Faster Batch Verification of Standard ECDSA Signatures Using Summation Polynomials

Sabyasachi Karati and Abhijit Das

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, India
{skarati,abhij}@cse.iitkgp.ernet.in

Abstract. Several batch-verification algorithms for original ECDSA signatures are proposed for the first time in AfricaCrypt 2012. Two of these algorithms are based on the naive idea of taking square roots in the underlying fields, and the others perform symbolic manipulation to verify small batches of ECDSA signatures. In this paper, we use elliptic-curve summation polynomials to design a new ECDSA batch-verification algorithm which is theoretically and experimentally much faster than the symbolic algorithms of AfricaCrypt 2012. Our experiments on NIST prime and Koblitz curves demonstrate that our proposed algorithm increases the optimal batch size from seven to nine. We also mention how our algorithm can be adapted to Edwards curves.

Keywords: Elliptic Curve, ECDSA, Batch Verification, Summation Polynomial, Koblitz Curve, Edwards Curve, EdDSA.

1 Introduction

When multiple signatures sharing common system parameters need to be verified, the concept of batch verification turns out to be useful. The basic incentive is a reduction in the running time of individually verifying the signatures. The elliptic-curve digital signature algorithm (ECDSA) [12] has been accepted as a standard signature scheme. An ECDSA signature on a message M is a pair (r, s), where r is x-coordinate of an elliptic-curve point $R = (r, y)$, and s absorbs the hash of M and the private key of the signer. The absence of the y-coordinate of the point R in an ECDSA signature resists a straightforward adaptation of the previously proposed batch-verification methods [16,11]. There exist two y-coordinates corresponding to the x-coordinate r. This results in an ambiguity in identifying the correct y-coordinate and leads to a sizable overhead for eliminating the y-coordinate from the batch-verification equation. In a variant ECDSA* [7], the entire point R replaces r in the signature. As a result, batch verification of ECDSA* signatures is straightforward. However, since ECDSA* is not standardized and leads to an expansion in the signature size without any increase in security, batch verification of standard ECDSA signatures continues to remain a problem of both theoretical and practical importance in cryptography.

I. Boureanu, P. Owesarski, and S. Vaudenay (Eds.): ACNS 2014, LNCS 8479, pp. 438–456, 2014.

Karati et al. [13] propose several batch-verification algorithms for standard ECDSA signatures. Their naive algorithm N is based upon the computation of the y-coordinate by taking a square root in the underlying field. The algorithms S1 and S2 of [13] trades the square-root computation time by symbolic manipulations that treat the y-coordinates as symbols satisfying the elliptic-curve equation. Algorithm S1 performs linearization during the elimination of the unknown y-coordinates. Algorithms S2 adopts a separate and more efficient elimination method. Both S1 and S2 outperform the naive method N for small batch sizes. For a batch of size t, S1 runs is $\mathrm{O}(m^3)$ time and S2 runs in $\mathrm{O}(mt^2)$ time, where $m = 2^t$, and the running times are measured in the number of field operations. Since m is already an exponential function in t, these algorithms become impractical except only for small batch sizes. Reducing the running time to below $\mathrm{O}(mt^2)$ is stated as an open problem in [13].

In this paper, we address this open problem. We propose a new batch-verification algorithm (we call it SP) which is theoretically more efficient and experimentally faster than the symbolic-computation algorithms of [13]. Our proposed algorithm uses a separate elimination technique which is based on Semaev's elliptic-curve summation polynomials [21]. Algorithm SP has a running-time complexity of $\mathrm{O}(m)$ and so is theoretically superior than the earlier symbolic algorithms. Practically, Algorithm SP can handle batches of size up to ten, whereas the earlier symbolic algorithms are effective for batch sizes $t \leqslant 8$ only. We show that Algorithm SP (like S1 and S2) supplies security guarantees equivalent to the standard batch-verification algorithm for ECDSA* [7].

Algorithms S1 and S2 proceed in two phases. In the first phase, a sum of the elliptic-curve points (r_i, y_i) is computed. In this phase, r_i are known and y_i are treated as symbols. The second phase eliminates all y_i values using more symbolic manipulations. The elimination phase effectively determines the running times of S1 and S2 as $\mathrm{O}(m^3)$ and $\mathrm{O}(mt^2)$, respectively. Algorithm SP, on the contrary, completely avoids the symbolic addition phase, and manages the elimination of all y_i values in $\mathrm{O}(m)$ time only.

The rest of the paper is organized as follows. Section 2 introduces the notations and a quick overview of the ECDSA scheme and the batch-verification algorithms of [13]. In section 3, we propose the new Algorithm SP. Section 4 contains the complexity analysis and the security analysis of Algorithm SP. NIST Koblitz curves are dealt with in Section 5. We provide our experimental results for NIST prime and Koblitz curves in Section 6. Section 7 deals with the adaptation of Algorithm SP to Edwards curves. The security of Algorithm SP depends on the structures of the elliptic-curve groups over quadratic extensions of the base fields. These structures are studied in Section 8 for some of the NIST curves. Section 9 concludes the paper after highlighting some pertinent open problems.

2 Notations and Background

In the rest of this paper, we plan to verify a batch of t ECDSA signatures (M_1, r_1, s_1), (M_2, r_2, s_2), $\ldots$, (M_t, r_t, s_t).

2.1 ECDSA over NIST Prime Fields

Let

$$E : y^2 = x^3 + ax + b \tag{1}$$

be an elliptic curve defined over the prime field $\mathbb{F}_p$. The size of the group $E(\mathbb{F}_p)$ is assumed to be a prime n close to p. Let P be a fixed generator of $E(\mathbb{F}_p)$.

An ECDSA private key d is randomly chosen from $\{1, 2, \ldots, n-1\}$. The public key is computed as $Q = dP$.

The ECDSA signature (r, s) on a message M is generated as follows. A random session key $k \in \{1, 2, \ldots, n-1\}$ is selected. The point $R = kP$ is computed, and r is taken as the x-coordinate $x(R)$ of R reduced modulo n. Finally, s is computed as $s = k^{-1}(H(M) + dr) \pmod{n}$, where H is a cryptographic hash function like SHA-1 [18].

By Hasse's theorem, we have $|n - p - 1| \leqslant 2\sqrt{p}$. If $n \geqslant p$, then r as an element of $\mathbb{Z}_n$ has a unique representation in $\mathbb{Z}_p$, otherwise it has two representations. The density of elements of $\mathbb{Z}_n$ having two representations in $\mathbb{Z}_p$ is $\leqslant 2/\sqrt{p}$ which is close to zero if p is large. Consequently, we ignore the cases where the modulo n and the modulo p values of r may be different.

To verify an ECDSA signature (M, r, s), we compute $w = s^{-1} \pmod{n}$, $u = H(M)w \pmod{n}$ and $v = rw \pmod{n}$. The point R is reconstructed as

$$R = uP + vQ. \tag{2}$$

The signature is accepted if and only if $x(R) = r \pmod{n}$.

As mentioned before, an ECDSA* signature on M is the pair (R, s). Verification proceeds as in the case of ECDSA signatures, and the validity of Eqn(2) is used as the acceptance criterion.

For a given x-coordinate r, there are in general two y-coordinates $\pm y$. The point R is one of (r, y) and $(r, -y)$. In another variant of ECDSA, henceforth referred to as ECDSA#, an extra bit is appended to a standard ECDSA signature in order to identify which of $(r, \pm y)$ is equal to R. Unlike ECDSA*, ECDSA#—although not accepted as a standard—does not suffer from an unacceptable expansion in the signature size. Since ECDSA# has important bearings on the naive batch-verification algorithm of [13], we refer to it in Section 6. ECDSA* is considered only in the security proof of Algorithm SP.

