Randomized Batch Verification of Standard ECDSA Signatures

Sabyasachi Karati, Abhijit Das, and Dipanwita Roychoudhury

Department of Computer Science and Engineering
IIT Kharagpur, India
{skarati,abhij,drc}@cse.iitkgp.ernet.in

Abstract. In AfricaCrypt 2012, several algorithms are proposed for the batch verification of ECDSA signatures. In this paper, we propose three randomization methods for these batch-verification algorithms. Our first proposal is based on Montgomery ladders, and the second on computing square-roots in the underlying field. Both these techniques use numeric arithmetic only. Our third proposal exploits symbolic computations leading to a seminumeric algorithm. We theoretically and experimentally establish that for standard ECDSA signatures, our seminumeric randomization algorithm in tandem with the batch-verification algorithm S2′ gives the best speedup over individual verification. If each ECDSA signature contains an extra bit to uniquely identify the correct y-coordinate of the elliptic-curve point appearing in the signature, then the second numeric randomization algorithm followed by the naive batch-verification algorithm N′ yields the best performance gains. We detail our study for NIST prime and Koblitz curves.

Keywords: ECDSA, Elliptic curve, Koblitz curve, Montgomery ladder, Symbolic computation, Batch verification, Randomization.

1 Introduction

An ECDSA signature on a message M is a triple (M,r,s), where r is the x-coordinate of an elliptic-curve point R, and s is an integer that absorbs the hash of M. Both r and s are reduced modulo the size n of the elliptic-curve group. During verification, two scalars u,v are computed using modulo n arithmetic, and the point R is reconstructed as $R = uP + vQ$, where P is the base point in the elliptic-curve group, and Q is the signer's public key. Verification succeeds if and only if $x(R) = r$.

Suppose that we want to verify a batch of t ECDSA signatures (M_i, r_i, s_i). For the i-th signature, the verification equation is $R_i = u_i P + v_i Q_i$. The t signatures can be combined as

$$\sum_{i=1}^{t} R_i = \left(\sum_{i=1}^{t} u_i\right) P + \left(\sum_{i=1}^{t} v_i Q_i\right). \tag{1}$$

Since the y-coordinates of R_i are not available in the signatures, we cannot straightaway compute the sum on the left side. In AfricaCrypt 2012, several batch-verification algorithms are proposed to solve this problem [1]. The naive algorithms are based upon the

R.S. Chakraborty et al. (Eds.): SPACE 2014, LNCS 8804, pp. 237–255, 2014.

determination of the missing y-coordinate of each R_i using a square-root computation (we have $y_i^2 = r_i^3 + ar_i + b$). The symbolic-manipulation algorithms treat the unknown y-coordinates as symbols. Batch verification involves the eventual elimination of all these y-coordinates from Eqn(1) using the elliptic-curve equation. The symbolic algorithm S2$'$ turns out to be the fastest of the batch-verification algorithms proposed in [1].

In IndoCrypt 2012, Bernstein et al. [2] propose two attacks on these algorithms. They suggest that these attacks can be largely eliminated by randomizing batch verification (a concept introduced by Naccache et al. [3]). For random non-zero multipliers $\xi_1, \xi_2, \ldots, \xi_t$, the verification equations are now combined as

$$\sum_{i=1}^{t} \xi_i R_i = \left(\sum_{i=1}^{t} \xi_i u_i \right) P + \left(\sum_{i=1}^{t} \xi_i v_i Q_i \right). \tag{2}$$

Since the y-coordinates of R_i are not available, Eqn(2) is not directly applicable. In this paper, we propose three efficient ways of randomizing the batch-verification algorithms of [1]. We mostly concentrate on standard ECDSA signatures. If the ECDSA signature contains an extra bit to identify the correct square-root y of $r^3 + ar + b$ [4], we call this an ECDSA# signature. In another variant known as ECDSA* [4,5], the entire point R replaces r in the signature. Neither ECDSA# nor ECDSA* is accepted as a standard. Since ECDSA* results in an unreasonable expansion in the signature size without any increase in the security, we do not consider this variant in this paper. ECDSA#, however, adds only one extra bit to a signature, and so we study the implications of having this extra bit. Our three randomization techniques are based on the following ideas.

- Montgomery ladders: Given only the x-coordinate of an elliptic-curve point R, one can uniquely obtain the x-coordinate of any non-zero multiple ξR [6]. We first compute $x(\xi_i R_i)$ for all signatures in the batch. Then we feed these x-coordinates to the batch-verification algorithms.
- Numeric computation: We explicitly compute y_i from each r_i by taking a square-root of $r_i^3 + ar_i + b$. In ECDSA#, we uniquely obtain R_i from the extra bit. In ECDSA, we have two possibilities $\pm R$. We start with any possibility and numerically compute ξR or $-\xi R$ using standard elliptic-curve doubling and addition formulas.[1]
- Seminumeric computation (Joye's method): We treat each y_i as a symbol, and compute $\xi_i R_i$ as a point in the form $(h_i, k_i y_i)$, where the field elements h_i and k_i are computed from the knowledge of $r_i = x(R_i)$ alone. We precompute the quantity $r_i^3 + ar_i + b$ and follow a slightly modified version of the standard elliptic-curve scalar-multiplication algorithm. Joye in [7] proves that in prime fields the y-coordinate of $\xi_i R_i$ is of the form $(h_i(r_i), k_i(r_i) y_i)$ for functions h_i, k_i of r_i alone. Here, we complement that study by providing explicit computational determination of h_i, k_i, and exploit this procedure to obtain a randomization algorithm that performs better than the above two methods for standard ECDSA signatures. We also derive such explicit formulas for Koblitz curves.

[1] A study of this method is inspired by a comment from an anonymous referee of an earlier version of this paper.

Since the only batch-verification algorithms that deal with standard ECDSA signatures are found in [1], randomizing these algorithms is of practical importance in real-time cryptographic applications like the authentication of messages in vehicular ad hoc networks.

We experiment with the NIST prime family of elliptic curves [8]. Montgomery ladders face a few problems. Each iteration in the scalar-multiplication loop involves one addition and one doubling. More importantly, it is not known how to adapt Montgomery ladders to windowed scalar multiplication.Montgomery's paper [9] proposes some ways of generating short Montgomery chains. As pointed out in [10,11], the creations of the addition chains in these improved variants are rather costly. The practical method of [9] is effective only when the scalar multiplier remains constant, so the addition chain can be precomputed. Since this is not the case with randomizers, we have implemented only the binary ladder. The numeric and the seminumeric randomization algorithms can be adapted to any windowed variant. We theoretically and experimentally establish that the binary Montgomery ladder is slower than the best known windowed variants of the numeric and the seminumeric algorithms. Montgomery arithmetic is efficient for prime curves of the particular form $By^2 = x^3 + Ax^2 + x$. However, the NIST prime curves have large prime orders and cannot be converted to a curve in the Montgomery form which contains the point $(0,0)$ of order two. Point multiplication using Montgomery ladders is more resistant to simple side-channel attacks (SCA) than the numeric and seminumeric methods. In this paper, SCA resistance is not of concern, since verification of signatures uses no private keys.

