]);Sub(T[4],T[4],T[5]);
Sub(T[u],T[4],T[1-u]);Sub(T[4],T[5*(1-u)+u*v],T[5*u+(1-u)*v]);
Sub(T[7],T[1],T[0]);Mul(T[2],T[3],T[4]);Sub(T[2],T[2],T[6]);
Mul(T[4],T[2],T[7]);Add(T[4],T[4],T[4]);Sqr(T[7],T[2]);Add(T[2],T[7],T[7]);
Mul(T[7],T[3],T[4]);Neg(T[3],T[7]);Sub(T[3],T[2],T[7-4*(u^v)]);}

void Recovery(xx,yy,T){ Sub(T[1],T[0],T[1]);Mul(T[2],T[3],T[1]);
Mul(T[7],T[4],T[6]);Mul(T[6],T[7],T[5]);Inv(T[1],T[6]);Mul(T[3],T[1],T[7]);
Mul(T[7],T[3],T[0]);Mul(T[0],T[7],xx);Mul(T[4],T[5],T[1]);
Mul(T[3],T[4],T[2]);Mul(T[1],T[3],yy);}
```

In the codes, `Mul(z,x,y)` is a void type function which computes `z=x`$\cdot$`y` on input of `x` and `y`, and `Sqr(z,x)` computes `z=x`^{2} on input `x`, where `x,y,z` $\in \mathbb{F}_q$.

`Add(z,x,y)` computes `z=x+y` on input of `x` and `y`. `Sub(z,x,y)` computes `z=x-y` on input of `x` and `y`. `Neg(z, x)` computes `z=-x` on input `x`. `Inv(z, x)` computes `z=x`$^{-1}$ on input `x` with a definition $0^{-1} := 0$. `Mov(z, x)` copies the input `x` to `z`. `^` denotes the bitwise-exclusive-OR operator. In the `Setup` routine `cu_a` means the curve coefficient `a` and `rand()` a RNG which outputs non-zero field elements uniformly at random. As in [48], all given functions make only use of out-of-place operations. Algorithm 2 costs $I + 17M + 7S + 16A + (n-1)(8M + 4S + 15.5A)$, where I is a field inversion.

4.1 Randomization Thwarting Template and Horizontal Attacks

Recently it has been known that the template and horizontal analysis attacks can break the blinded regular ECSM algorithms even with a single side-channel trace [19, 44, 30, 43, 3, 24, 73]. In SAC'13, Bauer, Jaulmes, Prouff, Reinhard and Wild [3] and in 2015, Danger, Guilley, Hoogvorst, Murdica and Naccache [24] proposed practical horizontal collision attacks against the atomicity and unified formulae countermeasures, based on the Big Mac principle. Ladder-based algorithms have been known to be broken by the localized electromagnetic analysis attack [44] which exploits register location based leakage using a high-resolution inductive EM probe, and by the cmov attack [73] which targets address-dependent information leakage. Some countermeasures are presented in [49, 44, 60], which consist in randomizing addressing of registers, repeatedly during execution.

Attacks. Now, we will show that ladder-based algorithms equipped with the countermeasures proposed in [49, 44] remain to be vulnerable to the address-bit DPA attacker [49] and the cmov attacker [73] which will be denoted by $\mathcal{A}$: Algorithm 8 on pp. 388 of [49] is claimed to implement Montgomery ladder in such a way that be resistant against address-bit DPA. But, the algorithm is broken by $\mathcal{A}$: for every $0 \leq i \leq n-2$, $\mathcal{A}$ extracts $d_i \oplus r_i$ from the line 6 of i-indexed loop iteration, $d_{i-1} \oplus r_i$ from the line 4 of $(i-1)$-indexed loop iteration, and thus $(d_i \oplus r_i) \oplus (d_{i-1} \oplus r_i) = d_i \oplus d_{i-1}$, i.e., $\mathcal{A}$ can get all secret key bits d_i only with a bit guess (for d_0); The combination of Algorithm 2 on pp. 237 and Algorithm 3 on pp. 242 of [44] is claimed to implement Montgomery ladder in such a way that renders the localized EM analysis ineffective, with overhead of $6A$ per scalar bit. However, for every i, at the i-indexed loop iteration, $\mathcal{A}$ can extract $c = swapstate \oplus r$ from the line 12 of Algorithm 3, $swapstate \oplus d_i$ from line 3 of Algorithm 2. $\mathcal{A}$ can also get $r \oplus d_{i-1}$ from line 3 of Algorithm 2 at the $(i-1)$-indexed loop iteration. Thus, $\mathcal{A}$ get $(swapstate \oplus r) \oplus (swapstate \oplus d_i) \oplus (r \oplus d_{i-1}) = d_i \oplus d_{i-1}$. $\mathcal{A}$ obtains all secret key bits d_i only with a bit guess; Evidently, $\mathcal{A}$ can break Algorithm 4 of [60] without any hardness.

Countermeasure. We propose to combine the addressing randomization with the so called operation-order randomization which consists in changing the positions of operations involving secret-dependent addressing at random during ECSM execution. The proof-of-concept is done through development of Algorithm 3 that is a randomized version of Algorithm 2: one can see that the 19-th and 20-th operations in `Update` can be swaped without affecting the function output. Also, in regard to the loop iteration, the last and first operations can be

swaped. Therefore, in addition to swapping the `T[0]` and `T[1]` at random, we perform these two swaps at random during ECSM execution.

Algorithm 3 seems to overcomes the template attacks in [44, 73] and the collision attacks in [43, 30], because all secret-dependent conditional moves and register assignments are randomized. `rand_bit()` in the codes shown below means a function which outputs a random bit. Note that `swap,s,r` and `t` are bit variables, so that our countermeasure is essentially for free.

Algorithm 3. Montgomery scalar multiplication (randomized version)

Input: A curve point $P = (x_0, y_0) \in E(\mathbb{F}_q)$ with $x_0 \cdot y_0 \neq 0$ and
a scalar $k = (1k_{n-2} \cdots k_1 k_0) \in$ `(1,`$\mathtt{Ord}_E(P) - 1$)

Output: $Q = k \cdot P$

1: `(T,swap)` $\leftarrow$ `randSetup`$(x_0, y_0, k_{n-2}$,`T,swap`)
2: **for** $i = n - 2$ **downto** 1 **do**
3: `(T,swap)` $\leftarrow$ `randUpdate`$(k_i, k_{i-1}$,`T,swap`)
4: **end for**
5: `(T,swap)` $\leftarrow$ `randUpdate`$(k_0, 1$,`T,swap`)
6: `(T[0],T[1])` $\leftarrow$ `randRecovery`$(x_0, y_0$,`T,swap`)
7: **return** $Q =$`(T[0],T[1])`

```
void randSetup(xx,yy,b,T,swap){ s=rand_bit(); r= rand_bit();
Mov(T[0],xx);Mov(T[1],yy);T[2]=rand();Sqr(T[3],T[2]);Mul(T[4],T[0],T[3]);
Mul(T[5],T[1],T[3]);Mul(T[1],T[5],T[2]);Sqr(T[7],T[1]);Mul(T[0],T[4],T[7]);
Add(T[0],T[0],T[0]);Add(T[s],T[0],T[0]);Sqr(T[6],T[7]);Add(T[6],T[6],T[6]);
Add(T[6],T[6],T[6]);Add(T[6],T[6],T[6]);Sqr(T[5],T[4]);Add(T[4],T[5],T[5]);
Add(T[4],T[4],T[5]);Sqr(T[5],T[3]);Mul(T[7],T[5],cu_a);Add(T[7],T[4],T[7]);
Sqr(T[1-s],T[7]);Sub(T[1-s],T[1-s],T[s]);Sub(T[1-s],T[1-s],T[s]);
Sub(T[2],T[s],T[1-s]);Mul(T[4],T[7],T[2]);Sub(T[5],T[6],T[4]);
Mul(T[3+r],T[5+(1-b)*r],T[6-4*r]);Mul(T[4-r],T[(6-b)-(1-b)*r],T[2+4*r]);