2.2 Batch Verification of ECDSA Signatures

We assume that all of the t signatures (M_i, r_i, s_i) come from the same signer with public key Q (an adaptation to the case of multiple signers being straightforward). A batch-verification attempt aggregates the t signatures as

$$\sum_{i=1}^{t} R_i = \left(\sum_{i=1}^{t} u_i\right) P + \left(\sum_{i=1}^{t} v_i\right) Q. \tag{3}$$

The right side of Eqn(3) 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 [13] get around this difficulty in several 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(3) holds for any of these choices, the batch of signatures is accepted. If we use ECDSA#, then the y_i values can be uniquely identified, and we can avoid trying all the $m = 2^t$ combinations. This variant of the naive algorithm is referred to as N′. The computation of the square roots of $r_i^3 + ar_i + b$ cannot be avoided in Algorithms N and N′. If the underlying field is large, this overhead may be huge.

The symbolic-manipulation algorithms S1 and S2 avoid computing these square roots altogether. They instead compute the left side of Eqn(3) symbolically. Each y_i is treated as a symbol satisfying $y_i^2 = r_i^3 + ar_i + b$. This symbolic addition gives an equality of the form

$$(g(y_1, y_2, \ldots, y_t), h(y_1, y_2, \ldots, y_t)) = (\alpha, \beta), \tag{4}$$

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 in order 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 standing for the square-free monomials in $y_1, y_2, \ldots, y_t$ of even degrees. The linearized system is in general dense, and is solved by Gaussian elimination in $\mathrm{O}(m^3)$ time. The equation $h(y_1, y_2, \ldots, y_t) = \beta$ is subsequently 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 avoids the massive $\mathrm{O}(m^3)$ overhead of Gaussian elimination as follows. The equation $g(y_1, y_2, \ldots, y_t) = \alpha$ is rewritten 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. This elimination phase takes $\mathrm{O}(mt^2)$ time.

An improved variant of S1 and S2 significantly speeds up the symbolic-addition phase. Let $\tau = \lceil t/2 \rceil$. Eqn(3) 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. This reduces the running time of the symbolic-addition phase from $\mathrm{O}(mt^2)$ to $\mathrm{O}(\sqrt{m}t^2)$. These variants of S1 and S2 are referred to as S1′ and S2′. The elimination phases of S1′ and S2′ run in $\mathrm{O}(m^{3/2})$ and $\mathrm{O}(mt^2)$ times, respectively.

2.3 Randomization of Batch Verification

Bernstein et al. [2] propose two attacks on these batch-verification algorithms. They also suggest that these attacks can be largely eliminated by randomizing the batch-verification process (see [1,16]). For randomly chosen non-zero multipliers $\xi_1, \xi_2, \ldots, \xi_t$, the individual verification equations are now combined as

$$\sum_{i=1}^{t} \xi_i R_i = \left(\sum_{i=1}^{t} \xi_i u_i \right) P + \left(\sum_{i=1}^{t} \xi_i v_i \right) Q. \tag{5}$$

The right side can again be computed numerically. The x-coordinates of $\xi_i R_i$ can be computed from $x(R_i)$ [15,14], and are supplied as inputs to the batch-verification algorithms. The randomization process is external to batch verification. However, individual verification does not require randomization. Although randomized batch verification is the cryptographically meaningful implementation of the algorithms, we also study batch verification without randomization in order to compare the raw performances of various batch-verification algorithms.

It is worthwhile to note that Eqns(3) and (5) can be readily modified to the case when the t signatures come from different signers having different public keys Q_i. For example, the sum $\left(\sum_{i=1}^{t} \xi_i v_i\right) Q$ in Eqn(5) should be replaced by $\sum_{i=1}^{t} (\xi_i v_i Q_i)$. But then, the number of scalar multiplications increases from two to $t+1$. Since randomization incurs additional overheads similar to several scalar multiplications, randomized batch verification using the algorithms S2′ or SP is expected to be slower than individual verification. Consequently, we do not study the case of multiple signers in this paper.

3 A New Batch-Verification Algorithm (SP) for ECDSA

The new batch-verification algorithm we propose in this paper is based on elliptic-curve summation polynomials introduced by Semaev [21] in the context of improving the known bounds of the index-calculus method for solving the elliptic-curve discrete-logarithm problem.

Let E be the elliptic curve defined over a prime field $\mathbb{F}_p$ by Eqn(1). Let $x_1, x_2, \ldots, x_t$ be $t \geqslant 2$ elements of $\mathbb{F}_p$. The t-variable summation polynomial f_t is defined by the following recurrences:

$$f_2(x_1, x_2) = x_1 - x_2, \tag{6}$$

$$f_3(x_1, x_2, x_3) = (x_1 - x_2)^2 {x_3}^2 - 2((x_1 + x_2)(x_1 x_2 + a) + 2b)x_3 + ((x_1 x_2 - a)^2 - 4b(x_1 + x_2)), \tag{7}$$

$$f_t(x_1, x_2, \ldots, x_t) = \mathrm{Res}_X(f_{t-k}(x_1, \ldots, x_{t-k-1}, X), f_{k+2}(x_{t-k}, \ldots, x_t, X)) \quad \text{for } t \geqslant 4 \text{ and for any } k \text{ in the range } 1 \leqslant k \leqslant t-3. \tag{8}$$

Here, Res_X stands for the resultant of two polynomials with respect to the variable X. Semaev proves that $f_t(x_1, x_2, \ldots, x_t) = 0$ if and only if there exist $y_1, y_2, \ldots, y_t \in \overline{\mathbb{F}}_p$ with (x_i, y_i) satisfying Eqn(1) for all $i = 1, 2, \ldots, t$ such that we have the following sum in the elliptic-curve group $E(\overline{\mathbb{F}}_p)$:

$$(x_1, y_1) + (x_2, y_2) + \cdots + (x_t, y_t) = \mathcal{O}, \tag{9}$$

where $\mathcal{O}$ is the point at infinity on E, and $\overline{\mathbb{F}}_p$ is the algebraic closure of $\mathbb{F}_p$.

For the batch verification of t ECDSA signatures (M_i, r_i, s_i), we first compute the numeric sum R on the right side of Eqn(5). In the non-randomized case of Eqn(3), we have $\xi_1 = \xi_2 = \cdots = \xi_t = 1$. Let $R = (\alpha, \beta)$, where $\alpha, \beta \in \mathbb{F}_p$. Let $\xi_i(r_i, y_i) = (r_i', y_i')$. Eqn(5) can be rewritten as

$$(r_1', y_1') + (r_2', y_2') + \cdots + (r_t', y_t') + (\alpha, -\beta) = \mathcal{O}. \tag{10}$$

By Eqn(9), this is equivalent to the condition $f_{t+1}(r'_1, r'_2, \ldots, r'_t, \alpha) = 0$. Algorithm 1 incorporates this idea to verify a batch of t ECDSA signatures.

Algorithm 1. ECDSA Batch-verification Algorithm SP for NIST Prime Curves

INPUT: Domain Parameters, ECDSA signatures (M_1, r_1, s_1), (M_2, r_2, s_2), $\ldots$, (M_t, r_t, s_t) and public keys $Q_1, Q_2, \ldots, Q_t$ of the signers.
OUTPUT: Accept/Reject the batch of t signatures.