The rest of this paper is organized as follows. In Section 2, we provide a brief introduction to the ECDSA batch-verification algorithms and the attacks against those. Sections 3 elaborates the three randomization methods introduced above. Section 4 makes a theoretical and experimental comparison of the relative performances of the three methods, and discusses the effects of randomization on the performance of the batch verification of ECDSA and ECDSA# signatures. Section 5 deals with NIST Koblitz curves. Section 6 concludes the paper.

2 Background and Notations

Let (M_i, r_i, s_i), $i = 1,2,\ldots,t$, be a batch of t ECDSA signatures that we want to verify simultaneously. We work over the elliptic curve

$$y^2 = x^3 + ax + b. \tag{3}$$

defined over a large prime field $\mathbb{F}_p$. We assume that the group $E(\mathbb{F}_p)$ is of prime order n with a generator P. For simplicity, we assume that all of the t signatures come from the same signer, that is, $Q_i = Q$ for all i.

2.1 ECDSA Batch Verification

The right side of Eqn(1) can be computed numerically using two scalar multiplications (or one double scalar multiplication). Let this point be (α, β). If R_i are reconstructed as

$u_iP + v_iQ$, the effort is essentially the same as individual verification. The algorithms of [1] solve this problem in many ways.

The naive method N computes y_i by taking the square root of $r_i^3 + ar_i + b$. Since there are two square roots (in general) for each r_i, the ambiguity in the *sign* of y_i can be removed by trying all of the $m = 2^t$ combinations. If Eqn(1) holds for any of these choices, the batch of signatures is accepted. In ECDSA#, the y_i values can be uniquely identified, and we can avoid trying all the $m = 2^t$ combinations. This variant of the naive method is referred to as N$'$. If the underlying field is large, the square-root computations may have huge overheads.

Algorithms S1 and S2 avoid this overhead by computing the left side of Eqn(1) symbolically. Each y_i is treated as a symbol satisfying $y_i^2 = r_i^3 + ar_i + b$. Symbolic addition gives $(g(y_1, y_2, \ldots, y_t), h(y_1, y_2, \ldots, y_t)) = (\alpha, \beta)$, where g and h are polynomials in y_i with each y_i-degree $\leqslant 1$.

Algorithm S1 makes a linearization by repeatedly squaring $g(y_1, y_2, \ldots, y_t) = \alpha$ (or multiplying by even-degree monomials). At this stage too, the equations $y_i^2 = r_i^3 + ar_i + b$ are used to keep the y_i-degrees $\leqslant 1$ in each generated equation. The linearized system has $2^{t-1} - 1 = \frac{m}{2} - 1$ variables corresponding to the square-free monomials in $y_1, y_2, \ldots, y_t$ of even degrees. The system is solved by Gaussian elimination. The equation $h(y_1, y_2, \ldots, y_t) = \beta$ is then used to solve for each y_i. Finally, it is verified whether $y_i^2 = r_i^3 + ar_i + b$ for all i.

Algorithm S2 uses a faster elimination trick. The equation $g(y_1, y_2, \ldots, y_t) = \alpha$ is written as $\gamma(y_2, y_3, \ldots, y_t)y_1 + \delta(y_2, y_3, \ldots, y_t)$. Multiplying this by $\gamma y_1 - \delta$ and using $y_1^2 = r_1^3 + ar_1 + b$ gives an equation free from y_1. The other variables $y_2, y_3, \ldots, y_t$ are eliminated one by one in the same way. Eventually, the batch is accepted if we obtain the zero polynomial after all y_i are eliminated.

An improved variant of S1 and S2 significantly speeds up the symbolic-addition phase. Let $\tau = \lceil t/2 \rceil$. Eqn(1) is rewritten as $\sum_{i=1}^{\tau} R_i = (\alpha, \beta) - \sum_{i=\tau+1}^{t} R_i$. The two sides are individually computed symbolically. These variants of S1 and S2 are referred to as S1$'$ and S2$'$.

2.2 Attacks on ECDSA Batch Verification

In the first attack of Bernstein et al. [2], the batch verifier handles $t - 2$ genuine signatures along with the two forged signatures (r, s) and $(r, -s)$ on the same message M. Since the sum of the elliptic-curve points (r, s) and $(r, -s)$ is $\mathcal{O}$, the entire batch of t signatures is verified as genuine.

In the second attack, the forger knows a valid key pair (d_1, Q_1), and can fool the verifier by a forged signature for any message M_2 under any valid public key Q_2 along with a message M_1 under the public key Q_1. The forger selects a random k_2, computes $R_2 = k_2P$ and $r_2 = x(R_2)$. For another random s_2, the signature on M_2 under Q_2 is presented as (r_2, s_2). For the message M_1, the signature (r_1, s_1) is computed as $R_1 = r_2 s_2^{-1} Q_2$, $r_1 = x(R_1)$, and $s_1 = (e_1 + r_1 d_1)(k_2 - e_2 s_2^{-1})^{-1}$, where $e_1 = H(m_1)$, $e_2 = H(m_2)$, and H is a secure hash function. Now, $R_1 + R_2$ and $(e_1 s_1^{-1} + e_2 s_2^{-1})P + r_1 s_1^{-1} Q_1 + r_2 s_2^{-1} Q_2$ have the same value as $(k_2 P + r_2 s_2^{-1} Q_2)$. These forged signatures are verified if they are in the same batch.

Both these attacks become infeasible by the use of randomizers. If the verifier chooses l-bit randomizers, the security of the batch-verification procedure increases by 2^l. The randomizers need not be of full lengths (of lengths close to that of the prime order p of the relevant elliptic-curve group). As discussed in [12], much smaller randomizers typically suffice to make most attacks on batch-verification schemes infeasible. If the underlying field is of size d bits, then the best known algorithms (the square-root methods) to solve the ECDLP take $\tilde{O}(2^{d/2})$ times. As a result, $d/2$-bit randomizers do not degrade the security of the ECDSA scheme. Another possibility is to take $l = 128$ to get 128-bit security independent of the security guarantees of ECDSA.

3 Randomization of ECDSA Batch Verification

For randomizing ECDSA batch verification as per Eqn(2), the basic problem is to compute the x-coordinate $x(\xi R)$ from $r = x(R)$ and an l-bit scalar $\xi = 1\xi_{l-2}\xi_{l-3}\ldots\xi_1\xi_0$.

3.1 Montgomery Ladders

Montgomery ladders are discussed in [13,14,6]. For the sake of completeness, we present the relevant formulas for point addition and doubling. Suppose that $x(P_1) = h_1$, $x(P_2) = h_2$ and $x(P_1 - P_2) = h_4$ are known to us. We can compute $h_3 = x(P_1 + P_2)$ and $h_5 = x(2P_1)$ by Eqns (4) and (5), respectively.

$$h_3h_4(h_1 - h_2)^2 = (h_1h_2 - a)^2 - 4b(h_1 + h_2). \tag{4}$$

$$4h_5(h_1^3 + ah_1 + b) = (h_1^2 - a)^2 - 8bh_1. \tag{5}$$

The above formulas [13] are adapted from Montgomery's original derivation [6]. Fischer et al. [15] propose a slightly improved addition formula given by

$$(h_4 + h_3)(h_1 - h_2)^2 = 2(h_1 + h_2)(h_1h_2 + a) + 4b.$$

The Montgomery ladder described in Algorithm 1 never uses nor computes the y-coordinate of any point in its repeated double-and-add point-multiplication loop. The loop maintains the invariance $T - S = R$. Since $x(T)$, $x(S)$ and $x(T - S) = x(R)$ are known, we can compute $x(T + S)$, $x(2S)$ and $x(2T)$.