Add(T[3],T[3],T[3]);Add(T[4],T[4],T[4]);Sqr(T[7-5*r],T[5+(1-b)*r]);
Sqr(T[2+5*r],T[(6-b)-(1-b)*r]);Add(T[2],T[2],T[2]);Mov(T[5],T[s]);
Sub(T[(1-b^s)*(1-r)+7*r],T[(b^s)*(1-r)],T[(1-b^s)+r*(b^s)]);
Sub(T[(1-b^s)*r+7*(1-r)],T[(b^s)*r],T[1-r*(b^s)]); swap=s;}

void randUpdate(u,v,T,swap) { s = swap;t=rand_bit();r=rand_bit();
Sub(T[u^s],T[5],T[u^s]);Sqr(T[7],T[1-u^s]);Mul(T[1-u^s],T[u^s],T[7]);
Sqr(T[u^s],T[4]);Mul(T[7],T[u^s],T[5]);Mov(T[5],T[7]);Mul(T[7],T[4],T[6]);
Mul(T[6],T[u^s],T[7]);Mul(T[4],T[2],T[3]);Add(T[2],T[2],T[3]);
Add(T[3],T[1-u^s],T[2]);Add(T[4],T[4],T[4]);Add(T[4],T[4],T[4]);
Add(T[1-u^t],T[4],T[5]);Sqr(T[4],T[3]);Sub(T[4],T[4],T[5]);
Sub(T[u^t],T[4],T[1-u^t]);
Sub(T[4+3*r],T[(5+((v^t)-5)*u)*(1-r)+(1-t)*r],T[((v^t)*(1-u)+5*u)*(1-r)+t*r]);
Sub(T[7-3*r],T[(1-t)*(1-r)+(5+((v^t)-5)*u)*r],T[t*(1-r)+((v^t)*(1-u)+5*u)*r]);
Mul(T[2],T[3],T[4]);Sub(T[2],T[2],T[6]);Mul(T[4],T[2],T[7]);
Add(T[4],T[4],T[4]);Sqr(T[7],T[2]);
Add(T[2],T[7],T[7]);Mul(T[7],T[3],T[4]);Neg(T[3],T[7]);
Sub(T[3*(1-r)+(1-v^t)*r],T[2*(1-r)+(v^t)*r],T[(7-4*(u^v))*(1-r)+(1-v^t)*r]);
Sub(T[(1-v^t)*(1-r)+3*r],T[(v^t)*(1-r)+2*r],T[(1-v^t)*(1-r)+(7-4*(u^v))*r]);
```

```
swap=t;}

void randRecovery(xx,yy,T,swap) { s=swap; Sub(T[s],T[1-s],T[s]);
Sub(T[1-s],T[s],T[1-s]);Mul(T[2],T[3],T[1-s]);Mul(T[7],T[4],T[6]);
Mul(T[6],T[7],T[5]);Inv(T[1-s],T[6]); Mul(T[3],T[1-s],T[7]);
Mul(T[7],T[3],T[s]); Mul(T[4],T[5],T[1-s]); Mul(T[0],T[7],xx);
Mul(T[3],T[4],T[2]); Mul(T[1],T[3],yy);}
```

4.2 Applying Built-In Countermeasures against SCA

Combining Montgomery ladder based ECSM algorithms with PCR, base point blinding and point-validity check before and after ECSM has been known to prevent classical Vertical Analysis attacks such as SPA, timing, DPA and the zero-value attack, and also prevent Fault Attacks such as invalid point, C safe-error, M safe-error and differential fault attack [28, 26, 38, 39]. For variable-base ECSM (by a fixed scalar) which be needed for the key establishment and ElGamal encryption applications, we propose to use a point out of an ECSM computation for blinding the base point at next ECSM: the system keeps a point pair (G, H) where $H = kG$ for the secret scalr k. At the initial setting of system, G is selected at random, $H = kG$ is computed, and then (G, H) is stored inside the device. (G, H) is refreshed after each new ECSM execution through Algorithm 4. This method for point blinding is cheaper than Coron's doubling method [21], and makes applications of the Doubling Attack [32] and the Comparative Power Analysis [47] impossible. Such countermeasures as randomized operation [19] and elimination of conditional branches [64] can be further applied in the field arithmetic level.

Algorithm 4. Montgomery scalar multiplication
(version with built-in SCA countermeasures)

Input: A curve point $P = (x_0, y_0) \in E(\mathbb{F}_q)$ with $x_0 \cdot y_0 \neq 0$, a scalar $k \in$ (1,$\mathrm{Ord}_E(P) - 1$) and a point pair (G, H) with $H = kG$
Output: $Q = k \cdot P$
1: **if** $P \notin E(\mathbb{F}_q)$ or a special point, **then** no response; **else**, then goto Step 2
2: $I \leftarrow P + G$ // Blinding the base point P
3: $J \leftarrow k \cdot I$ // By Algorithm 3
4: $Q \leftarrow J - H$
5: **if** $Q \in E(\mathbb{F}_q)$ goto Step 6, **else** no response
6: $b \leftarrow$ `rand_bit()`
7: $G \leftarrow (-1)^b I$, $H \leftarrow (-1)^b J$
8: **return** Q

5 Comparison

In Table 1 and Table 2, we compare efficiency of our new algorithm with previous SPA-resistant algorithms.

In Table 1, number of registers to store the curve coefficients and base points are not taken into account. As in [48], for algorithms that involve in-place operations, we take into account that one more (in-place) register is needed besides claimed. A field multiplication by constant c is denoted as M_c. We assume $M_{4\mathfrak{b}} = M_{\mathfrak{b}} + 2A = M + 2A$, $M_{3\mathfrak{a}} = M_{\mathfrak{a}} + 2A$, $M_{-3} = 2A$ and $M_{\mathfrak{a}} = M$ for a random $\mathfrak{a}$.

In Table 2, for window algorithms with online precomputations, we assume the window width $\omega = 3$, since for fields of 521 bits (which is a field size specified by the NIST standard) a field element occupies 66 bytes and a 2 Kbyte SRAM (which is usual for 8-bit microprocessors) carries at most 31 field elements, thus no $3 \cdot 2^{\omega-1} + 5$ field elements for $\omega \geq 4$ besides data missing cache. Taking into account that the precomputations in window algorithms cost $48M + 24S + 57A$ for general $\mathfrak{a}$ [80] and $21M + 13S + 28A$ for $\mathfrak{a} = -3$ [11], we set $E' = 0.2M + E = 0.6M$. Note that we ignore the window algorithms using mixed coordinates [80] because they need one more inversion which increases the cost per bit about by M [11]. Regarding to the previous implementation results [34, 82, 14, 5] and our software implementation results (the source code for ECSM on NIST P-384 curve will be given as data additional to this paper by a separate file), we consider three cases: (1) $\frac{S}{M} = 1$, $\frac{A}{M} = 0.2$, (2) $\frac{S}{M} = 0.8$, $\frac{A}{M} = 0.2$, and (3) $\frac{S}{M} = 0.67$, $\frac{A}{M} = 0.1$.

As seen in Table 2, among the previous SPA-resistant algorithms the regular window algorithms are fastest using relatively large memory, but it must be stressed that they are vulnerable to the zero-value attacks (see Sec. 4 of [80]) and the cache timing attacks (e.g. [11]): each of the linear pass, random permutation and loop randomization as countermeasures demands a non-trivial cost. Atomicity and unified formulae based algorithms are broken by the big Mac attack [24]. As shown in Sec. 4, the previous ladder based algorithms are vulnerable to the template attacks.

The new ECSM algorithm with built-in countermeasures against SCA is 9∼13% faster than the regular window and binary NAF algorithms for general curves, even competitive with algorithms specific to the $\mathfrak{a} = -3$ curves. Our smart card and desktop implementations (details of which have no place here by the limit of pages) prove the theoretical improvement of the new algorithms, given in Table 1 and Table 2.