1. Optional sanity check: For each $i = 1, 2, \ldots, t$, check whether $r_i^3 + ar_i + b$ is a quadratic residue modulo p. If not, reject the i-th signature and remove it from the batch. Let us assume that all the signatures in the batch pass the sanity check. (Also see Section 4.5.)
2. Compute $w_i = s_i^{-1} \pmod n$ for all $i = 1, 2, \ldots, t$.
3. Compute $u_i = H(M_i)w_i \pmod n$ for all $i = 1, 2, \ldots, t$.
4. Compute $v_i = r_i w_i \pmod n$ for all $i = 1, 2, \ldots, t$.
5. Choose t random integers $\xi_1, \xi_2, \ldots, \xi_t \in \{1, 2, \ldots, n-1\}$, where n is the order of the base point of the elliptic curve. For the non-randomized version, we take $\xi_1 = \xi_2 = \cdots = \xi_t = 1$.
6. Compute $R = (\sum_{i=1}^{t} \xi_i u_i)P + (\sum_{i=1}^{t} \xi_i v_i)Q = (\alpha, \beta)$.
7. For the randomized version, compute $r'_i = x(\xi_i(r_i, y_i))$ using Montgomery ladders or seminumeric scalar multiplication of [14,15]. For the non-randomized version, take $r'_i = r_i$.
8. Compute the value of the summation polynomial $\phi = f_{t+1}(r'_1, r'_2, \ldots, r'_t, \alpha)$.
9. Accept the batch of signatures if and only if $\phi = 0$.

4 Analysis of Algorithm SP

4.1 Properties of Summation Polynomials

For $t = 2$ or 3, we straightaway use the formulas given in Eqn(6) or (7). For $t \geqslant 4$, we make two recursive calls as given in Eqn(8). In order to optimize efficiency, the number of variables in each recursive call should be about $t/2$. More precisely, we always choose $k = \lceil t/2 \rceil$ (this is in the allowed range of values of k), so the first recursive call computes $f_{\lceil t/2 \rceil + 1}$ and the second recursive call computes $f_{\lfloor t/2 \rfloor + 1}$. The leaves of the recursion tree deal with the base cases of Eqns(6) and (7).

Theorem 1: Let $t = 2^h + 2$ for some $h \geqslant 0$. Then, the recursion tree for the computation of f_t is a complete binary tree of height h, and all the leaves correspond to the computation of f_3 by Eqn(7).

Proof We proceed by induction on h. For $h = 0$, we compute f_3 straightaway from Eqn(7) without making any recursive call, that is, the recursion tree is of height zero. For $h \geqslant 1$, suppose that the assertion holds for the computation of $f_{2^{h-1}+1}$. The computation of f_t proceeds as

$$f_t(x_1, x_2, \ldots, x_t) = \mathrm{Res}_X(f_{\frac{t}{2}+1}(x_1, \ldots, x_{\frac{t}{2}}, X), f_{\frac{t}{2}+1}(x_{\frac{t}{2}+1}, \ldots, x_t, X)).$$

Here, t is even, so $\lceil t/2 \rceil = \lfloor t/2 \rfloor = t/2$. Moreover, $t/2 = 2^{h-1} + 1$, so by the induction hypothesis, the sub-trees for the two recursive calls are complete binary trees with each leaf computing f_3. •

In general, let h be the height of the recursion tree for the computation of f_t. By Theorem 1, we have $2^{h-1} + 2 < t \leqslant 2^h + 2$, that is, $\log_2(t-2) \leqslant h < 1 + \log_2(t-2)$, that is, the height of the recursion tree is $\Theta(\log t)$.

Theorem 2: Let $t = 2^h + 2$ with $h \geqslant 1$. If we compute f_t recursively as

$$f_t(x_1, x_2, \ldots, x_t) = \mathrm{Res}_X\left(f_{\frac{t}{2}+1}\left(x_1, \ldots, x_{\frac{t}{2}}, X\right), f_{\frac{t}{2}+1}\left(x_{\frac{t}{2}+1}, \ldots, x_t, X\right)\right), \quad (11)$$

then we take the resultant of two polynomials in X of degrees equal to $2^{\left(\frac{t-2}{2}\right)}$.

Proof. We first supply a direct proof based upon induction on h. For the base case $h = 1$, we have four elements x_1, x_2, x_3, x_4. We compute $f_4(x_1, x_2, x_3, x_4) = \mathrm{Res}_X(f_3(x_1, x_2, X), f_3(x_3, x_4, X))$. By Eqn(7), the X-degree of each of the two arguments of Res_X is $2 = 2^{\left(\frac{4-2}{2}\right)}$.

Now, let $h \geqslant 2$ and $t' = \frac{t}{2} + 1 = 2^{h-1} + 2$. We have $f_{t'}(x_1, x_2, \ldots, x_{t'-1}, X) = \mathrm{Res}_Y\left(f_{\frac{t'}{2}+1}\left(x_1, \ldots, x_{\frac{t'}{2}}, Y\right), f_{\frac{t'}{2}+1}\left(x_{\frac{t'}{2}+1}, \ldots, x_{t'-1}, X, Y\right)\right)$. We inductively assume that this computation of $f_{t'}$ involves the resultant calculation of two polynomials of Y-degree $\delta = 2^{\left(\frac{t'-2}{2}\right)}$ each. But each summation polynomial is symmetric about its arguments. Therefore, the X-degree of the second argument is again $\delta = 2^{\left(\frac{t'-2}{2}\right)}$. Let us write

$$f_{\frac{t'}{2}+1}\left(x_1, \ldots, x_{\frac{t'}{2}}, Y\right) = a_\delta Y^\delta + a_{\delta-1} Y^{\delta-1} + \cdots + a_0, \quad (12)$$

$$f_{\frac{t'}{2}+1}\left(x_{\frac{t'}{2}+1}, \ldots, x_{t'-1}, X, Y\right) = b_\delta Y^\delta + b_{\delta-1} Y^{\delta-1} + \cdots + b_0. \quad (13)$$

Here, the coefficients a_i do not involve X, whereas the coefficients b_i are polynomials in X. Since the X-degree of the second polynomial is δ, and the polynomial is symmetric about X and Y, we conclude that the X-degree of b_δ is δ. The X-degrees of the other coefficients b_i are $\leqslant \delta$.

The $(2\delta) \times (2\delta)$ Sylvester matrix of the polynomials in Eqns(12) and (13) is

$$S = \begin{pmatrix} a_\delta & a_{\delta-1} & \cdots & a_0 & 0 & \cdots & 0 \\ 0 & a_\delta & a_{\delta-1} & \cdots & a_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & a_\delta & a_{\delta-1} & \cdots & a_0 \\ b_\delta & b_{\delta-1} & \cdots & b_0 & 0 & \cdots & 0 \\ 0 & b_\delta & b_{\delta-1} & \cdots & b_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & b_\delta & b_{\delta-1} & \cdots & b_0 \end{pmatrix}$$

The X-degree of $\det S$ is the X-degree of b_δ^δ, that is, $\delta^2 = 2^{t'-2} = 2^{\left(\frac{t-2}{2}\right)}$. Similarly, the X-degree of $f_{\frac{t}{2}+1}\left(x_{\frac{t}{2}+1}, \ldots, x_t, X\right)$ is again $2^{\left(\frac{t-2}{2}\right)}$. •