Algorithm 1. Montgomery Ladder for Computing $x(\xi R)$ from ξ and $x(R)$

Initialize $x(S) := x(R)$ and $x(T) := x(2R)$.
For $(i = l-2, l-3, \ldots, 1, 0)$ {
  If $(\xi_i = 0)$, assign $x(T) := x(T + S)$ and $x(S) := x(2S)$;
  else assign $x(S) := x(T + S)$ and $x(T) := x(2T)$;
}
Return $x(S)$.

In many cases, using projective coordinates can speed up the Montgomery-ladder loop. Both the x- and the z-coordinates can be computed from the knowledge of $x(R)$ alone (we assume $z(R) = 1$). Some explicit formulas can be found at [16,17].

Fischer et al. [15] propose an optimization of the Montgomery loop. Irrespective of the bit value ξ_i, the loop computes the x- and z-coordinates of two points $P+Q$ and $2P$, where P is one of the points S,T, and Q is the other point. These operations can be combined together yielding a reduced count of field operations. The problem with Algorithm 1 is that no effective windowed adaptation of it is known (see [10,11]).

3.2 Numeric Computation

We first compute a square-root y of r^3+ar+b. The point R is either (r,y) or $(r,-y)$. An ECDSA# signature has enough information to identify which of these two points is the correct R. An ECDSA signature cannot resolve this ambiguity. However, that is not a serious problem, since both ξR and $-\xi R$ have the same x-coordinate. Therefore, we start with any of the two points $\pm R$, and compute its ξ-th multiple using any standard elliptic-curve scalar-multiplication algorithm. The y-coordinate of this multiple is also computed as a byproduct.

A square root of r^3+ar+b modulo the prime p can be computed by well-known algorithms (like Tonelli-Shanks algorithm). If $p \equiv 3 \pmod 4$, then $(r^3+ar+b)^{(p+1)/4} \pmod p$ is such a square root. Each square-root finding algorithm essentially requires the cost of an exponentiation in the field $\mathbb{F}_p$.

The numeric method has an important bearing on the naive batch-verification methods N and N′ of [1]. These two algorithms start by computing the square roots of $r_i^3+ar_i+b$. If these algorithms are randomized by the numeric method, the y-coordinates of $\xi_i R_i$ are already available (up to sign in ECDSA, and uniquely in ECDSA#), and need not be computed again from the x-coordinates of $\xi_i R_i$. This lets the naive algorithms save significant time. The symbolic batch-verification methods (like S2′) do not use and therefore do not benefit from an explicit knowledge of the y-coordinates.

The numeric method in the context of ECDSA# has another advantage. Since the points R_i are now known uniquely, we can use multiple scalar multiplication. For example, computing $\xi_1 R_1 + \xi_2 R_2$ in a single double-and-add loop needs only l doubling and at most l addition operations, where l is the length of the randomizers. On the contrary, computing $\xi_1 R_1$ and $\xi_2 R_2$ separately by even the best windowed method requires $2l$ doubling operations and some more additions. Thus, the naive batch-verification algorithm N′ derives an additional boost in its performance from multiple scalar multiplication.

3.3 Seminumeric Computation (Joye's Method)

We treat the y-coordinate of $R=(r,y)$ as a symbol satisfying $y^2 = r^3+ar+b$.

Theorem 1: Any non-zero multiple uR of R can be expressed as (h,ky), where h and k are field elements fully determined by (u and) the x-coordinate r of R.

Proof. R itself can be so expressed with $h=r$ and $k=1$. Next, suppose that $P_1 = (h_1,k_1 y)$ and $P_2 = (h_2,k_2 y)$ are two distinct non-zero multiples of R with $P_3 = P_1+P_2 \neq \mathcal{O}$. The addition formula gives $P_3 = (h_3,k_3 y)$, where

$$h_3 = \left(\frac{k_1-k_2}{h_1-h_2}\right)^2 (r^3+ar+b) - h_1 - h_2, \text{ and } k_3 = \left(\frac{k_1-k_2}{h_1-h_2}\right)(h_1-h_3) - k_1.$$

Let $P_4 = 2P_1$. We have $P_4 = (h_4, k_4 y)$, where

$$h_4 = \left(\frac{3h_1^2 + a}{2k_1}\right)^2 \left(\frac{1}{r^3 + ar + b}\right) - 2h_1, \text{ and } k_4 = \left(\frac{3h_1^2 + a}{2k_1}\right)\left(\frac{h_1 - h_4}{r^3 + ar + b}\right) - k_1.$$

Finally, the opposite of (h, ky) is $(h, (-k)y)$. This completes the inductive proof. •

We represent the multiple (h, ky) of R by the pair (h, k) of field elements. The *symbol* y need not be explicitly maintained. R itself is represented by the pair $(r, 1)$. Upon $(r, 1)$ as input, we precompute the quantity $r^3 + ar + b$ and its inverse, and run the standard repeated double-and-add loop of Algorithm 2 with these revised addition and doubling formulas. At the end of the loop, the two computed field elements h, k yield the desired multiple $\xi R = (h, ky)$. In short, we do not need to carry out any symbolic computation at all for obtaining ξR.

Algorithm 2. Seminumeric Computation of $\xi R = (h, ky)$ from ξ and $R = (r, y)$

Precompute the field elements $r^3 + ar + b$ and $(r^3 + ar + b)^{-1}$.
Initialize $S := (r, 1)$.
For $(i = l - 2, l - 3, \ldots, 1, 0)$ {
    Assign $S := 2S$ (use seminumeric doubling formula).
    If $(\xi_i = 1)$, assign $S := S + R$ (use seminumeric addition formula).
}
Return S.

The modified addition formula involves only one extra field multiplication (by the precomputed quantity $r^3 + ar + b$) compared to the standard elliptic-curve addition formula. Point doubling requires two extra field multiplications (each by the precomputed inverse $(r^3 + ar + b)^{-1}$). In Jacobian projective coordinates, we can rearrange the doubling formula to absorb those two extra field multiplications. The standard double-and-add algorithm can be adapted to any windowed variant. Some variants require precomputing multiples uR of R for some small values of u. These multiples are precomputed as pairs of field elements.