6 Conclusion

It has been a long-standing open problem whether there exists a regular scalar multiplication algorithm on general elliptic curves over odd characteristic fields which should require no precomputation and cost less than the binary NAF method, like Montgomery scalar multiplication on binary elliptic curves. This work solves the problem. New presented algorithms with countermeasures against known Vertical and Horizontal analyses cost $8M + 4S + 15.5A$, $8M + 4S + 12.5A$ and $6M + 6S + 20.5A$ per scalar bit using 8, 9 and 10 field registers, respectively, for arbitrary elliptic curves over any fields with characteristic greater than 3, thus significantly outperform the binary NAF method as well as all previous

Table 1. Comparison of costs per bit and memory usages between SPA-resistant scalar multiplication algorithms for short Weierstrass curves. Algorithms specific to a special curve family are marked by (*). E is the cost of a secured on-the-fly NAF scalar recoding. E' equals E plus the precomputation cost per scalar bit. A field negation is assumed to cost $0.5A$.

Algorithm	Cost per scalar bit	#field regs.
Classic		
Double-addition-always [21, 80]	$11M + 9S + M_a + 18A$	12
Window algorithm based on Joye-Tunstall recoding		
Rivain [80]	$\frac{(4M+4S+11A)\omega+12M+4S+8A}{\omega} + E'$	$3 \cdot 2^{\omega-1} + 5$
Rivain* [80]	$\frac{(4M+4S+12A)\omega+12M+4S+8A}{\omega} + E'$	$3 \cdot 2^{\omega-1} + 4$
Bos-Costello-Longa-Naehrig* [11]	$\frac{(4M+4S+8A)\omega+12M+1S+5A}{\omega} + E'$	$5 \cdot 2^{\omega-1} + 8$
Indistinguishable formulae		
Unified formulae [12]	$16M + \frac{20}{3}S + \frac{4}{3}M_a + \frac{40}{3}A + \frac{4}{3}E$	N/A
Complete formulae* [79, 68]	$16M + 4M_a + \frac{8}{3}M_b + 36A + \frac{4}{3}E$	N/A
Atomicity		
Chevallier-Mames-Ciet-Joye [16]	$\frac{41}{3}M + \frac{205}{6}A + \frac{41}{3}E$	10
Longa* [62]	$6M + 6S + 24A + 6E$	12
Giraud-Verneuil [34]	$7M + 7S + 28A + 7E$	11
Giraud-Verneuil* [34]	$10M + \frac{10}{3}S + \frac{50}{3}A + \frac{5}{3}E$	11
Rondepierre [82]	$\frac{32}{3}M + 4S + 12A + \frac{4}{3}E$	11
Rondepierre* [82, 81]	$\frac{32}{3}M + \frac{8}{3}S + \frac{40}{3}A + \frac{4}{3}E$	11
Montgomery ladder and Joye's double-add		
Brier-Joye [12]	$10M + 5S + 2M_a + 2M_b + 15A$	N/A
Izu-Takagi [51]	$10M + 5S + 2M_a + 2M_b + 15A$	18
Fischer-Giraud-Knudsen-Seifert [31]	$10M + 5S + 2M_a + 2M_b + 14A$	9
Izu-Möller-Takagi [50]	$10M + 4S + 2M_a + 1M_b + 18A$	8
Goundar-Joye-Miyaji Alg.13 [36, 35]	$9M + 7S + 27A$	9
Goundar-Joye-Miyaji Alg.14 [36]	$8M + 7S + 3M_a + 1M_b + 27A$	8
Venelli-Dassance [83]	$9M + 5S + (\mathrm{N/A})A$	N/A
Hutter-Joye-Sierra Alg.5 [48]	$9M + 5S + 1M_a + 1M_b + 16A$	8
Hutter-Joye-Sierra Alg.6 [48]	$10M + 5S + 13A$	10
Goundar-Joye-Miyaji-Rivain-Venelli [37]	$9M + 5S + 19A$	10
Rivain Alg.13+Alg.14 [80]	$9M + 5S + 18A$	8
Rivain suggestion [80]	$8M + 6S + 26A$	8
Fay [29]	$13M + 5S + 14A$	8
Chung-Costello-Smith [18]	$8M + 7S + 2M_a + 3M_b + 12A$	N/A
This work (Alg. 3, Sect. 4)	$\mathbf{8M+4S+15.5A}$	**8**
This work (Alg. 3, Appendix B)	$\mathbf{8M+4S+12.5A}$	**9**
This work (Alg. 3, Appendix C)	$\mathbf{6M+6S+20.5A}$	**10**
Binary NAF method vulnerable to SPA: for the purpose of comparison only		
[42, 50]	$\frac{20}{3}M + 7S + \frac{40}{3}A + E$	8

Table 2. Historical comparison of costs per scalar bit between SPA-resistant scalar multiplication algorithms on short Weierstrass curves, depending on various A/M and S/M ratios (*Case 1*: $\frac{S}{M} = 1, \frac{A}{M} = 0.2$; *Case 2*: $\frac{S}{M} = 0.8, \frac{A}{M} = 0.2$; *Case 3*: $\frac{S}{M} = 0.67, \frac{A}{M} = 0.1$). Only the fastest algorithm in each work is shown. Costs for $\mathfrak{a} = -3$ occur in parenthesis. $E = 0.4M$ [82] and $E' = 0.6M$ be assumed.

Algorithm	general $\mathfrak{a}$ ($\mathfrak{a} = -3$)	*Case 1*	*Case 2*	*Case 3*
DA-always [21, 80]	$12M + 9S + 18A$ $(12M + 7S + 19A)$	$24.6M$ $(22.8M)$	$22.8M$ $(21.4M)$	$19.83M$ $(18.59M)$
U-formulae [12]	$\frac{52}{3}M + \frac{20}{3}S + \frac{40}{3}A + \frac{4}{3}E$ $(16M + \frac{20}{3}S + 16A + \frac{4}{3}E)$	$27.2M$ $(26.4M)$	$25.86M$ $(25.06M)$	$23.66M$ $(22.6M)$
BJ [12], IT [51]	$14M + 5S + 15A$ $(12M + 5S + 19A)$	$22.0M$ $(20.8M)$	$21.0M$ $(19.8M)$	$18.85M$ $(17.25M)$
FGKS [31]	$14M + 5S + 14A$ $(12M + 5S + 18A)$	$21.8M$ $(20.6M)$	$20.8M$ $(19.6M)$	$18.75M$ $(17.15M)$
IMT [50]	$13M + 4S + 18A$ $(11M + 4S + 22A)$	$20.6M$ $(19.4M)$	$19.8M$ $(18.6M)$	$17.48M$ $(15.88M)$
CCJ [16]	$\frac{41}{3}M + \frac{205}{6}A + \frac{41}{3}E$	$25.96M$	$25.96M$	$22.55M$
Longa [62]	$(6M + 6S + 24A + 6E)$	$(19.2M)$	$(18.0M)$	$(14.82M)$
GV [34]	$7M + 7S + 28A + 7E$ $(10M + \frac{10}{3}S + \frac{50}{3}A + \frac{5}{3}E)$	$22.4M$ $(17.33M)$	$21.0M$ $(16.66M)$	$17.29M$ $(14.56M)$
GJM [35, 36]	$9M + 7S + 27A$	$21.4M$	$20.0M$	$16.39M$
HJS [48]	$10M + 5S + 13A$	$17.6M$	$16.6M$	$14.65M$
RGJMVD [80, 37, 83]	$9M + 5S + 18A$	$17.6M$	$16.6M$	$14.15M$
Rondepierre [82, 81]	$\frac{32}{3}M + 4S + 12A + \frac{4}{3}E$ $(\frac{32}{3}M + \frac{8}{3}S + \frac{40}{3}A + \frac{4}{3}E)$	$17.6M$ $(16.53M)$	$16.8M$ $(16.0M)$	$15.08M$ $(14.32M)$
$\omega = 3$ Window [80] ([11])	$8M + \frac{16}{3}S + \frac{41}{3}A + E'$ $(8M + \frac{13}{3}S + \frac{29}{3}A + E')$	$16.66M$ $(14.86M)$	$15.6M$ $(14.0M)$	$13.54M$ $(12.47M)$
This work	$\mathbf{8M+4S+12.5A}$	$\mathbf{14.5M}$	$\mathbf{13.7M}$	$\mathbf{11.93M}$
Binary NAF [42, 50]	$\frac{20}{3}M + 7S + \frac{40}{3}A + E$ $(\frac{20}{3}M + 5S + \frac{43}{3}A + E)$	$16.73M$ $(14.93M)$	$15.33M$ $(13.93M)$	$13.09M$ $(11.85M)$

SPA-resistant ECSM algorithms and even be very competitive with the binary NAF method on $\mathfrak{a} = -3$ curves. For general elliptic curves over ternary fields, similar algorithms exist, which we will present in a subsequent paper.