Alternative proof In [21], Semaev proves that the summation polynomial f_k, $k \geqslant 3$, is of the form

$$f_k(x_1, x_2, \ldots, x_k) = f_{k-1}^2(x_1, x_2, \ldots, x_{k-1}) x_k^{2^{k-2}} + \cdots . \tag{14}$$

Now, we put $k = \frac{t}{2} + 1$ and $x_k = X$, and rewrite Eqn(14) as

$$f_{\frac{t}{2}+1}\left(x_1, \ldots, x_{\frac{t}{2}}, X\right) = f_{\frac{t}{2}}^2\left(x_1, x_2, \ldots, x_{\frac{t}{2}}\right) X^{2^{\left(\frac{t-2}{2}\right)}} + \cdots ,$$

that is, $f_{\frac{t}{2}+1}\left(x_1, \ldots, x_{\frac{t}{2}}, X\right)$ is a polynomial of degree $2^{\left(\frac{t-2}{2}\right)}$ in X. Likewise, $f_{\frac{t}{2}+1}\left(x_{\frac{t}{2}+1}, \ldots, x_n, X\right)$ too is a polynomial of degree $2^{\left(\frac{t-2}{2}\right)}$ in X. •

Theorem 2 can be generalized to any value of t (that is, values not only of the form $2^h + 2$). However, the resulting formulas involve many floor and ceiling expressions. For the sake of simplicity, we restrict only to the special case which already portrays the performance of SP as a function of t.

4.2 A Strategy to Handle the Variables in the Recursion Tree

Let r_i denote the known x-coordinates, and X_j the variables used in the recursion of Eqn(8). For achieving good performance, we reduce the number of variables in each node of the recursion tree. Each child of the root has one variable. Now, let some node compute the summation polynomial of $r_i, r_{i+1}, \ldots, r_{i+k-1}, X_j$ (the case of one variable). Its two child nodes compute the summation polynomials of $r_i, r_{i+1}, \ldots, r_{i+\lceil k/2 \rceil - 1}, X_{j'}$ and $r_{i+\lceil k/2 \rceil}, \ldots, r_{i+k-1}, X_j, X_{j'}$. On the other hand, if a node computes the summation polynomial of $r_i, r_{i+1}, \ldots, r_{i+k-1}, X_j, X_{j'}$ (the case of two variables), then its two child nodes compute the summation polynomials of $r_i, r_{i+1}, \ldots, r_{i+\lceil k/2 \rceil - 1}, X_j, X_{j''}$ and $r_i, r_{i+1}, \ldots, r_{i+k-1}, X_{j'}, X_{j''}$. This is allowed since summation polynomials are symmetric about its arguments. It is thus ensured that the number of variables in each node never exceeds two. At each node of the leftmost paths in the two subtrees of the root, the number of variables is $\leqslant 1$. At every other node in the tree, the number of variables is exactly two. Figure 1 shows the recursive construction of $f_{10}(r_1, r_2, \ldots, r_{10})$. Only the nodes on the paths from the root to the leaves (r_1, r_2, X_4) and (r_6, r_7, X_6) have numbers of variables $\leqslant 1$.

4.3 Running Time of SP

Let $C(t)$ denote the running time of Algorithm SP in the number of field operations on a batch of size t. In view of Eqn(10), we need to compute f_{t+1}. If $t = 2$, we use the base case which return in constant time. For $t \geqslant 3$, we use the

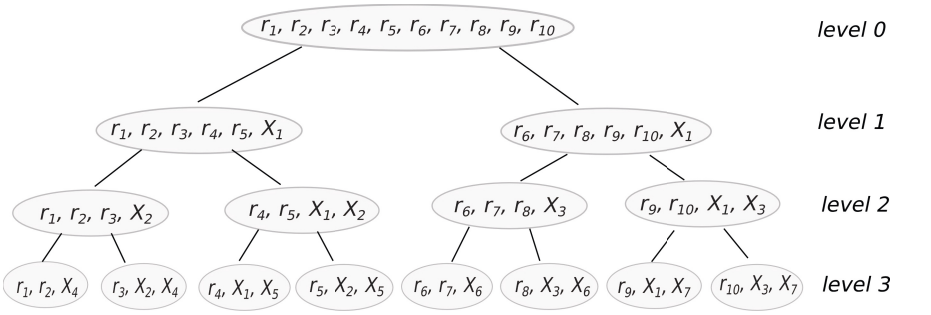

Fig. 1. Recursion tree for computing the summation polynomial of ten variables

recursive strategy of Eqn(11). Recursion stops in all cases at the base case of the computation of f_3. By Theorem 2, we need to take the resultant of two polynomials of degree $2^{\left(\frac{t-1}{2}\right)}$ each. The time complexity of resultant computation for two k-degree polynomials by the subresultant PRS algorithm [6,9] is $\mathrm{O}(k^2)$.

The running time of SP is dominated by the times for the computation of the resultants. The degrees of the polynomials, of which the resultant is computed, is a function of the level λ in the tree. In addition, the resultant-computation time depends on how many variables are involved at that node, call it ν. We can have $\nu = 0, 1, 2$ only. Let $C_\nu^{(\lambda)}$ denote the time for resultant computation for a given λ and ν. The case of $C_0^{(\lambda)}$ occurs at level $\lambda = 0$ only. The case of $C_1^{(\lambda)}$ occurs on the leftmost paths of the two subtrees of the root. At all other nodes, the resultant-computation cost is $C_2^{(\lambda)}$.

For simplicity, we assume that the recursion tree is a complete binary tree of height h, that is, $t + 1 = 2^h + 2$. We have $C_0^{(0)} = \mathrm{O}(2^{2^h})$, $C_1^{(\lambda)} = \mathrm{O}(2^{2^{h-\lambda-1}\times 3})$, and $C_2^{(\lambda)} = \mathrm{O}(2^{2^{h-\lambda+1}})$. At level zero, we have the cost C_0, whereas at any other level λ we have exactly two cases of $C_1^{(\lambda)}$ and $2^\lambda - 2$ cases of $C_2^{(\lambda)}$. Moreover, $C_1^{(\lambda)} < C_2^{(\lambda)}$ for each fixed λ. Therefore, the total cost $C(t)$ of computing f_{t+1} is of the order of

$$C_0^{(0)} + \sum_{\lambda=1}^{h-1} \left(2C_1^{(\lambda)} + (2^\lambda - 2)C_2^{(\lambda)}\right) < C_0^{(0)} + \sum_{\lambda=1}^{h-1} 2^\lambda C_2^{(\lambda)}$$
$$= \mathrm{O}\left(2^{2^h} + \sum_{\lambda=1}^{h-1} 2^{\lambda+2^{h-\lambda+1}}\right).$$

The inequality $\lambda + 2^{h-\lambda+1} > (\lambda+1) + 2^{h-(\lambda+1)+1}$ holds if and only if we have $2^{h-\lambda} > 1$. For all λ in the range $1 \leqslant \lambda \leqslant h-1$, we have $2^{h-\lambda} > 1$. Therefore, $C(t)$ is of the order of

$$2^{2^h} + \sum_{\lambda=1}^{h-1} \left[2^{\lambda+2^{h-\lambda+1}}\right] = 2^{2^h} + \left[2^{1+2^h} + 2^{2+2^{h-1}} + \cdots + 2^{(h-1)+2^2}\right]$$

$$< 2^{2^h} + \sum_{i=0}^{1+2^h} 2^i < 2^{2^h} + 2^{2+2^h} = 2^{2^h} + 2^{2+2^h} = 5 \times 2^{2^h}.$$

Substituting 2^h by $t-1$, we see that $C(t) = \mathrm{O}(m)$, where $m = 2^t$.