The knowledge of the entire points R_1, R_2 allows us to compute $\xi_1 R_1 + \xi_2 R_2$ using a single double-and-add loop, yielding noticeable speedup over two point multiplications. If the y-coordinates of R_1 and R_2 are treated as symbols y_1, y_2, then too $\xi_1 R_1 + \xi_2 R_2$ can be computed seminumerically. Any non-zero point of the form $uR_1 + vR_2$ can be expressed as $(h + jy_1y_2, ky_1 + ly_2)$ for field elements h, j, k, l uniquely determined by the x-coordinates r_1, r_2 (and u, v) alone. Addition and doubling of such points can be rephrased numerically in terms of these field elements. For example, let $P_1 = (h_1 + j_1y_1y_2, k_1y_1 + l_1y_2)$ and $P_2 = (h_2 + j_2y_1y_2, k_2y_1 + l_2y_2)$ be two (distinct) points of the form $uR_1 + vR_2$. In order to compute their sum, we first compute the slope $\lambda = \frac{(k_1 - k_2)y_1 + (l_1 - l_2)y_2}{(h_1 - h_2) + (j_1 - j_2)y_1y_2}$. Using the symbolic-manipulation techniques of [1], we free the denominator of y_1, y_2 (multiply by $(h_1 - h_2) - (j_1 - j_2)y_1y_2$ and substitute $y_i^2 = r_i^3 + ar_i + b$ for $i = 1, 2$). We simplify the numerator too to express λ as $\alpha y_1 + \beta y_2$. Therefore, λ^2 and $x(P_1 + P_2) = \lambda^2 - x(P_1) - x(P_2)$ are of the form $\gamma + \delta y_1y_2$. This process of symbolic computation of $x(P_1 + P_2)$ can be replaced by explicit numeric formulas in $h_1, k_1, j_1, l_1, h_2, k_2, j_2, l_2$. The y-coordinate of

P_1+P_2 and point doubling can be analogously handled. The resulting numeric formulas turn out to be clumsy, and are not expected to benefit the computation of $\xi_1 R_1+\xi_2 R_2$ in a single double-and-add loop. For the weighted sum of three or more points, this idea of seminumeric computation can be extended at least in theory, but chances of getting practical benefits are rather slim.

4 Comparison among the Randomization Algorithms

In this section, we count the field operations in the randomization algorithms. For each of these algorithms, we take the best variant (windowed, if applicable, and with a suitable choice of the coordinate systems) known to us. We then experimentally validate our theoretical observations.

4.1 Comparison of Montgomery Ladders and Seminumeric Method

For the purpose of theoretical comparison, we use standard projective coordinates in the Montgomery-ladder method, and Jacobian projective coordinates in the NAF variant of the seminumeric method. The Montgomery-ladder method in standard projective coordinates produced the best results almost always, whereas the NAF variant of the seminumeric method in Jacobian projective coordinates gave us the best results for curves over large fields. Comparisons among other variants can be analogously carried out.

Let us analyze the Montgomery-ladder implementation first. Let $P_1=(h_1,k_1,\ l_1)$, $P_2=(h_2,k_2,l_2)$, and $P_1-P_2=(r,-y,1)\in E(\mathbb{F}_p)$ be given in projective coordinates. We do not use the y-coordinates k_1,k_2,y. We only compute the x- and z-coordinates of P_1+P_2 and $2P_1$ using the following formulas [15]:

$$x(P_1+P_2)=2(h_1l_2+h_2l_1)(h_1h_2+al_1l_2)+4bl_1^2l_2^2-r(h_1l_2-h_2l_1)^2,$$
$$z(P_1+P_2)=(h_1l_2-h_2l_1)^2,$$
$$x(2P_1)=(h_1^2-al_1^2)^2-8bh_1l_1^3,\text{ and } z(2P_1)=4h_1l_1(h_1^2+al_1^2)+4bl_1^4.$$

If we precompute the field element $-4b$, point addition and point doubling require $M_{Mont}=14M+5S+9A+5(2*)$ field operations (see Table 1 for the notations, and [15] for the derivation of this count). For an l-bit randomizer, the Montgomery-ladder scalar-multiplication does lM_{Mont} operations.

Next, we analyze the seminumeric method. Any non-zero multiple of $(r,y,1)\in E(\mathbb{F}_p)$ is of the form $(\beta_x,\beta_y y,\beta_z)$ with $\beta_x,\beta_y,\beta_z\in\mathbb{F}_p$. Let $P_1=(h_1,k_1y,l_1)$ and $P_2=(h_2,k_2y,l_2)$ be two such multiples, where $P_1\neq\pm P_2$, and y satisfies the equation $y^2=r^3+ar+b$ with r known. We modify the point-addition and doubling formulas of Section 3.3 as given in [18]. These formulas assume that $a=-3$. In particular, the x-, y- and z-coordinates of $P_3=P_1+P_2=(h_3,k_3y,l_3)$ and $P_4=2P_2=(h_4,k_4y,l_4)$ are computed as:

$$H=h_2l_1^2-h_1l_2^2,\ R=k_2l_1^3-k_1l_2^3,\ R'=R^2y^2,\ h_3=R'-H^3-2h_1l_2^2H^2,$$
$$k_3=R(k_1l_2^2-h_3)-k_1l_2^3,\text{ and } l_3=Hl_1l_2.$$

$$H_1=3(h_1-l_1^2)(h_1+l_1^2),\ H_2=H_1^2/y^2,\ R''=4h_1k_1^2,\ h_4=H_2-2R'',$$
$$k_4=H_1(R''-h_4)/y^2-8k_1^4,\text{ and } l_4=2k_1l_1.$$

Table 1. Descriptions of the Symbols

Symbol	Description
M	Finite field Multiplication
S	Finite field Square
I	Finite field Inverse
A	Finite field Addition or subtraction
$(u*)$	Finite field multiplication by the constant element u
M_{Mont}	Montgomery-ladder merged addition-doubling in projective coordinates
A_{Semi}	Seminumeric point addition in the mentioned coordinates
D_{Semi}	Seminumeric point doubling in the mentioned coordinates

We need to perform $A_{Semi} = 13M + 4S + 6A + 1(2*)$ and $D_{Semi} = 6M + 3S + 5A + 1(3*) + 4(2*) + 1(\frac{1}{2}*)$ field operations for point addition and doubling, respectively (with $\frac{1}{2}$, y^2 and y^{-2} precomputed). Each point addition requires only one extra field multiplication than the best implementations mentioned in [16]. Point doubling has the same multiplication count as these best implementations. If we use the w-NAF representation of l-bit randomizers, then there are on an average $\frac{l}{w+1}$ non-zero digits. For each of these non-zero digits, A_{Semi} operations are required. Point doubling (D_{Semi}) is done for each of the l bits. Furthermore, for precomputing 2^{w-2} multiples of $(r, y, 1)$, we need $2^{w-2} - 1$ point additions and one point doubling. Opposites of these multiples take almost zero computation cost.

The seminumeric algorithm is faster than Montgomery-ladder algorithm if:

$$\left(2^{w-2} - 1 + \frac{l}{w+1}\right) A_{Semi} + (l+1) D_{Semi} \leqslant l M_{Mont} \tag{6}$$

Following the convention of [17], we ignore the times required to multiply a field element by a constant (such as 2, 3 or $1/2$) and to add two field elements, since these operations take negligible times compared to field multiplication and squaring. Moreover, as suggested in [16], we take the squaring and multiplication times the same (that is, $1S = 1M$). With these simplifications, Eqn(6) can be rewritten as $17\left(2^{w-2} - 1 + \frac{l}{w+1}\right) + 9(l+1) \leqslant 19l$. Rearrangement of this equation gives $\left(10 - \frac{17}{w+1}\right) l \geqslant 9 + 17(2^{w-2} - 1)$. Putting $w = 4$ in the equation, we get $l \geqslant 9.09$. This theoretically establishes that for $l \geqslant 10$, the seminumeric algorithm is faster than the Montgomery ladder.