We hope the ECC community would be encouraged to investigate optimized hardware implementations of algorithms presented in this paper, practical side-channel attacks against the new algorithms, and applications of new developed techniques in various scenarios (e.g., special curve forms, hyperelliptic curves, multi-dimensional Montgomery ladders [74, 4, 13, 78]).

Acknowledgment. The authors would like to thank the anonymous reviewers of CHES 2017 conference for their helpful comments.

References

1. Akishita, T., Takagi, T.: Zero-value point attacks on elliptic curve cryptosystem. In: Boyd, C., Mao, W. (eds.) ISC 2003. LNCS, vol. 2851, pp. 218-233. Springer, Heidelberg (2003)
2. Bajard, J.C., Duquesne, S., Ercegovac, M.: Combining leak-resistant arithmetic for elliptic curves defined over $\mathbb{F}_p$ and RNS representation. Cryptology ePrint Archive, Report 2010/311 (2010), http://eprint.iacr.org/
3. Bauer, A., Jaulmes, E., Prouff, E., Reinhard, J.R., Wild, J.: Horizontal collision correlation attack on elliptic curves - extended version. Cryptography and Communications 7(1), 91-119 (2015)
4. Bernstein, D.J.: Differential addition chains (2006). http://cr.yp.to/ecdh/diffchain-20060219.pdf.
5. Bernstein, D.J., Chuengsatiansup, C., Lange, T.: Double-base scalar multiplication revisited. Cryptology ePrint Archive, Report 2017/037 (2017), http://eprint.iacr.org/
6. Bernstein, D.J., Lange, T.: Explicit formulas database. http://www.hyperelliptic.org/EFD/
7. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 29-50. Springer, Heidelberg (2007)
8. Bernstein, D.J., Lange, T.: Inverted Edwards coordinates. In: Boztas, S., Lu, H. (eds.) Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. LNCS, vol. 4851 , pp. 20-27. Springer, Heidelberg (2007)
9. Billet, O., Joye, M.: The Jacobi model of an elliptic curve and side-channel analysis. In: Fossorier, M., Hoeholdt, T., Poli, A. (eds.) Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. LNCS, vol. 2643, pp. 34-42. Springer, Heidelberg (2003)
10. Blake, I.F., Seroussi, G., Smart, N.P.: Advances in Elliptic Curve Cryptography. Cambridge University Press, Cambridge (2005)
11. Bos, J.W., Costello, C., Longa, P., Naehrig, M.: Selecting elliptic curves for cryptography: An efficiency and security analysis. J. Cryptogr. Eng. 6(4), 259-286 (2015)
12. Brier, É., Joye, M.: Weierstras elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 335-345. Springer, Heidelberg (2002)
13. Brown, D.R.L.: Multi-dimensional Montgomery ladders for elliptic curves. Cryptology ePrint Archive, Report 2006/220 (2006), http://eprint.iacr.org/

14. Brown, M., Hankerson, D., López, J., Menezes, A.: Software implementation of the NIST elliptic curves over prime fields. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 250-265. Springer, Heidelberg (2001)
15. Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0. (January 2010), http://www.secg.org/
16. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. Cryptology ePrint Archive, Report 2003/237 (2003), http://eprint.iacr.org/
17. Chudnovsky, D.V., Chudnovsky, G.V.: Sequences of numbers generated by addition informal groups and new primality and factorization tests. Adv. Appl. Math. 7, 385-434 (1986)
18. Chung, P.N., Costello, C., Smith, B.: Fast, uniform, and compact scalar multiplication for elliptic curves and genus 2 Jacobians with applications to signature schemes. Cryptology ePrint Archive, Report 2015/983 (2015), http://eprint.iacr.org/
19. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal correlation analysis on exponentiation. In: Soriano, M., Qing, S., López, J. (eds.) ICISC 2010. LNCS, vol. 6476, pp. 46-61. Springer, Heidelberg (2010)
20. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation using mixed coordinates. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 51-65. Springer, Heidelberg (1998)
21. Coron, J.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Ç.K., Paar, C. (Eds.) CHES 1999. LNCS, vol. 1717, pp. 292-302. Springer, Heidelberg (1999)
22. Costello, C., Longa, P.: FourQ: four-dimensional decompositions on a Q-curve over the mersenne prime. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 214-235. Springer, Heidelberg (2015).
23. Courrège, J.-C., Feix, B., Roussellet, M.: Simple power analysis on exponentiation revisited. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 65-79. Springer, Heidelberg (2010)
24. Danger, J.-L., Guilley, S., Hoogvorst, P., Murdica, C., Naccache, D.: Improving the big Mac attack on elliptic curve cryptography. Cryptology ePrint Archive, Report 2015/819 (2015), http://eprint.iacr.org/
25. Doche, C., Icart, T., Kohel, D.R.: Efficient scalar multiplication by isogeny decompositions. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 191-206. Springer, Heidelberg (2006)
26. Domínguez-Oviedo, A., Hasan, M.A., Ansari, B.: Fault-based attack on Montgomery's ladder ECSM algorithm. Technical report CARC 2010-12, University of Waterloo, Centre for Applied Cryptographic Research (2010)
27. ECC Brainpool. ECC Brainpool Standard Curves and Curve Generation (2005). http://www.ecc-brainpool.org/download/Domain-parameters.pdf
28. Fan, J., Guo, X., Mulder, E.D., Schaumont, P., Preneel, B., Verbauwhede, I.: State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures. In: IEEE Int. Workshop on Hardware-Oriented Security and Trust, HOST, pp. 30-41 (2010)
29. Fay, B.: Double-and-add with relative Jacobian coordinates. Cryptology ePrint Archive, Report 2014/1014 (2014), http://eprint.iacr.org/
30. Feix, B., Roussellet, M., Venelli, A.: Side-channel analysis on blinded regular scalar multiplications. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 3-20. Springer, Heidelberg (2014)

31. Fischer, W., Giraud, C., Knudsen, E.W., Seifert, J.-P.: 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), http://eprint.iacr.org/
32. Fouque, A.P., Valette, F.: The doubling attack - why upwards is better than downwards. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 269-280. Springer, Heidelberg (2003)
33. Galbraith, S. D., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 518-535. Springer, Heidelberg (2009)
34. Giraud, C., Verneuil, V.: Atomicity improvement for elliptic curve scalar multiplication. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 80-101. Springer, Heidelberg (2010)
35. Goundar, R.R., Joye, M., Miyaji, A.: Co-Z addition formulae and binary ladders on elliptic curves. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 65-79. Springer, Heidelberg (2010)
36. Goundar, R.R., Joye, M., Miyaji, A.: Co-Z addition formulae and binary ladders on elliptic curves. Cryptology ePrint Archive, Report 2010/309 (2010), http://eprint.iacr.org/
37. Goundar, R.R., Joye, M., Miyaji, A., Rivain, M., Venelli, A.: Scalar multiplication on weierstrass elliptic curves from co-Z arithmetic. J. Cryptogr. Eng. 1(2), 161-176 (2011)
38. Guerrini, E., Imbert, L., Winterhalter, T.: Randomizing scalar multiplication using exact covering systems of congruences. Cryptology ePrint Archive, Report 2015/475 (2015), http://eprint.iacr.org/
39. Guerrini, E., Imbert, L., Winterhalter, T.: Randomized mixed-radix scalar multiplication. Cryptology ePrint Archive, Report 2016/1022 (2016), http://eprint.iacr.org/
40. Halak, B., Waizi, S.S., Islam, A.: A survey of hardware implementations of elliptic curve cryptographic systems. Cryptology ePrint Archive, Report 2016/712 (2016), http://eprint.iacr.org/
41. Hamburg, M.: Decaf: eliminating cofactors through point compression. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 705-723. Springer, Heidelberg (2015)
42. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2004)
43. Hanley, N., Kim, H.S., Tunstall, M.: Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. In: Nyberg, K. (ed.) CT-RSA 2015. LNCS, vol. 9048, pp. 429-446. Springer, Heidelberg (2015)