The above analysis implies that the computation of the resultants at the top two levels determines the order of $C(t)$. For a general t of the form $2^{h-1}+2 < t+1 \leqslant 2^h+2$, we let $\tau = \left\lceil \frac{t+1}{2} \right\rceil$, and conclude that $C(t)$ is of the order of

$$2^{\left\lceil \frac{t+1}{2} \right\rceil + \left\lfloor \frac{t+1}{2} \right\rfloor - 2} + 2 \times \left(2^{\left(\left\lceil \frac{\tau+1}{2} \right\rceil + \left\lfloor \frac{\tau+1}{2} \right\rfloor - 2\right)}\right)^2$$
$$= 2^{t-1} + 2^{2(\tau+1)-3} \leqslant 2^{t-1} + 2^t = \frac{3}{2} \times 2^t = \mathrm{O}(m).$$

4.4 Security of SP

In this section, we prove the equivalence between the security of Algorithm SP and the security of the standard ECDSA* batch-verification algorithm. Suppose that the x-coordinates $r_1, r_2, \ldots, r_t$ in ECDSA signatures are available to an adversary and that the batch is accepted by Algorithm SP. By Eqn(10), there exist exactly two solutions $(y_1, y_2, \ldots, y_t)$ and $(-y_1, -y_2, \ldots, -y_t)$ for the y-coordinates satisfying $y_i^2 = r_i^3 + ar_i + b$ for $i = 1, 2, \ldots, t$ such that $(r_1, y_1) + (r_2, y_2) + \cdots + (r_t, y_t) = (\alpha, \beta)$, and $(r_1, -y_1) + (r_2, -y_2) + \cdots + (r_t, -y_t) = (\alpha, -\beta)$. These are the only cases in which $f_{t+1}(r_1, r_2, \ldots, r_t, \alpha) = 0$. Both these solutions are consistent with $\phi = 0$ (Step 9 of Algorithm SP). One of these solutions corresponds to the ECDSA* signatures based upon the disclosed values $r_1, r_2, \ldots, r_t$. For ECDSA*, the y-coordinates are known, and we have only one possibility $(r_1, y_1) + (r_2, y_2) + \cdots + (r_t, y_t) = (\alpha, \beta)$. Given r_i alone, the adversary can obtain the y-coordinates y_i up to sign by making t square-root computations which demand only moderate computing resources. The sign ambiguity can be removed by trying all of the 2^t sign combinations (as in Algorithm N). For small values of t (as we deal with), this too is a tolerable overhead to the adversary. To sum up, if the adversary can forge ECDSA signatures that pass Algorithm SP, (s)he can produce in feasible time ECDSA* signatures too that pass the standard ECDSA* batch-verification algorithm. The converse is obvious.

4.5 Necessity of the Sanity Check

The security proof in the last section assumes that all y_i reside in $\mathbb{F}_p$ itself, that is, the points (r_i, y_i) lie on the curve E defined over $\mathbb{F}_p$. The sanity check made in Step 1 of Algorithm 1 ensures this.

The sanity check may be unnecessary in many situations. Suppose that an adversary chooses an r_i for which $r_i^3 + ar_i + b$ is a quadratic non-residue modulo p. The square roots of all quadratic non-residues in $\mathbb{F}_p$ lie in $\mathbb{F}_{p^2}$, that is, we now get two y-coordinates in $\mathbb{F}_{p^2}$ (but outside $\mathbb{F}_p$). The corresponding points $(r_i, \pm y_i)$ lie in $E(\mathbb{F}_{p^2})$. The right sides of Eqns(3) and (5) always lie in the group $E(\mathbb{F}_p)$ generated by the base point P. The batch-verification condition

demands the sum of $R_1, R_2, \ldots, R_t$ to lie in $E(\mathbb{F}_p)$ in order to pass the test $f_{t+1}(r_1, r_2, \ldots, r_t, \alpha) = 0$ (see Eqn(9)). If one or more of the points R_i are defined over $\mathbb{F}_{p^2}$ (but not over $\mathbb{F}_p$), then what is the probability of $\sum_{i=1}^{t} R_i \in E(\mathbb{F}_p)$?

A satisfactory answer to this question can be given if the group structure of $E(\mathbb{F}_{p^2})$ is known to us. $E(\mathbb{F}_p)$ is already a cyclic subgroup of $E(\mathbb{F}_{p^2})$ of large prime order n. If $E(\mathbb{F}_{p^2})$ is cyclic too, randomly chosen points $R_i \in E(\mathbb{F}_{p^2})$ have a probability of about $1/p$ to have their sum in $E(\mathbb{F}_p)$. Even when $E(\mathbb{F}_{p^2})$ is of rank two with no small-order subgroups (like $\mathbb{Z}_n \oplus \mathbb{Z}_n$), there may be little problem. The use of randomizers makes the probability of $\sum_{i=1}^{t} R_i \in E(\mathbb{F}_p)$ negligible even when the x-coordinates r_i are carefully crafted by the adversary. Only when $E(\mathbb{F}_{p^2})$ contains subgroups of small orders, the adversary may win with non-negligible probability. Randomizers do not seem to help much in this case. Section 8 deals with the cases of some of the NIST curves.

In Algorithm 1, the sanity check involves the computation of t Legendre symbols $\left(\frac{r_i^3+ar_i+b}{p}\right)$. This is anyway not a huge overhead compared to the computation of f_{t+1} (unless t is very small). Consequently, there is little harm in conducting the sanity check even when the probability of $\sum_{i=1}^{t} R_i \in E(\mathbb{F}_p)$ for points R_i defined over $\mathbb{F}_{p^2}$ is overwhelmingly small.

A sanity check like this may be needed for the previously published algorithms S2 and S2′ too. This issue is only mentioned but not discussed in detail in [13].

4.6 Cases of Failure of SP

The symbolic-manipulation algorithms of [13] have a few cases of failure. Algorithm SP is robust against most of these failures. First, computations which treat y_i as symbols cannot distinguish between the cases of point addition and point doubling. On the contrary, summation polynomials work equally well for both of these operations. Second, Algorithms S1, S2, S1′ and S2′ fail when the point $R = (\alpha, \beta)$ computed from the right side of Eqn(3) or (5) is the point $\mathcal{O}$ at infinity. Algorithm SP continues to work. Eqn(10) is now rewritten as $(r'_1, y'_1) + (r'_2, y'_2) + \cdots + (r'_t, y'_t) = \mathcal{O}$. That is, instead of computing $f_{t+1}(r'_1, r'_2, \ldots, r'_t, \alpha)$, we now compute $f_t(r'_1, r'_2, \ldots, r'_t)$.

5 Adaptation of Algorithm SP to Koblitz Curves

Let E be a Koblitz curve defined over a binary field $\mathbb{F}_{2^d}$ by the equation

$$E : y^2 + xy = x^3 + ax^2 + 1, \text{ where } a \in \{0, 1\}. \tag{15}$$

Let n be the order of the group we work in, and $\hat{h} = |E(\mathbb{F}_{2^d})|/n$ the cofactor. For Koblitz curves, $\hat{h} = 2$ or 4. Because $\hat{h}$ is small, appending a few extra bit(s) to ECDSA signatures, we can uniquely retrieve the x-coordinates from the

published value of r in a signature. We therefore assume that the x-coordinates are known to us. We denote these x-coordinates by r_i itself. We can apply our batch-verification Algorithm SP *mutatis mutandis* to Koblitz curves.