It is important to highlight that the worst-case overhead ($A_{Semi} + D_{Semi}$) of an iteration of the seminumeric loop is more than the overhead M_{Mont} of each iteration of the Montgomery-ladder loop. However, the windowed variants of the seminumeric iteration are much more efficient than this worst case, on an average. Montgomery ladders, on the other hand, are unable to take this advantage.

4.2 Comparison of Numeric and Seminumeric Methods

The numeric and seminumeric methods use essentially the same formulas of scalar multiplication. A seminumeric point addition uses one extra field multiplication by the precomputed quantity $r^3 + ar + b$. Seminumeric point doubling requires exactly the same

number of field multiplications as needed by numeric point doubling. The numeric algorithm, on the other hand, has the extra overhead of a square-root computation. As mentioned in Section 3.2, this overhead is essentially that of an exponentiation in $\mathbb{F}_p$. We use an efficient windowed modular exponentiation algorithm. The effect of this overhead on the computation of ξR depends on the bit length of ξ. If ξ is a full-length scalar (that is, of bit length near that of p), then the extra overhead is slightly less than that associated with the extra multiplication in the seminumeric loop. The cryptographically most meaningful length of ξ is about half of that of p. In this case, the square-root computation overhead per bit of the randomizer ξ is doubled, and we expect the seminumeric to be faster than the numeric method.

More precisely, let d be the bit length of q. Each square-root computation by the w-NAF method needs $(1S + (2^{w-2} - 1)M) + (dS + \frac{d}{w+1}M)$ field operations. If we put $1S = 1M$, this is the same as $2^{w-2} + d\left(1 + \frac{1}{w+1}\right)$ multiplications. The w-NAF scalar-multiplication time with an explicitly known y-coordinate and an l-bit scalar is about the same as that of $16\left(2^{w-2} - 1 + \frac{l}{w+1}\right) + 9(l+1)$ field multiplications [16]. Thus, the total overhead of the numeric method is that of

$$2^{w-2} + d\left(1 + \frac{1}{w+1}\right) + 16\left(2^{w-2} - 1 + \frac{l}{w+1}\right) + 9(l+1)$$

multiplications for each point. On the other hand, the seminumeric method with an l-bit scalar needs an equivalent of $17\left(2^{w-2} - 1 + \frac{l}{w+1}\right) + 9(l+1)$ field multiplications for each point, yielding a saving of $\left(d + \frac{d-l}{w+1} + 1\right)$ field multiplications. For $l = \frac{d}{2}$ and $w = 4$, $\left(\frac{11}{10}\right)d + 1$ multiplications are saved.

4.3 Experimental Comparison

The algorithms are implemented in a 2.33 GHz Xeon server running Ubuntu Linux Server Version 2012 LTS. The algorithms are implemented using the GP/PARI calculator [19] (version 2.5.0 compiled by the GNU C compiler 4.6.2). We have used the symbolic-computation facilities of the calculator in our programs. All other functions (like scalar multiplication and square-root computation) are written as subroutines with minimal function-call overheads. Since the algorithms are evaluated in terms of number of field operations, this gives a fair comparison of experimental data with the theoretical estimates. We have implemented windowed, w-NAF and frac-w-NAF methods. We have used affine and projective (standard or Jacobian) coordinates.

The average times of randomization achieved by the Montgomery-ladder and the seminumeric algorithms are listed in Tables 2 and 3 for two NIST prime curves. Here, w is the window size, and l is the bit length of the scalar multiplier (randomizer in the batch-verification application). As mentioned in Section 2.2, we have chosen l to be 128, $d/2$ and d (where $d = |p|$). The seminumeric algorithm is found to be faster than the Montgomery-ladder algorithm, particularly for large randomizers. For NIST prime curves, the experimental speedup is about 25–30%, which is consistent with the theoretical estimates.

Tables 2 and 4 list the overheads associated with the numeric randomization method. In order to compare the performances of the numeric method and the seminumeric

Table 2. Times of Numeric and Seminumeric Scalar Multiplication

Curve	Length of randomizer (in bits)	Numeric Scalar Multiplication (Jacobian Projective Coordinates)		Semiumeric Scalar Multiplication (Jacobian Projective Coordinates)	
		Time (in ms)	w-Algorithm	Time (in ms)	w-Algorithm
P-256	128	2.04	5-NAF-numeric	2.04	4-NAF-seminumeric
	256	3.92	4-NAF-numeric	4.12	4-NAF-seminumeric
P-521	128	2.69	4-NAF-numeric	2.81	4-NAF-seminumeric
	256	5.48	4-NAF-numeric	5.92	4-NAF-seminumeric
	521	10.60	5-NAF-numeric	11.44	5-NAF-seminumeric

Table 3. Times of Montgomery-Ladder and Multiple-Scalar Multiplication

Curve	Length of randomizer (in bits)	Montgomery Ladder * (Standard Projective Coordinates)	Multiple Scalar Multiplication (Jacobian Projective Coordinates)
		Time (in ms)	Time (in ms)
P-256	128	2.72	2.93
	256	5.29	5.67
P-521	128	3.76	4.37
	256	7.45	7.89
	521	15.09	16.30

* No effective windowed variant of Montgommer ladders is known

Table 4. Times for the computation of square roots

Curve	Time (in ms)	Algorithm	w
P-256	0.28	w-numeric	5
P-521	0.76	w-numeric	5

method, we add the best possible numeric scalar multiplication time to the best possible square-root computation time. For full-length randomizers, the best total overheads of the numeric algorithm are $3.92 + 0.28 = 4.20$ and $10.60 + 0.76 = 11.36$ for the two curves P-256 and P-521. In this case, the best overheads incurred by the seminumeric method are very close: 4.12 and 11.44. For half-length randomizers, the best total overheads of the numeric method are $2.04 + 0.28 = 2.32$ and $5.48 + 0.76 = 6.24$ for the two curves. The same overheads for the seminumeric method are slightly better: 2.04 and 5.92. This is the expected pattern as evident from our theoretical estimates. Both the numeric and the seminumeric methods run significantly faster than Montgomery ladders.

In ECDSA#, there is a possibility of using multiple scalar multiplication. Table 3 lists the times for computing the sum $\xi_1 R_1 + \xi_2 R_2$ using a single double-and-add loop. These times are much less than two separate scalar-multiplication times even by the best windowed method.

In an individual verification, one can compute $u_iP + v_iQ$ by a double scalar multiplication. Since P is a fixed base, we can use fixed-base scalar multiplication to compute u_iP. However, Q is not considered to be fixed across different batches (it is assumed to be the same in each batch). So the benefits of double fixed-base scalar multiplication is not clear during individual verification, particularly if the number of signatures with fixed Q is small.