44. Heyszl, J., Mangard, S., Heinz, B., Stumpf, F., Sigl, G.: Localized electromagnetic analysis of cryptographic implementations. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 231-244. Springer, Heidelberg (2012)
45. Hişil, H.: Elliptic Curves, Group Law and Efficient Computation. PhD. thesis, Queensland University of Technology (2010)
46. Hişil, H., Wong, K.K.-H., Carter, G., Dawson, E.: Twisted Edwards curves revisited. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 326-343. Springer, Heidelberg (2008)
47. Homma, N., Miyamoto, A., Aoki, T., Satoh, A., Shamir, A.: Collision-based power analysis of modular exponentiation using chosen-message pairs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol.5154, pp. 15-29. Springer, Heidelberg (2008)

48. Hutter, M., Joye, M., Sierra, Y.: Memory-constrained implementations of elliptic curve cryptography in co-Z coordinate representation. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 170-187. Springer, Heidelberg (2011)
49. Itoh, K., Izu, T., Takenaka, M.: A practical countermeasure against address-bit differential power analysis. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 382-396. Springer, Heidelberg (2003)
50. Izu, T., Möller, B., Takagi, T.: Improved elliptic curve multiplication methods resistant against side channel attacks. In: Menezes, A., Sarkar, P. (eds.) INDOCRYPT 2002. LNCS, vol. 2551, pp. 296-313. Springer, Heidelberg (2002)
51. Izu, T., Takagi, T.: A fast parallel elliptic curve multiplication resistant against side channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 280-296. Springer, Heidelberg (2002)
52. Järvinen, K., Miele, A., Azarderakhsh, R., Longa, P.: FourQ on FPGA: New hardware speed records for elliptic curve cryptography over large prime characteristic fields. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 517-538. Springer, Heidelberg (2016)
53. Joye, M.: Highly regular right-to-left algorithms for scalar multiplication. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 135-147. Springer, Heidelberg (2007)
54. Joye M., Tunstall, M.: Exponent recoding and regular exponentiation algorithms. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 334-349. Springer, Heidelberg (2009)
55. Joye, M., Yen, S.-M.: The Montgomery powering ladder. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 291-302. Springer, Heidelberg (2003)
56. Kim, K.H., Kim, S.I.: A new method for speeding up arithmetic on elliptic curves over binary fields. Cryptology ePrint Archive, Report 2007/181 (2007), http://eprint.iacr.org
57. Kim, K.H., Kim, S.I., Choe, J.S.: New fast algorithms for arithmetic on elliptic curves over finite fields of characteristic three. Cryptology ePrint Archive, Report 2007/179 (2007), http://eprint.iacr.org
58. Kim, K.H., Negre, C.: Point multiplication on supersingular elliptic curves defined over fields of characteristic 2 and 3. SECRYPT 2008. pp. 373–376, INSTICC Press (2008)
59. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104-113. Springer, Heidelberg (1996)
60. Le, D.-P., Tan, C.H., Tunstall, M.: Randomizing the Montgomery powering ladder. Cryptology ePrint Archive, Report 2015/657 (2015), http://eprint.iacr.org/
61. Lochter, M., Merkle, J., Schmidt, J.-M., Schütze, T.: Requirements for standard elliptic curves. Cryptology ePrint Archive, Report 2014/832 (2014), http://eprint.iacr.org/
62. Longa, P.: Accelerating the Scalar Multiplication on Elliptic Curve Cryptosystems over Prime Fields. Masters thesis, School of Information Technology and Engineering, University of Ottawa, Canada (2007)
63. Longa, P.: FourQNEON: Faster elliptic curve scalar multiplications on ARM processors. Cryptology ePrint Archive, Report 2016/645 (2016), http://eprint.iacr.org/

64. Longa, P., Gebotys, C.: Efficient techniques for high-speed elliptic curve cryptography. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 80-94. Springer, Heidelberg (2010)
65. Longa, P., Miri, A.: Fast and flexible elliptic curve point arithmetic over prime fields. IEEE Trans. Comput. 57(3), 289-305 (2008)
66. López, J., Dahab, R.: Improved algorithms for elliptic curve arithmetic in $GF(2^n)$. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 201-212. Springer, Heidelberg (1999)
67. López, J., Dahab, R.: Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 316-327. Springer, Heidelberg (1999)
68. Massolino, P.M.C., Renes, J., Batina, L.: Implementing complete formulas on Weierstrass curves in hardware. Cryptology ePrint Archive, Report 2016/1133 (2016), http://eprint.iacr.org/
69. Meloni, N.: Fast and secure elliptic curve scalar multiplication over prime fields using special addition chains. Cryptology ePrint Archive, Report 2006/216 (2006), http://eprint.iacr.org/
70. Meloni, N.: New point addition formulae for ECC applications. In: Carlet, C., Sunar, B. (eds.) WAIFI 2007. LNCS, vol. 4547, pp. 189-201. Springer, Heidelberg (2007)
71. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comp. 48, 243-264 (1987)
72. Möller, B.: Securing elliptic curve point multiplication against side-channel attacks. In: Davida, G.I., Frankel, Y. (eds.) ISC 2001. LNCS, vol. 2200, pp. 324-334. Springer, Heidelberg (2001)
73. Nascimento, E., Chmielewski, Ł., Oswald, D., Schwabe, P.: Attacking embedded ECC implementations through cmov side channels. Cryptology ePrint Archive, Report 2016/923 (2016), http://eprint.iacr.org/
74. Okeya, K., Sakurai, K.: Fast multi-scalar multiplication methods on elliptic curves with precomputation strategy using Montgomery trick. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, P. (eds.) CHES 2002. LNCS, vol. 2523, pp. 564-578. Springer, Heidelberg (2003)
75. Okeya, K., Takagi, T.: The width-w NAF method provides small memory and fast elliptic scalar multiplications secure against side channel attacks. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 328-343. Springer, Heidelberg (2003)
76. Oliveira, T., López, J., Aranha, D.F., Rodríguez-Henríquez, F.: Lambda coordinates for binary elliptic curves. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 311-330. Springer, Heidelberg (2013)
77. Peng, B.-Y., Hsu, Y.-C., Chueh, D.-C., Cheng, C.-M., Yang, B.-Y.: Multi-core FPGA implementation of ECC with homogeneous co-Z coordinate representation. Cryptology ePrint Archive, Report 2016/909 (2016), http://eprint.iacr.org/
78. Rao, S.R.S.: Three dimensional Montgomery ladder, differential point tripling on Montgomery curves and point quintupling on Weierstrass' and Edwards curves. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2016. LNCS, vol. 9646, pp. 84-106. Springer, Heidelberg (2016)
79. Renes, J., Costello, C., Batina, L.: Complete addition formulas for prime order elliptic curves. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 403-428. Springer, Heidelberg (2016)
80. Rivain, M.: Fast and regular algorithms for scalar multiplication over elliptic curves. Cryptology ePrint Archive, Report 2011/338 (2011), http://eprint.iacr.org/

81. Rondepierre, F.: Revisiting atomic patterns for scalar multiplications on elliptic curves. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 171-186. Springer, Heidelberg (2014)
82. Rondepierre, F.: Revisiting atomic patterns for scalar multiplications on elliptic curves. Cryptology ePrint Archive, Report 2015/408 (2015), http://eprint.iacr.org/
83. Venelli, A., Dassance, F.: Faster side-channel resistant elliptic curve scalar multiplication. In: Kohel, D., Rolland, R. (eds.) Arithmetic, Geometry, Cryptography and Coding Theory 2009. Contemporary Mathematics, vol. 521, pp. 29-40. American Mathematical Society (2010)

Appendix

A. Proof of Theorem 3

Let $P_1 = (x_1, y_1), P_2 = (x_2, y_2), P_1' = (x_1', y_1')$ and $P_2' = (x_2', y_2')$ in the affine point representation, which satisfy (1) with $x_3 = x_1'$, $y_3 = y_1'$, $x_5 = x_2'$ and $y_5 = y_2'$ if $b = 1$, and satisfy (2) with $x_3 = x_2'$, $y_3 = y_2'$, $x_5 = x_1'$ and $y_5 = y_1'$ if $b = 0$.