5.1 Summation Polynomials for Koblitz Curves

Here, we supply only the first three base cases of summation polynomials f_2, f_3, and f_4. The recurrence relation for Koblitz-curve summation polynomials f_t with $t \geqslant 5$ is identical to the case of prime curves.

$$\begin{aligned} f_2(x_1, x_2) &= x_1 + x_2, \\ f_3(x_1, x_2, x_3) &= (x_1x_2 + x_1x_3 + x_2x_3)^2 + x_1x_2x_3 + 1, \\ f_4(x_1, x_2, x_3, x_4) &= (x_1+x_2+x_3+x_4)^4 + (x_1x_2x_3+x_1x_2x_4+x_1x_3x_4+x_2x_3x_4)^4 + \\ & \quad x_1x_2x_3x_4(x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 + x_1 + x_2 + x_3 + x_4)^2 + \\ & \quad (x_1x_2x_3x_4)^2(x_1 + x_2 + x_3 + x_4)^2 + (x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4)^2. \end{aligned}$$

Eqn(9) holds for Koblitz curve too, that is, there exist $y_1, y_2, \ldots, y_t \in \mathbb{F}_{2^d}$ with each (x_i, y_i) satisfying Eqn (15) if and only if $f_t(x_1, x_2, \ldots, x_t) = 0$.

For prime curves, we always reduce the recursion to the computation of f_3, since the explicit formula for f_4 is rather clumsy. For Koblitz curves, we use both the cases f_3 and f_4 as those that terminate recursion. This helps us to reduce the height of the recursion tree for most of the batch sizes.

Notice that all batch-verification algorithms for Koblitz curves can be readily adapted to other ordinary (non-supersingular) curves over binary fields. We deal with the NIST family of Koblitz curves as an illustrative sample.

5.2 Adaptation of the Sanity Check

The sanity check (the equivalent of Step 1 in Algorithm 1) is quite easy in the context of NIST Koblitz curves. In order that the point $R_i = (r_i, y_i)$ is defined over $\mathbb{F}_{2^d}$, we now need the equation $y_i^2 + r_iy_i + (r_i^3 + ar_i^2 + 1) = 0$ to be solvable (for y_i) in $\mathbb{F}_{2^d}$. This is equivalent to the condition that the trace of $\frac{r_i^3 + ar_i^2 + 1}{r_i^2}$ over $\mathbb{F}_2$ is zero. Let the field $\mathbb{F}_{2^d} = \mathbb{F}_2[X]/\langle F(X)\rangle$ be defined by the irreducible polynomial $F(X) = X^d + a_{d-1}X^{d-1} + \cdots + a_1X + a_0$, where $a_i \in \mathbb{F}_2$. Let $\theta \in \mathbb{F}_{2^d}$ be a root of $F(X)$. Any element $c \in \mathbb{F}_{2^d}$ can be represented as $c = \sum_{k=0}^{d-1} c_k\theta^k$. We can compute the trace of c as $\mathrm{Tr}(c) = c_0 + \sum_{k=1}^{d-1} kc_ka_{d-k}$ (see [8] for a discussion). For NIST Koblitz curves, the defining polynomial $F(X)$ has very few non-zero coefficients, so the computation of $\mathrm{Tr}(c)$ is essentially a constant-time effort given any $c \in \mathbb{F}_{2^d}$.

Even when the solutions for y_i lie in $\mathbb{F}_{2^d}$, there is no guarantee that the point $R_i = (r_i, y_i)$ belongs to the subgroup of $E(\mathbb{F}_{2^d})$ generated by the base point P, since Koblitz curves have cofactors $\hat{h} > 1$. At present, we do not know any efficient solution of this problem. If $E(\mathbb{F}_{2^d})$ is cyclic, then R_i is in the subgroup generated by P if and only if $nR_i = \mathcal{O}$. However, computing the scalar multiplication nR_i for each i lets us forfeit the speedup obtained by batch verification.

Table 1. Times (in ms) for finding y_i from r_i

Square-root method in prime fields	P-256	0.28
	P-521	0.76
Factorization method in binary fields	K-283	8.31
	K-571	30.83

Table 2. Times (in ms) of scalar multiplication for prime curves

Scalar-multiplication algorithm	P-256		P-521		
	$l = 128$	$l = 256$	$l = 128$	$l = 256$	$l = 521$
Numeric scalar multiplication	2.04	3.92	2.69	5.48	10.60
Seminumeric scalar multiplication	2.04	4.12	2.81	5.92	11.44
Multiple scalar multiplication $\xi_1 R_1 + \xi_2 R_2$	2.93	5.67	4.37	7.89	16.30

l is the bit length of the randomizers

Table 3. Times (in ms) of scalar multiplication for Koblitz curves

Scalar-multiplication algorithm	K-283		K-571		
	$l = 128$	$l = 283$	$l = 128$	$l = 256$	$l = 571$
Numeric scalar multiplication	200.00	216.00	517.00	964.00	1076.00
Seminumeric scalar multiplication	267.10	288.93	718.88	1378.06	1532.89
Multiple scalar multiplication $\xi_1 R_1 + \xi_2 R_2$	443.46	973.75	1170.09	2344.40	5215.14

l is the bit length of the randomizers

6 Experimental Results

All experiments are carried out in a 2.33 GHz Xeon server running Ubuntu Linux Server Version 2012 LTS. The algorithms are implemented using the GP/PARI calculator [20] (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 in which function-call overheads are minimized as much as possible. We have used the best formulas supplied in [5,8]. We only used the built-in field arithmetic provided by the calculator. Since all algorithms are evaluated in terms of number of field operations, this gives a fare comparison of experimental data with the theoretical estimates. The GP/PARI library is much slower for binary fields than for prime fields. However, this speed difference matters only slightly in the experimental speedup figures which are ratios. As argued in Section 2.3, we consider only the case of the same signer.

The average times of finding the roots y_i from r_i are listed in Table 1. The average times of randomization achieved by the seminumeric algorithm [14] and numeric scalar multiplication are listed in Tables 2 and 3 for prime curves and Koblitz curves. Here, l denotes the length of the randomizers. We consider only two cryptographically meaningful values of l: 128 (giving 128-bit security irrespective of the difficulty of the ECDLP) and half-length (same security as offered

by the square-root methods for the ECDLP). Tables 4 and 5 list the overheads associated with different ECDSA batch-verification algorithms for several batch sizes with all the signatures coming from the same signer. Finally, the speedup figures (over individual verification) are listed in Tables 6 and 7 for prime curves and Koblitz curves.