Table 5. Speedup obtained by randomized batch-verification algorithms

Batch-verification algorithm	Randomization algorithm	t	P-256		P-521		
			None*	$l = 128$	None*	$l = 128$	$l = 256$
N	Numeric	3	2.55	1.33	2.60	1.82	1.39
		4	3.15	1.48	3.25	2.12	1.55
		5	3.50	1.55	3.72	2.30	1.65
		6	3.46	1.54	3.85	2.36	1.68
		7	2.93	1.43	3.53	2.23	1.61
		8	2.10	1.20	2.75	1.89	1.43
N′	Numeric	3	2.61	1.48	2.63	1.91	1.53
		4	3.34	1.79	3.37	2.32	1.86
		5	4.01	1.89	4.05	2.58	1.97
		6	4.63	2.11	4.69	2.88	2.20
		7	5.20	2.15	5.28	3.05	2.25
		8	5.73	2.31	5.83	3.27	2.42
S2′	Seminumeric	3	2.98	1.37	2.99	1.97	1.43
		4	3.91	1.54	3.95	2.35	1.62
		5	4.73	1.65	4.87	2.65	1.76
		6	5.41	1.72	5.70	2.87	1.86
		7	5.61	1.74	6.24	3.01	1.91
		8	4.96	1.68	6.08	2.97	1.90

* Without randomization

4.4 Effects of Randomization on Batch-Verification Algorithms

Table 5 illustrates the performance degradation caused by randomization. The speedup figures are computed over individual verification and pertain to the situation where all the signatures come from the same signer. In the table, t is the batch size and l is the bit length of the randomizer. We have taken two cryptographically meaningful values of l (half-length and 128). For original ECDSA signatures, the seminumeric randomization method gives the best performance. For ECDSA#, the extra square-root identifying bits give the points R_i uniquely, so the numeric randomization method is the preferred choice. In each case, the best possible windowed variant is used to compute the speedup. Whenever possible, the best windowed variants are replaced by the faster multiple scalar multiplication method. Table 5 also lists the speedup figures without randomization. Although the increased security provided by randomization incurs reasonable overhead, we still have sizable speedup over individual verification.

5 Adaptation of Randomization Methods to Ordinary Binary Curves

As an illustrative example, we take the family of Koblitz curves recommended by NIST [8]. These curves are defined over binary fields $\mathbb{F}_{2^d}$ by the equation

$$y^2 + xy = x^3 + ax^2 + 1, \text{ with } a = 0 \text{ or } 1. \tag{7}$$

5.1 Montgomery-Ladder Formulas

We now represent elliptic-curve points in the standard projective coordinates. Let $P_1 = (h_1, k_1, l_1)$ and $P_2 = (h_2, k_2, l_2)$ be two non-zero multiples of R. Suppose that $P_1 \neq \pm P_2$ and $P_1 - P_2 = (h_4, k_4, l_4)$. We assume that only the x- and z-coordinates of these points are available. We can compute these coordinates of $P_1 + P_2 = (h_3, k_3, l_3)$ using the following formulas:

$$l_3 = (h_1 k_2)^2 + (h_2 k_1)^2, \qquad x_3 = l_3 h_4 + (h_1 k_2)(h_2 k_1).$$

Compute point doubling $2P_1 = (h_5, k_5, l_5)$ as:

$$h_5 = (h_1^2 + l_1^2)^2, \qquad l_5 = h_1^2 l_1^2.$$

We can easily modify Algorithm 1 to the case of projective coordinates.

5.2 Numeric-Computation Formulas

Now, the relevant problem is the computation of the two values of y from the equation $y^2 + ry + (r^3 + ar^2 + 1) = 0$. We first replace y by ry to convert the equation to the form $y^2 + y + \alpha = 0$, where $\alpha = \frac{r^3 + ar^2 + 1}{r^2}$. The converted equation is solvable if and only if the absolute trace $\mathrm{Tr}(\alpha)$ is zero. In that case, if d is odd, a solution for y is $\alpha^{2^1} + \alpha^{2^3} + \alpha^{2^5} + \cdots + \alpha^{2^{d-2}}$, and the other solution is 1 plus the first solution. These solutions can be efficiently obtained using a half-trace calculation [18].

5.3 Seminumeric-Computation (Joye-Method) Formulas

Let $R = (r, y)$ with y treated as a symbol satisfying the Koblitz-curve equation $y^2 + ry = r^3 + ar^2 + 1$.

Theorem 2: Any non-zero multiple of R can be expressed as $\left(h, k + \left(\frac{y}{r}\right) h\right)$.

Proof First, notice that P itself can be so expressed with $h = r$ and $k = 0$. Next, suppose that $P_1 = (h_1, k_1 + \left(\frac{y}{r}\right) h_1)$ and $P_2 = (h_2, k_2 + \left(\frac{y}{r}\right) h_2)$ are two distinct non-zero multiples of R with $P_3 = P_1 + P_2 \neq \mathcal{O}$. The point-addition formula on Koblitz curves implies that $P_3 = (h_3, k_3 + \left(\frac{y}{r}\right) h_3)$, where

$$\begin{aligned} h_3 &= \left(\frac{k_1 + k_2}{h_1 + h_2}\right)^2 + \left(\frac{k_1 + k_2}{h_1 + h_2}\right) + h_1 + h_2 + a + \left(\frac{r^3 + ar^2 + b}{r^2}\right) \\ &= \frac{h_1(h_2^2 + k_2) + h_2(h_1^2 + k_1)}{h_1^2 + h_2^2}, \\ k_3 &= \left(\frac{k_1 + k_2}{h_1 + h_2}\right)(h_1 + h_3) + h_3 + k_1. \end{aligned}$$

The double P_4 of P_1, if non-zero, can be expressed as $(h_4, k_4 + \left(\frac{y}{r}\right) h_4)$, where:

$$h_4 = h_1^2 + \frac{b}{h_1^2}, \qquad k_4 = h_1^2 + \left(h_1 + \frac{k_1}{h_1} + 1\right) h_3.$$

The opposite of $(h, k + \left(\frac{y}{r}\right) h)$ is $(h, (k+h) + \left(\frac{y}{r}\right) h)$. •

For Koblitz curves, the τ-NAF point-multiplication algorithm is computationally very efficient. This motivates using the following theorem.

Theorem 3: The second-power Frobenius endomorphism on a point of the form $\left(h, k + \left(\frac{y}{r}\right) h\right)$ gives a point in the same form.

Proof Let $P_1 = (h_1, k_1 + \left(\frac{y}{r}\right) h_1)$. Then, $P_5 = (h_1^2, (k_1 + \left(\frac{y}{r}\right) h_1)^2)$ can be expressed as $(h_5, k_5 + \left(\frac{y}{r}\right) h_5)$, where:

$$h_5 = h_1^2, \qquad k_5 = k_1^2 + \left(\frac{r^3 + ar^2 + b}{r^2}\right) h_1^2.$$ •

It follows that for all relevant points of the form $(h, k + \left(\frac{y}{r}\right) h)$, it suffices to store the values of h and k alone. The second term $\left(\frac{y}{r}\right) h$ in the y-coordinate carries no extra information, and does not hamper the arithmetic operations on the points. Indeed, the point negation, doubling, and the second addition formulas are now exactly the same as the numeric formulas for Koblitz curves, without any extra operation. If $a + \frac{r^3+ar^2+b}{r^2} = \frac{r^3+b}{r^2}$ is precomputed, the first formula for computing h_3 does not lead to an increased operation count. Application of the second-power Frobenius endomorphism (computation of k_5), however, now involves a multiplication of h_5 with the precomputed field element $\frac{r^3+ar^2+b}{r^2}$, followed by an addition of this product to k_1^2. After the h and k values of ξR are computed by any addition-chain method, one obtains the point $\xi R = \left(h, k + \left(\frac{h}{r}\right) y\right)$.