1) Case $b = 0$ and $b' = 0$:

By the definition of *BPSJ* representation, in this case we have

$$\begin{array}{l} x_1 = \frac{X_1}{S} x_0, y_1 = \frac{K(X_1 - X_2)}{AT} y_0, \\ x_2 = \frac{X_2}{S} x_0, y_2 = \frac{L(X_2 - X_1)}{AT} y_0, \\ K = \frac{A^2}{2(X_1 - X_2)^2}, y_0^2 S^3 = x_0^3 T^2, \end{array} \tag{13}$$

and for the proof of the theorem it is enough to verify validity of following equations:

$$\begin{array}{l} x_1' = \frac{X_1'}{S'} x_0, y_1' = \frac{K'(X_1' - X_2')}{A'T'} y_0, \\ x_2' = \frac{X_2'}{S'} x_0, y_2' = \frac{L'(X_2' - X_1')}{A'T'} y_0, \\ K' = \frac{A'^2}{2(X_1' - X_2')^2}, y_0^2 S'^3 = x_0^3 T'^2 \ . \end{array} \tag{14}$$

First, $K' = \frac{A'^2}{2(X_1' - X_2')^2}$ is obvious from (9). Second, from (9) and the sixth equality of (13), it is checked that $y_0^2 S'^3 = y_0^2 S^3 A^6 = x_0^3 T^2 A^6 = x_0^3 T'^2$.

Then, from the second and the fourth equalities of (13) we get $K + L = (y_1 - y_2)\frac{AT}{y_0(X_1 - X_2)}$, $X_1 = \frac{Sx_1}{x_0}$ and $X_2 = \frac{Sx_2}{x_0}$. Substituting these equalities and regarding the sixth equality of (13) give

$$U = \frac{S^3}{x_0^3} \cdot u = \frac{T^2}{y_0^2} \cdot u, \tag{15}$$

where u is given in (2). Next, from the second, the fourth and the fifth equalities of (13) it follows $\frac{4KL}{SA^2} \cdot x_0 = \frac{-4x_0 T^2 y_1 y_2}{S y_0^2 (X_1 - X_2)^2}$. Substitution of $X_1 = \frac{Sx_1}{x_0}$ and $X_2 = \frac{Sx_2}{x_0}$ to this equality regarding the sixth equality of (13) gives

$$\frac{4KL}{SA^2} \cdot x_0 = \frac{-4y_1 y_2}{(x_1 - x_2)^2} \ . \tag{16}$$

Next, from the second and fifth equalities of (13) it follows $y_1 = \frac{A}{2(X_1-X_2)T}\cdot y_0$ and substituting the first and third equalities of (13) to this equality we have

$$y_1 = \frac{x_0 y_0 A}{2ST(x_1-x_2)}, \tag{17}$$

and regarding $U^2 = \frac{S^3T^2u^2}{x_0^3y_0^2}$ (from (15)) leads to

$$\frac{U^2}{SA^2}\cdot x_0 = \frac{u^2}{4y_1^2(x_2-x_1)^2}\ . \tag{18}$$

- Verification of $x_2' = \frac{X_2'}{S'}\cdot x_0$:

$\frac{X_2'}{S'}\cdot x_0 = \frac{S\cdot A^2+4KL}{S\cdot A^2}\cdot x_0 = x_0 - \frac{4y_1y_2}{(x_2-x_1)^2} = x_2'$ (by (16) and (2)).

- Verification of $x_1' = \frac{X_1'}{S'}\cdot x_0$:

$$\begin{aligned}\tfrac{X_1'}{S'}\cdot x_0 &= \tfrac{X_1'+2X_2'}{S'}\cdot x_0 - 2x_2' \text{ (by using the verified equality } x_2' = \tfrac{X_2'}{S'}\cdot x_0)\\ &= \tfrac{U^2+4KL}{S\cdot A^2}\cdot x_0 - 2x_2' \text{ (by (9))}\\ &= \tfrac{u^2}{4y_1^2(x_2-x_1)^2} - \tfrac{4y_1y_2}{(x_1-x_2)^2} - 2x_2' \text{ (by (16) and (18))}\\ &= x_1' \text{ (by (2)).}\end{aligned}$$

- Verification of $y_2' = \frac{L'(X_2'-X_1')}{A'T'}\cdot y_0$:

$$\begin{aligned}\tfrac{L'(X_2'-X_1')}{A'T'}\cdot y_0 &= \tfrac{(K'-UA')(X_2'-X_1')}{2V(X_2'-X_1')T'}\cdot y_0 = \tfrac{K'-UA'}{2VT'}\cdot y_0 = \tfrac{2V^2-2UV(X_2'-X_1')}{2VT'}\cdot y_0\\ &= \tfrac{V-U(X_2'-X_1')}{T'}\cdot y_0 = \tfrac{U(X_1'-X_2')-U(X_1'-S')-T'}{T'}\cdot y_0\\ &= -y_0 + \tfrac{U(S'-X_2')}{T'}\cdot y_0 = -y_0 + \tfrac{-4KLU}{TA^3}\cdot y_0 \text{ (by (9))}\\ &= -y_0 + \tfrac{-2LTu}{y_0A(X_1-X_2)^2} \text{ (by (15) and the fifth equality of (13))}\\ &= -y_0 + \tfrac{2y_2T^2u}{y_0^2(X_1-X_2)^3} \text{ (by the fourth equality of (13))}\\ &= -y_0 + \tfrac{2y_2u}{(x_1-x_2)^3} \text{ (by the sixth equality of (13))}\\ &= y_2' \text{ (by (2)).}\end{aligned}$$

- Verification of $y_1' = \frac{K'(X_1'-X_2')}{A'T'}\cdot y_0$:

$$\begin{aligned}\tfrac{K'(X_1'-X_2')}{A'T'}y_0 &= \tfrac{(K'-L')(X_1'-X_2')}{A'T'}\cdot y_0 - y_2' \text{ (by } y_2' = \tfrac{L'(X_2'-X_1')}{A'T'}\cdot y_0 \text{ verified above)}\\ &= \tfrac{U(X_1'-X_2')}{T'}\cdot y_0 - y_2' \text{ (by (9))}\\ &= \tfrac{US'(x_1'-x_2')}{x_0T'}\cdot y_0 - y_2' \text{ (by } x_1' = \tfrac{X_1'}{S'}\cdot x_0 \text{ and } x_2' = \tfrac{X_2'}{S'}\cdot x_0 \text{ verified above)}\\ &= \tfrac{uTS(x_1'-x_2')}{y_0x_0A} - y_2' \text{ (by (9) and (15))}\\ &= \tfrac{u(x_1'-x_2')}{2y_1(x_1-x_2)} - y_2' \text{ (by (17))}\\ &= y_1' \text{ (by (2)).}\end{aligned}$$

2) Case $b = 0$ and $b' = 1$:

The difference between (9) and (10) lies only in K-, L-, A-coordinates: $K' = 2(U\cdot(S'-X_1')-T')^2, A' = 2(U\cdot(S'-X_1')-T')\cdot(X_2'-X_1'), L' = K'-U\cdot A'$ for (9); $K'' = 2(U\cdot(S'-X_2')-T')^2, A'' = 2(U\cdot(S'-X_2')-T')\cdot(X_2'-X_1'), L'' = K''+U\cdot A''$ for (10). Here, to distinguish two cases, we replaced K', L', A' of (10) by two-dashed symbols. Hence, in order to show that the points computed by (9) and (10) coincide, according to the definition (Definition 1) of *BPSJ* it is enough to prove equalities $\frac{K''}{A''} = \frac{L'}{A'}$ and $\frac{L''}{A''} = \frac{K'}{A'}$.

- Verification of $\frac{K''}{A''} = \frac{L'}{A'}$:

$\frac{K''}{A''} = \frac{U(S'-X_2')-T'}{X_2'-X_1'} = \frac{U(X_1'-S')+T'}{X_1'-X_2'} - U = \frac{K'}{A'} - U = \frac{L'}{A'}$.