Table 4. Overheads (in ms) of different batch-verification algorithms for prime curves

Batch size	N		N′		S2′		SP	
(t)	P-256	P-521	P-256	P-521	P-256	P-521	P-256	P-521
2	0.091	0.126	0.022	0.031	–	–	0.038	0.080
3	0.289	0.401	0.034	0.050	0.081	0.158	0.121	0.153
4	0.788	1.097	0.048	0.067	0.183	0.315	0.144	0.267
5	1.813	2.585	0.063	0.086	0.391	0.596	0.211	0.377
6	4.316	6.229	0.075	0.106	0.701	1.062	0.446	0.789
7	10.104	14.667	0.080	0.112	1.493	2.213	0.663	1.167
8	23.191	33.265	0.098	0.130	3.574	5.398	1.464	2.698
9	–	–	–	–	–	–	2.385	4.240
10	–	–	–	–	–	–	8.234	15.598

Table 5. Overheads (in ms) of different batch-verification algorithms for Koblitz curves

Batch size	N		N′		S2′		SP	
(t)	K-283	K-571	K-283	K-571	K-283	K-571	K-283	K-571
2	19.75	53.16	4.96	13.86	3.96	10.57	1.640	4.545
3	73.60	200.21	9.42	25.87	12.00	32.71	4.378	12.710
4	212.18	581.49	13.80	37.65	33.84	91.12	48.927	179.337
5	556.25	1544.59	18.16	49.03	131.00	354.30	62.790	220.271
6	1372.07	3791.10	22.34	60.61	303.08	825.00	141.345	458.448
7	3356.48	9161.95	27.00	72.89	1000.61	2749.37	585.654	1784.127
8	–	–	–	–	–	–	734.916	2277.102
9	–	–	–	–	–	–	1409.530	4340.618
10	–	–	–	–	–	–	3258.385	9602.883

In Algorithm N′, there is a possibility of using multiple scalar multiplication. The times for computing the sum $\xi_1 R_1 + \xi_2 R_2$ using a single double-and-add loop are also listed in the Tables 2 and 3. For prime curves, the multiple scalar-multiplication times are significantly less than that of two separate scalar multiplications by the most efficient windowed NAF variant. However, for Koblitz curves, the sum of times to compute two numeric scalar multiplications by the τ-NAF method [22] is much smaller than the time of a double scalar multiplication. While calculating the speedup figures, we have considered the best available options. Algorithm N′ is suited to ECDSA#. We first obtain each y_i uniquely by a square-root computation. Randomization in this case uses numeric scalar multiplication (or double scalar multiplication whichever is better).

Table 6. Speedup obtained by batch-verification algorithms for prime curves

Batch-verification algorithm	Randomization algorithm	t	P-256		P-521		
			None*	$l = 128$	None*	$l = 128$	$l = 256$
N	Numeric	3	2.50	1.32	2.58	1.81	1.38
		4	2.99	1.44	3.19	2.09	1.54
		5	3.19	1.49	3.59	2.26	1.63
		6	2.92	1.42	3.61	2.26	1.63
		7	2.24	1.24	3.14	2.07	1.53
		8	1.46	0.96	2.34	1.69	1.31
N′	Numeric	3	2.60	1.48	2.63	1.90	1.53
		4	3.32	1.79	3.36	2.32	1.85
		5	3.98	1.89	4.04	2.58	1.97
		6	4.59	2.10	4.67	2.87	2.19
		7	5.15	2.14	5.25	3.04	2.24
		8	5.68	2.31	5.80	3.27	2.41
S2′	Seminumeric	3	2.96	1.36	2.97	1.96	1.43
		4	3.87	1.53	3.92	2.34	1.62
		5	4.68	1.64	4.82	2.63	1.75
		6	5.34	1.72	5.63	2.86	1.85
		7	5.54	1.74	6.16	2.99	1.90
		8	4.91	1.67	6.01	2.95	1.89
SP	Seminumeric	3	2.94	1.36	2.97	1.97	1.43
		4	3.90	1.54	3.94	2.34	1.62
		5	4.82	1.66	4.89	2.65	1.76
		6	5.56	1.74	5.72	2.88	1.86
		7	6.27	1.80	6.53	3.07	1.94
		8	6.36	1.81	6.86	3.14	1.97
		9	6.34	1.81	7.14	3.20	1.99
		10	4.08	1.56	5.11	2.72	1.79

* Without randomization

The experimental results clearly indicate that SP is the most efficient batch-verification algorithm for standard ECDSA signatures. Even for ECDSA# signatures, Algorithm SP often outperforms the naive method N′. The optimal batch size for Algorithm S2′ is $t = 7$ (for prime curves) and $t = 5$ or 6 (for Koblitz curves). With Algorithm SP, the optimal batch sizes are $t = 9$ (for prime curves) and $t = 6$ (for Koblitz curves). For both these families, the maximum speedup is noticeably higher in Algorithm SP than what is achieved by Algorithm S2′.

The implications associated with the sanity check (Step 1 of Algorithm 1) are now discussed. For prime curves, the sanity check incurs negligible overhead. Without this check, the maximum achievable speedup figures for $t = 9$ are 6.55, 1.95, 7.25, 3.22 and 2.00. The corresponding row in Table 6 shows slightly smaller speedup values 6.34, 1.81, 7.14, 3.20 and 1.99 caused by the check. Similar observations hold for Algorithm S2′ too. For Koblitz curves, the sanity check is very efficient and does not produce noticeable performance degradation.

Table 7. Speedup obtained by batch-verification algorithms for Koblitz curves

Batch-verification algorithm	Randomization algorithm	t	K-283		K-571		
			None*	$l = 128$	None*	$l = 128$	$l = 256$
N	Numeric	2	1.84	0.99	1.90	1.30	1.03
		3	2.44	1.15	2.64	1.62	1.21
		4	2.55	1.17	3.01	1.75	1.28
		5	2.10	1.06	2.79	1.67	1.24
		6	1.40	0.85	2.11	1.40	1.08
		7	0.79	0.58	1.31	0.99	0.82
N′	Numeric	2	1.90	1.01	1.93	1.32	1.04
		3	2.78	1.22	2.84	1.69	1.25
		4	3.61	1.35	3.72	1.96	1.40
		5	4.39	1.45	4.57	2.18	1.50
		6	5.14	1.52	5.39	2.35	1.58
		7	5.85	1.58	6.17	2.49	1.64
S2′	Seminumeric	2	1.99	1.18	1.99	1.49	1.17
		3	2.94	1.46	2.97	1.98	1.45
		4	3.78	1.64	3.88	2.35	1.64
		5	4.08	1.69	4.48	2.55	1.74
		6	3.94	1.67	4.73	2.63	1.78
		7	2.56	1.36	3.69	2.27	1.61
SP	Seminumeric	2	1.99	1.18	2.00	1.49	1.17
		3	2.98	1.47	2.99	1.99	1.46
		4	3.69	1.62	3.78	2.31	1.62
		5	4.51	1.76	4.66	2.61	1.77
		6	4.82	1.81	5.22	2.78	1.84
		7	3.48	1.58	4.42	2.53	1.78
		8	3.52	1.59	4.59	2.59	1.76
		9	2.62	1.37	3.73	2.29	1.61
		10	1.51	0.99	2.42	1.72	1.31

* Without randomization

7 Summation Polynomial for Edwards Curves

Edwards curves are introduced by Edwards in [10]. Bernstein and Lange apply these curves to cryptographic usage [4]. Edwards curves offer faster addition and doubling formulas than elliptic curves. Moreover, the unified addition and doubling formulas make Edwards-curve cryptosystems resistant to simple side-channel attacks. An Edwards curve over a prime field is defined by the equation:

$$x^2 + y^2 = c^2(1 + dx^2y^2), \text{ where } cd(1 - dc^4) \neq 0.$$

The sum of two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on this curve is given as

$$(x_3, y_3) = \left(\frac{x_1y_2 + x_2y_1}{c(1 + dx_1x_2y_1y_2)}, \frac{y_1y_2 - x_1x_2}{c(1 - dx_1x_2y_1y_2)} \right).$$

This formula holds even when $P_1 = \pm P_2$.

EdDSA is the Edwards-curve equivalent of ECDSA [3]. Like ECDSA, only the y-coordinate of an Edwards-curve point is sent in an EdDSA signature. An extra bit to identify the correct x-coordinate is included in the signature. As a result, all the batch-verification algorithms studied in connection with ECDSA apply equally well to EdDSA signatures. Here, we mention the adaptation necessary to make Algorithm SP work for EdDSA batch verification. The original proposal of EdDSA refers to a batch-verification method akin to Algorithm N′.