5.4 Comparison of Montgomery Ladders and Seminumeric Method

For Koblitz curves, we use standard projective coordinates in the Montgomery-ladder method, and affine coordinates in the *τ-NAF* windowed variant of the seminumeric method. These gave us the best respective running times. In fact, affine coordinates outperformed López-Dahab (LD) coordinates [14] in our implementations.

To analyze the Montgomery-ladder implementation, we take $P_1 = (x_1, y_1, z_1)$, $P_2 = (x_2, y_2, z_2)$, and $P_1 - P_2 = (r, r+y, 1) \in E(\mathbb{F}_{2^d})$ in standard projective coordinates. We only compute the x- and z-coordinates of $P_1 + P_2$ and $2P_1$ according to the formulas given in [14,17]:

$$z(P_1 + P_2) = (x_1 z_2)^2 + (x_2 z_1)^2,\ x(P_1 + P_2) = z(P_1 + P_2) \times r + x_1 x_2 z_1 z_2,$$

$$x(2P_1) = x_1^4 + b z_1^4, \text{ and } z(2P_1) = x_1^2 z_1^2$$

Following the implementation of [18], we need $5M + 5S + 2A$ to compute M_{Mont}.

Now, we analyze the seminumeric algorithm in affine coordinates. All non-zero multiples of $(r,y) \in E(\mathbb{F}_{2^d})$ are of the form $(\beta_x, \beta_y + \frac{\beta_x}{r}y)$ with $\beta_x, \beta_y \in \mathbb{F}_{2^d}$. Let $P_1 = (x_1, y_1 + \frac{x_1}{r}y)$ and $P_2 = (x_2, y_2 + \frac{x_2}{r}y)$ be two such multiples with $P_1 \neq \pm P_2$, and y satisfies the equation $y^2 + ry = r^3 + ar^2 + b$ with r known. The following formulas are derived assuming that $b = 1$ and that $B = (r^3 + ar^2 + b)/r^2$ is precomputed. The x- and y-coordinates of $P_3 = P_1 + P_2 = (x_3, y_3 + \frac{x_3}{r}y)$ and $P_4 = \tau(P_1) = (x_4, y_4 + \frac{x_4}{r}y)$ are computed as follows:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2},\ x_3 = \lambda^2 + \lambda + x_1 + x_2 + B,\ y_x = \lambda(x_1 + x_3) + x_3 + y_1,$$
$$x_5 = x_1^2,\ \text{and}\ y_5 = y_1^2 + Bx_5.$$

$A_{Semi} = 2M + 1S + 1I + 6A$ and $\tau_{Semi} = 1M + 2S$ field operations are needed for each point addition and application of τ, respectively (with B precomputed). Here, point addition does not need any extra multiplication in affine or LD coordinates compared to the formulas given in [20], but the application of τ needs one extra multiplication (in LD coordinates, two extra multiplications are needed). If the addition chain for the scalar multiplier is computed by the τ-*NAF* representation with w-bit windows [17,18,21], then the density of non-zero digits is on an average $\frac{1}{w+1}$. For each of these non-zero digits, A_{Semi} is required, and τ_{Semi} is required for each non-zero digit in the addition chain. In case of τ-*NAF*, we use special τ-chains in the precomputation stage [18,21], where 3-, 4- and 5-bit windows need $\Pi_3 = 1\tau_{Semi} + 1A_{Semi}$, $\Pi_4 = 3\tau_{Semi} + 3A_{Semi}$ and $\Pi_5 = 6\tau_{Semi} + 7A_{Semi}$ curve operations, respectively.

For Koblitz-curve scalar multiplication, we have to pay a special attention to the length of the addition chains. Let c be the co-factor of the Koblitz curve given by Eqn (7). Then, we have $\#E(\mathbb{F}_{2^d}) = cn$. Let $\mu = (-1)^{1-a}$, and $\alpha = (\alpha_1 + \tau\alpha_2) \in \mathbb{Z}[\tau]$. The *norm* of α is given by $N(\alpha) = {\alpha_1}^2 + \mu\alpha_1\alpha_2 + 2{\alpha_2}^2$. The length of the τ-*NAF* representation of α is approximately $\log_2(N(\alpha))$. After the partial modular reduction [21], the length of the addition chain reduces to a maximum of $d + a$. This reduction takes place only if $N(\alpha) \geqslant \frac{4}{7}n$. Therefore, we make our analysis on the basis of whether $N(\alpha) \geqslant \frac{4}{7}n$ or not.

- Case 1: $N(\alpha) < \frac{4}{7}n$
  Let $\alpha = (\alpha_1 + \tau\alpha_2)$ with $\alpha_2 = 0$ be the scalar multiplier, and $l = \log_2 \alpha$ the bit length of the multiplier. The length of the addition chain obtained by the τ-*NAF* representation is approximately $2l$. In contrast, the binary (and NAF) representations produce addition chains of approximate length l. The Montgomery-ladder scalar multiplication needs $C_{Mont} = lM_{Mont} = l(5M + 5S)$ field operations (ignoring additions and subtractions). For w-bit windowed τ-*NAF*, the required operation count in the seminumeric method is $C_{\tau NAF} = \Pi_w + 2l\left(\tau_{Semi} + \frac{1}{w+1}A_{Semi}\right) = \Pi_w + 2l\left((1M + 2S) + \frac{2M+1S+1I}{w+1}\right)$. Our experimental environment shows the relations between M, S and I as $1S = 0.88M$ and $1I = 4.75M$. Using these relations and putting $w = 5$, we simplify the above operation counts as $C_{Mont} = (9.40M)l$, $C_{\tau NAF} = (6.79M)l + (69.97M)$. Therefore, the seminumeric method is faster than Montgomery ladders for $l \geqslant 30$.
- Case 2: $N(\alpha) \geqslant \frac{4}{7}n$

In this case, the length of the τ-NAF addition chain remains nearly $d+a$ (irrespective of the length l of the scalar multiplier), whereas the length of the binary addition chain used by Montgomery ladders increases with l. As l increases, the running time of the seminumeric algorithm remains nearly constant, and the running time of the Montgomery ladder increases linearly with l. Similar operation counts as done in Case 1 now shows that the seminumeric algorithm is faster than Montgomery ladders for $l \geqslant 0.36d + 7.44$. On the other hand, the condition $N(\alpha) \geqslant \frac{4}{7}n$ requires $l \geqslant 0.5d$ approximately. Therefore, the inequality $l \geqslant 0.36d + 7.44$ is always satisfied. We have $\frac{C_{\tau NAF}}{C_{Mont}} \approx 0.36\frac{d}{l}$. For $l = d/2$, the seminumeric algorithms takes about 72% of the running time of Montgomery ladders, and for $l = d$, about 36%.