- Verification of $\frac{L''}{A''} = \frac{K'}{A'}$: $\frac{L''}{A''} = \frac{K''}{A''} + U = \frac{L'}{A'} + U = \frac{K'}{A'}$.

3) Case $b = 1$ and $b' = 0$:

By the definition of *BPSJ* representation, in this case we have

$$\begin{array}{l} x_1 = \frac{X_1}{S}x_0, y_1 = \frac{L(X_1-X_2)}{AT}y_0, \\ x_2 = \frac{X_2}{S}x_0, y_2 = \frac{K(X_2-X_1)}{AT}y_0, \\ K = \frac{A^2}{2(X_1-X_2)^2}, y_0^2S^3 = x_0^3T^2, \end{array} \tag{19}$$

and for the proof of the theorem it is enough to again verify validity of (14).

First, $K' = \frac{A'^2}{2(X_1'-X_2')^2}$ and $y_0^2S'^3 = x_0^3T'^2$ follow directly from (11).

Then, by similar ways to derivations of (15) and (16) we can get

$$U = \frac{S^3}{x_0^3}\cdot u = \frac{T^2}{y_0^2}\cdot u, \tag{20}$$

where u is given in (1), and

$$\frac{4KL}{SA^2}\cdot x_0 = \frac{-4y_1y_2}{(x_1-x_2)^2} \ . \tag{21}$$

Next, from the fourth and fifth equalities of (19) it follows $y_2 = \frac{A}{2(X_2-X_1)T}\cdot y_0$ and substituting the first and third equalities of (19) to this we get

$$y_2 = \frac{x_0y_0A}{2ST(x_2-x_1)}, \tag{22}$$

and regarding $U^2 = \frac{S^3T^2u^2}{x_0^3y_0^2}$ which follows from (20) leads to

$$\frac{U^2}{SA^2}\cdot x_0 = \frac{u^2}{4y_2^2(x_2-x_1)^2} \ . \tag{23}$$

- Verification of $x_1' = \frac{X_1'}{S'}\cdot x_0$:

$\frac{X_1'}{S'}\cdot x_0 = \frac{S\cdot A^2+4KL}{S\cdot A^2}\cdot x_0 = x_0 - \frac{4y_1y_2}{(x_2-x_1)^2} = x_1'$ (by (21) and (1)).

- Verification of $x'_2 = \frac{X'_2}{S'} \cdot x_0$:

$$\begin{aligned}\frac{X'_2}{S'} \cdot x_0 &= \frac{X'_2+2X'_1}{S'} \cdot x_0 - 2x'_1 \text{ (by } x'_1 = \frac{X'_1}{S'} \cdot x_0 \text{ verified above)}\\ &= \frac{U^2+4KL}{S \cdot A^2} \cdot x_0 - 2x'_1 \text{ (by (11))}\\ &= \frac{u^2}{4y_2^2(x_2-x_1)^2} - \frac{4y_1y_2}{(x_1-x_2)^2} - 2x'_1 \text{ (by (21) and (23))}\\ &= x'_2 \text{ (by (1)).}\end{aligned}$$

- Verification of $y'_1 = \frac{K'(X'_1-X'_2)}{A'T'} \cdot y_0$:

$$\begin{aligned}\frac{K'(X'_1-X'_2)}{A'T'} \cdot y_0 &= \frac{2V^2(X'_1-X'_2)}{2V(X'_2-X'_1)T'} \cdot y_0 = \frac{-V}{T'} \cdot y_0 = \frac{T'+U(S'-X'_1)}{T'} \cdot y_0\\ &= y_0 + \frac{U(S'-X'_1)}{T'} \cdot y_0 = y_0 + \frac{-4KLU}{TA^3} \cdot y_0 \text{ (by (11))}\\ &= y_0 + \frac{-2LTu}{y_0A(X_1-X_2)^2} \text{ (by (20) and the fifth equality of (19))}\\ &= y_0 + \frac{2y_1T^2u}{y_0^2(X_2-X_1)^3} \text{ (by the second equality of (19))}\\ &= y_0 + \frac{2y_1u}{(x_2-x_1)^3} \text{ (by the sixth equality of (19))}\\ &= y'_1 \text{ (by (1)).}\end{aligned}$$

- Verification of $y'_2 = \frac{L'(X'_2-X'_1)}{A'T'} \cdot y_0$:

$$\begin{aligned}\frac{L'(X'_2-X'_1)}{A'T'} \cdot y_0 &= \frac{(K'-L')(X'_1-X'_2)}{A'T'} \cdot y_0 - y'_1 \text{ (by } y'_1 = \frac{K'(X'_1-X'_2)}{A'T'} \cdot y_0 \text{ verified above)}\\ &= \frac{U(X'_2-X'_1)}{T'} \cdot y_0 - y'_1 \text{ (by (11))}\\ &= \frac{US'(x'_2-x'_1)}{x_0T'} \cdot y_0 - y'_1 \text{ (by } x'_1 = \frac{X'_1}{S'} \cdot x_0 \text{ and } x'_2 = \frac{X'_2}{S'} \cdot x_0 \text{ verified above)}\\ &= \frac{uTS(x'_2-x'_1)}{y_0x_0A} - y'_1 \text{ (by (11) and (20))}\\ &= \frac{u(x'_2-x'_1)}{2y_2(x_2-x_1)} - y'_1 \text{ (by (22))}\\ &= y'_2 \text{ (by (1)).}\end{aligned}$$

4) Case $b = 1$ and $b' = 1$:

The difference between (11) and (12) lies only in K-, L-, A-coordinates: $K' = 2(U \cdot (X'_1 - S') - T')^2, A' = 2(U \cdot (X'_1 - S') - T') \cdot (X'_2 - X'_1), L' = K' + U \cdot A'$ for (11); $K'' = 2(U \cdot (X'_2 - S') - T')^2, A'' = 2(U \cdot (X'_2 - S') - T') \cdot (X'_2 - X'_1), L'' = K'' - U \cdot A''$ for (12). Here, to distinguish two cases, we replaced K', L', A' for (12) by two-dashed symbols. Hence, in order to show that the points computed by (11) and (12) coincide, according to the definition (Definition 1) of *BPSJ* it is enough to prove equalities $\frac{K''}{A''} = \frac{L'}{A'}$ and $\frac{L''}{A''} = \frac{K'}{A'}$.

- Verification of $\frac{K''}{A''} = \frac{L'}{A'}$:

$\frac{K''}{A''} = \frac{U(X'_2-S')-T'}{X'_2-X'_1} = \frac{U(X'_1-S')-T'}{X'_2-X'_1} + U = \frac{K'}{A'} + U = \frac{L'}{A'}$.