The two base cases f_2 and f_3 of Edwards-curve summation polynomials are given by

$$\begin{aligned} f_2(y_1, y_2) &= y_1 - y_2, \\ f_3(y_1, y_2, y_3) &= c^2(B - d^2Ay_1^2y_2^2)y_3^2 - 2y_1y_2(B - dA)y_3 + (By_1^2y_2^2 - A), \\ &\quad \text{where } A = (c^2 - y_1^2)(c^2 - y_2^2) \text{ and } B = (1 - c^2dy_1^2)(1 - c^2dy_2^2). \end{aligned}$$

The recurrence relation for Edwards-curve summation polynomials f_t for $t \geqslant 4$ is the same as for prime/Koblitz curves. The sanity check for Edwards curves follows the same procedure as for elliptic curves.

8 The Group Structures in Quadratic Extensions

Here, we investigate the groups $E(\mathbb{F}_{p^2})$ and $E(\mathbb{F}_{2^{2d}})$ for the NIST prime and Koblitz curves [19] for which we have reported our experimental results. Since the sizes of the groups over the base fields are known, it is easy to compute the orders of the groups over quadratic extensions using a well-known result by Weil [8]. These sizes give an initial (sometimes complete) understanding of the structures of the groups over the extension fields.

The curve P-256 is defined over $\mathbb{F}_p$ for a 256-bit prime p. The order of $E(\mathbb{F}_p)$ is a prime n, so $E(\mathbb{F}_p)$ is cyclic. The size of $E(\mathbb{F}_{p^2})$ is $|E(\mathbb{F}_{p^2})| = 3 \times 5 \times 13 \times 179 \times n \times n'$, where $n' \neq n$ is a 241-bit prime. Since $|E(\mathbb{F}_{p^2})|$ is square-free, the group $E(\mathbb{F}_{p^2})$ is cyclic. However, it contains subgroups of small orders.

The curve P-521 is defined over $\mathbb{F}_p$ for a 521-bit prime p. The order of $E(\mathbb{F}_p)$ is a prime n, so $E(\mathbb{F}_p)$ is cyclic. The size of $E(\mathbb{F}_{p^2})$ is $|E(\mathbb{F}_{p^2})| = 5 \times 7 \times 69697531 \times 635884237 \times n \times n'$, where $n' \neq n$ is a 461-bit prime. Again $E(\mathbb{F}_{p^2})$ is cyclic, since its order is square-free. This group too has subgroups of small orders.

The Koblitz curve K-283 is defined over $\mathbb{F}_{2^d}$, $d = 283$, and has an order $4n$ for a prime n. If the group $E(\mathbb{F}_{2^d})$ is not cyclic, we must have $E(\mathbb{F}_{2^d}) \cong \mathbb{Z}_{2n} \oplus \mathbb{Z}_2$. But then, by the structure theorem of elliptic-curve groups of rank two, we have $2|(2^d - 1)$, which is impossible. So $E(\mathbb{F}_{2^{283}})$ is cyclic. In the quadratic extension $\mathbb{F}_{2^{2d}}$, the group has order $|E(\mathbb{F}_{2^{2d}})| = 2^3 \times 5 \times 250057 \times 43611431 \times n \times n'$ for

a 238-bit prime n' (different from n). As argued above, $E(\mathbb{F}_{2^{2d}})$ is easily seen to be cyclic. However, it contains subgroups of small orders.

The Koblitz curve K-571 is defined over $\mathbb{F}_{2^d}$, $d = 571$, and has order $4n$ for a prime n. We have the order $|E(\mathbb{F}_{2^{2d}})| = 2^3 \times 83520557720108799306580699 \times 59620168636271854235471070 1 \times n \times n'$ for a 395-bit prime $n' \neq n$. Both $E(\mathbb{F}_{2^d})$ and $E(\mathbb{F}_{2^{2d}})$ are cyclic. Again, $E(\mathbb{F}_{2^{2d}})$ contains subgroups of small orders.

Since each of these groups in the quadratic extension has small-order subgroups, the sanity check is apparently preferred for all these curves. However, if the points of small orders on a curve over the quadratic extension do not have x-coordinates in the base field, then we can eliminate the sanity check.

9 Conclusion

In this paper, we propose a new and efficient batch-verification algorithm for original ECDSA signatures. Our algorithm outperforms all previously known batch-verification algorithms for ECDSA. We theoretically and experimentally establish this superiority for the NIST prime and Koblitz families of elliptic curves. We also mention how the methods can be adapted to EdDSA signatures. Theoretical and experimental performance comparisons of different batch-verification algorithms for EdDSA signatures remains an open (albeit fairly straightforward) problem. The elliptic-curve group structures over quadratic extensions of the base fields need to be determined for all NIST curves, at the minimum to gauge the necessity of running the sanity check.

References

1. 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)
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. Ghosh, S., Roychowdhury, D., Das, A.: High speed cryptoprocessor for η_T pairing on 128-bit secure supersingular elliptic curves over characteristic two fields. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 442–458. Springer, Heidelberg (2011)
4. 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)
5. Bernstein, D.J., Lange, T.: Explicit-formulas database (2007), `http://www.hyperelliptic.org/EFD/`
6. Brown, W.S.: The subresultant PRS algorithm. ACM Transactions on Mathematical Software 4(3), 237–249 (1978)
7. 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)

8. Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of elliptic and hyperelliptic curve cryptography. CRC Press (2006)
9. Collins, G.E.: Subresultants and reduced polynomial remainder sequences. Journal of ACM 14(1), 128–142 (1967)
10. Edwards, H.M.: A normal form for elliptic curves. Bulletin of American Mathematical Society 44(3), 393–422 (2007)
11. Harn, L.: Batch verifying multiple RSA digital signatures. Electronics Letters 34(12), 1219–1220 (1998)
12. Johnson, D., Menezes, A.J., Vanstone, S.A.: The Elliptic Curve Digital Signature Algorithm (ECDSA). International Journal of Information Security 1(1), 36–63 (2001)
13. 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)
14. Karati, S., Das, A., Roychowdhury, D.: Using randomizers for batch verification of ECDSA signatures, IACR Cryptology ePrint Archive (2012), http://eprint.iacr.org/2012/582
15. Montgomery, P.L.: Speeding up Pollard and elliptic curve methods of factorization. Mathematics of Computation 48(177), 243–264 (1987)
16. Naccache, D., M'Raïhi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A. Be improved? In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 77–85. Springer, Heidelberg (1995)
17. NIST: Digital Signature Standard (DSS), http://csrc.nist.gov/publications/drafts/fips_186-3/Draft-FIPS-186-3
18. NIST: Secure Hash Standard, SHS (2007), http://csrc.nist.gov/publications/drafts/fips_180-3/draft_fips-180-3_June-08-2007.pdf
19. NIST: Recommended elliptic curves for federal government use (1999), http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf
20. PARI Group: PARI/GP Home (2003-2013), http://pari.math.u-bordeaux.fr/
21. Semaev, I.: Summation polynomials and the discrete logarithm problem on elliptic curves (2004), http://eprint.iacr.org/2004/031
22. Solinas, J.A.: Improved algorithms for arithmetic on anomalous binary curves, Combinatorics and Optimization Research Report CORR 99-46, University of Waterloo (1999), http://www.cacr.math.uwaterloo.ca/techreports/1999/corr99-46.ps