Table 6. Times of Numeric and Seminumeric Scalar Multiplication

Curve	Length of randomizer (in bits)	Numeric Scalar Multiplication (Affine Coordinates)		Semiumeric Scalar Multiplication (Affine Coordinates)	
		Time (in ms)	w-Algorithm	Time (in ms)	w-Algorithm
K-283	128	196.00	5-τNAF-numeric	265.04	5-τNAF-Seminumeric
	283	216.00	5-τNAF-numeric	286.64	5-τNAF-Seminumeric
K-571	128	516.00	4-τNAF-numeric	718.25	5-τNAF-Seminumeric
	256	968.00	5-τNAF-numeric	1375.88	5-τNAF-Seminumeric
	571	1072.00	5-τNAF-numeric	1530.30	5-τNAF-Seminumeric

Table 7. Times of Montgomery-ladder and Multiple-Scalar Multiplication

Curve	Length of randomizer (in bits)	Montgomery Ladder * (Standard Projective Coordinates) Time (in ms)	Multiple-Scalar Multiplication (Affine Coordinates) Time (in ms)
K-283	128	292.70	417.00
	283	651.62	908.62
K-571	128	824.02	1115.32
	256	1659.40	2233.56
	571	3714.01	4968.33

* No effective windowed variant of Montgommer ladders is known

Table 8. Times for Root Finding by Half-Trace Computation

Curve	Time (in ms)	Algorithm
K-283	8.31	Quarter memory*
K-571	30.84	Quarter memory*

* See [18]

5.5 Experimental Comparison

We have used the same experimental setup as described in Section 4.3. The average times of randomization achieved by the seminumeric and the Montgomery-ladder algorithms

are listed in Tables 6 and 7 for two NIST Koblitz curves. Here, w is the window size, and l is the bit length of the scalar multiplier (randomizer in the batch-verification application). We have chosen l to be 128, $d/2$ and d. The seminumeric algorithm is found to be faster than the Montgomery-ladder algorithm, particularly for large randomizers.

Tables 6 and 8 list the overheads associated with the numeric randomization method. In order to compare the performances of the numeric method and the seminumeric method, we add the best possible numeric scalar multiplication time to the best possible time of root finding using a half-trace computation. Both the numeric and the seminumeric methods run much faster than Montgomery ladders. The seminumeric algorithm is found to be faster than the Montgomery-ladder algorithm, particularly for large randomizers. For Koblitz curves, the speedup is about 10%, 15% and 60% for 128-bit, half-length and full-length randomizers. This pattern is consistent with the theoretical estimates given above.

In ECDSA#, there is a possibility of using multiple scalar multiplication. Table 7 lists the times for computing $\xi_1 R_1 + \xi_2 R_2$ using a single double-and-add loop. This method is much slower than two separate invocations of the best windowed τ-NAF method. In the windowed τ-NAF method, scalar-multiplication times for half-length and full-length scalars are nearly the same (see Section 5.4). Moreover, the τ operation is much more efficient than point doubling. Nevertheless, the randomization overhead being substantial, we do not obtain much speedup from randomized ECDSA batch verification on Koblitz curves (see Table 9). For individual verification, we do not consider fixed-base double exponentiation for the computation of $u_i P + v_i Q$, since Q is not treated as a fixed base across multiple batches.

Table 9. Speedup for NIST Koblitz Curves

Batch-Verification Algorithm	Randomization Algorithm	Batch Size(t)	K-283		k-571		
			None*	$l = 128$	None*	$l = 128$	$l = 256$
N	Numeric	2	1.85	1.00	1.90	1.30	1.02
		3	2.44	1.16	2.64	1.61	1.20
		4	2.55	1.18	3.01	1.75	1.28
		5	2.09	1.07	2.80	1.67	1.24
		6	1.39	0.85	2.10	1.40	1.08
		7	0.79	0.58	1.31	0.99	0.82
N′	Numeric	2	1.90	1.02	1.93	1.32	1.03
		3	2.78	1.23	2.84	1.69	1.24
		4	3.61	1.37	3.72	1.96	1.39
		5	4.39	1.47	4.57	2.18	1.49
		6	5.14	1.54	5.38	2.35	1.57
		7	5.85	1.60	6.17	2.48	1.63
S2′	Seminumeric	2	1.97	0.89	1.98	1.19	0.87
		3	2.89	1.04	2.94	1.48	1.02
		4	3.66	1.13	3.80	1.67	1.11
		5	3.78	1.14	4.25	1.75	1.14
		6	3.48	1.11	4.28	1.76	1.14
		7	2.09	0.93	3.05	1.51	1.03

*Speedup without randomization

6 Conclusion

In this paper, three methods are studied for randomized batch verification of ECDSA signatures. We theoretically and experimentally establish the superiority of the numeric and seminumeric methods over Montgomery ladders. This study is particularly relevant in the context of standard ECDSA signatures.

References

1. Karati, S., Das, A., Roychowdhury, D., Bellur, B., Bhattacharya, D., Iyer, A.: Batch verification of ECDSA signatures. In: Mitrokotsa, A., Vaudenay, S. (eds.) AFRICACRYPT 2012. LNCS, vol. 7374, pp. 1–18. Springer, Heidelberg (2012)
2. Bernstein, D.J., Doumen, J., Lange, T., Oosterwijk, J.-J.: Faster batch forgery identification. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 454–473. Springer, Heidelberg (2012)
3. Naccache, D., M'Raïhi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A. be improved?: Complexity trade-offs with the digital signature standard. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 77–85. Springer, Heidelberg (1995)
4. Antipa, A., Brown, D., Gallant, R., Lambert, R., Struik, R., Vanstone, S.: Accelerated verification of ECDSA signatures. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 307–318. Springer, Heidelberg (2006)
5. Cheon, J.H., Yi, J.H.: Fast batch verification of multiple signatures. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 442–457. Springer, Heidelberg (2007)
6. Montgomery, P.L.: Speeding up Pollard and elliptic curve methods of factorization. In: Mathematics of Computation, vol. 48(177), pp. 243–264 (1987)
7. Joye, M.: Security analysis of RSA-type cryptosystems. Phd thesis, UCL Crypto Group, Belgium (1997)
8. NIST: Recommended elliptic curves for federal government use (1999), `http://csrc.nist.gov/encryption`
9. Montgomery, P.L.: Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains. Microsoft research article, 582 (1992)
10. Stam, M.: On Montgomery-like representations for elliptic curves over $GF(2^k)$. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 240–253. Springer, Heidelberg (2002)
11. Stam, M.: Speeding up subgroup cryptosystems. PhD thesis, Technische Universiteit Eindhoven (2003)
12. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 236–250. Springer, Heidelberg (1998)
13. Brier, E., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 335–345. Springer, Heidelberg (2002)
14. 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)
15. Fischer, W., Giraud, C., Knudsen, E.W., Seifert, J.P.: Parallel scalar multiplication on general elliptic curves over F_p hedged against non-differential side-channel attacks. IACR Cryptology ePrint Archive 2002/007 (2002)
16. Bernstein, D.J., Lange, T.: Explicit-Formulas Database (2007), `http://www.hyperelliptic.org/EFD/index.html`

17. Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography, 2nd edn. Chapman & Hall/CRC (2012)
18. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer-Verlag New York, Inc., Secaucus (2003)
19. PARI Group: PARI/GP home (2008), http://pari.math.u-bordeaux.fr/
20. Lange, T.: A note on López-Dahab coordinates. IACR Cryptology ePrint Archive 2004/323 (2004)
21. Solinas, J.A.: Improved algorithms for arithmetic on anomalous binary curves. Technical report, Originally presented in Advances in Cryptography, Crypto 1997 (1997)