- Verification of $\frac{L''}{A''} = \frac{K'}{A'}$: $\frac{L''}{A''} = \frac{K''}{A''} - U = \frac{L'}{A'} - U = \frac{K'}{A'}$. □

B. Randomized Montgomery scalar multiplication in *BPSJ-v1*: $8M + 4S + 12.5A$ per scalar bit by 9 Registers T[0]∼T[8]

```
void randSetup(xx,yy,b,T,swap){s=rand_bit();r=rand_bit();
Mov(T[0],xx);Mov(T[1],yy);T[2]=rand();Sqr(T[3],T[2]);Mul(T[4],T[0],T[3]);
Mul(T[5],T[1],T[3]);Mul(T[1],T[5],T[2]);Sqr(T[7],T[1]);Mul(T[0],T[4],T[7]);
Add(T[0],T[0],T[0]);Add(T[s],T[0],T[0]);Sqr(T[6],T[7]);Add(T[6],T[6],T[6]);
Add(T[6],T[6],T[6]);Add(T[6],T[6],T[6]);Sqr(T[5],T[4]);Add(T[4],T[5],T[5]);
Add(T[4],T[4],T[5]);Sqr(T[5],T[3]);Mul(T[7],T[5],cu_a);Add(T[7],T[4],T[7]);
Sqr(T[1-s],T[7]);Sub(T[1-s],T[1-s],T[s]);Sub(T[1-s],T[1-s],T[s]);
Sub(T[2],T[s],T[1-s]);Mul(T[4],T[7],T[2]);Sub(T[5],T[6],T[4]);
Mul(T[3+r],T[5+(1-b)*r],T[6-4*r]);Mul(T[4-r],T[6-b-(1-b)*r],T[2+4*r]);
Add(T[3],T[3],T[3]);Add(T[3],T[3],T[3]);Add(T[4],T[4],T[4]);
Sqr(T[2+6*r],T[6-b-(1-b)*r]);Sqr(T[8-6*r],T[5+(1-b)*r]);
Add(T[2],T[2],T[2]);Add(T[2],T[2],T[2]);Add(T[6],T[6],T[6]);
Sub(T[7],T[1-s],T[s]);Mov(T[5],T[s]);Sub(T[b^s],T[5],T[b^s]);swap=s;}

void randUpdate(u,v,T,swap){s=swap; t=rand_bit(); r=rand_bit();
Sqr(T[1-u^s],T[7]); Mul(T[7],T[u^s],T[1-u^s]);Add(T[7],T[7],T[7]);
Add(T[7],T[7],T[2]);Mul(T[u^s],T[2],T[3]);Add(T[3],T[3],T[7]);
Mul(T[1-u^s],T[4],T[6]);Sqr(T[2],T[4]);Mul(T[7],T[2],T[5]);
Mov(T[5],T[7]);Mul(T[6],T[1-u^s],T[2]);Add(T[1-u^t],T[u^s],T[5]);
Half(T[7],T[3]);Sqr(T[4],T[7]);Sub(T[4],T[4],T[5]);
Sub(T[u^t],T[4],T[1-u^t]);
Sub(T[4+3*r],T[(5*(1-u)+(v^t)*u)*(1-r)+(1-t)*r],T[((v^t)*(1-u)+5*u)*(1-r)+t*r]);
Sub(T[7-3*r],T[(1-t)*(1-r)+(5*(1-u)+(v^t)*u)*r],T[t*(1-r)+((v^t)*(1-u)+5*u)*r]);
Mul(T[2],T[3],T[4]);Sub(T[8],T[2],T[6]);Mul(T[4],T[8],T[7]);
Sqr(T[2],T[8]);Mul(T[8],T[3],T[4]);Neg(T[3],T[8]);
Sub(T[3*(1-r)+(v^t)*r],T[2+3*r],T[(8-5*(u^v))*(1-r)+(v^t)*r]);
Sub(T[(v^t)*(1-r)+3*r],T[5-3*r],T[(v^t)*(1-r)+(8-5*(u^v))*r]);swap=t;}

void randRecovery(xx,yy,T,swap){ s=swap; Sub(T[1-s],T[5],T[1-s]);
Sub(T[1-s],T[s],T[1-s]);Mul(T[2],T[3],T[1-s]);Mul(T[7],T[4],T[6]);
Mul(T[6],T[7],T[5]);Inv(T[1-s],T[6]);Mul(T[3],T[1-s],T[7]);
Mul(T[7],T[3],T[s]);Mul(T[4],T[5],T[1-s]);Mul(T[0],T[7],xx);
Mul(T[3],T[4],T[2]);Mul(T[1],T[3],yy);}
```

C. Randomized Montgomery scalar multiplication in *BPSJ-v2*: $6M + 6S + 20.5A$ in serial and $4M + 2S + 14.5A$ in parallel per scalar bit by 10 registers T[0]∼T[9]

In a parallel environment equipped with two processors, two operations on a line in the below functions should be executed at the same time.

```
void randSetup(xx,yy,b,T,swap) {s=rand_bit();
r=rand_bit();Mov(T[s],xx); Mov(T[1-s],yy); T[2]=rand();

Sqr(T[3],T[2]);                    Mul(T[4],T[0],T[3]);
Mul(T[5],T[1],T[3]);               Mul(T[1],T[5],T[2]);
Sqr(T[7],T[1]);                    Mul(T[0],T[4],T[7]);
```

```
Add(T[0],T[0],T[0]);                Add(T[s],T[0],T[0]);
Sqr(T[6],T[7]);                     Sqr(T[5],T[4]);
Add(T[6],T[6],T[6]);                Add(T[6],T[6],T[6]);
Add(T[6],T[6],T[6]);                Add(T[4],T[5],T[5]);
Add(T[4],T[4],T[5]);
Sqr(T[5],T[3]);                     Mul(T[7],T[5],cu_a);
Add(T[7],T[4],T[7]);
Sqr(T[1-s],T[7]);
Sub(T[1-s],T[1-s],T[s]);
Sub(T[1-s],T[1-s],T[s]);
Sub(T[2],T[s],T[1-s]);
Mul(T[4],T[7],T[2]);                Sqr(T[7],T[2]);
Sub(T[5],T[6],T[4]);
Mul(T[3+r],T[5+(1-b)*r],T[6-4*r]);  Mul(T[4-r],T[6-b-(1-b)*r],T[2+4*r]);
Add(T[3],T[3],T[3]);                Add(T[4],T[4],T[4]);
Sqr(T[8-6*r],T[4+(2-b)*r]);         Sqr(T[2+6*r],T[6-b-(2-b)*r]);
Add(T[2],T[2],T[2]);                Mov(T[5],T[s]);
Sub(T[b^s],T[5],T[b^s]);
swap=s;}

void randUpdate(u, v, T, swap) { s=swap; t=rand_bit(); r=rand_bit();
m1=(5*(1-u)+(v^t)*u)*(1-r)+(1-t)*r;
n1=((v^t)*(1-u)+5*u)*(1-r)+t*r;
m2=(1-t)*(1-r)+(5*(1-u)+(v^t)*u)*r;
n2=t*(1-r)+((v^t)*(1-u)+5*u)*r;
m3=8*(1-r)+(v^t)*r;
n3=(8-4*(u^v))*(1-r)+(v^t)*r;
m4=(v^t)*(1-r)+8*r;
n4=(v^t)*(1-r)+(8-4*(u^v))*r;
Mul(T[1-u^s],T[2],T[3]);            Mul(T[9],T[4],T[6]);
Add(T[1-u^s],T[1-u^s],T[1-u^s]);    Add(T[3],T[2],T[3]);
Add(T[1-u^s],T[1-u^s],T[1-u^s]);
Mul(T[4],T[u^s],T[7]);              Mul(T[2],T[5],T[8]);
Add(T[1-u^t],T[1-u^s],T[2]);        Add(T[3],T[3],T[4]);
Mul(T[6],T[9],T[8]);                Sqr(T[8],T[3]);
Sub(T[7],T[8],T[2]);                Mov(T[5],T[2]);
Sub(T[u^t],T[7],T[1-u^t]);
Sub(T[7-3*r],T[m1],T[n1]);          Sub(T[4+3*r],T[m2],T[n2]);
Mul(T[2],T[7],T[3]);                Sqr(T[7],T[4]);
Mov(T[9],T[8]);                     Sub(T[8],T[2],T[6]);
Add(T[2],T[8],T[4]);
Sqr(T[4],T[2]);                     Sqr(T[2],T[8]);
Mov(T[8],T[2]);
Add(T[2],T[8],T[8]);                Sub(T[4],T[4],T[8]);
Sub(T[4],T[4],T[7]);
Neg(T[8],T[4]);
Sub(T[m3],T[3+2*r],T[n3]);          Sub(T[m4],T[5-2*r],T[n4]);
Sqr(T[3],T[8]);                     Sqr(T[8],T[4]);
Sub(T[3],T[3],T[9]);
Sub(T[3],T[3],T[8]);
```

```
Half(T[3],T[3]);
Sub(T[3],T[2],T[3]);
swap = t ;}

void randRecovery(xx, yy, T, swap) {s=swap;
Sub(T[1-s],T[5],T[1-s]);            Sub(T[1-s],T[s],T[1-s]);
Mul(T[2],T[3],T[1-s]);              Mul(T[7],T[4],T[6]);
Mul(T[6],T[7],T[5]);
Inv(T[1-s],T[6]);
Mul(T[3],T[1-s],T[7]);              Mul(T[7],T[3],T[s]);
Mul(T[4],T[5],T[1-s]);              Mul(T[0],T[7],xx);
Mul(T[3],T[4],T[2]);                Mul(T[1],T[3],yy);}
```