J Cryptogr Eng
DOI 10.1007/s13389-014-0082-x

New Algorithms for Batch Verification of Standard ECDSA Signatures

Sabyasachi Karati · Abhijit Das · Dipanwita Roychowdhury · Bhargav Bellur · Debojyoti Bhattacharya · Aravind Iyer

Received: 1 January 2014 / Accepted: 27 June 2014

Abstract In this paper, several algorithms for batch verification of ECDSA signatures are studied. The first of these algorithms is based upon the naive idea of taking square roots in the underlying field. In order to improve the efficiency beyond what can be achieved by the naive algorithm, two new algorithms are proposed which replace square-root computations by symbolic manipulations. Experiments carried out on NIST prime curves demonstrate a maximum speedup of above six over individual verification if all the signatures in the batch belong to the same signer, and a maximum speedup of about two if the signatures in the batch belong to different signers, both achieved by a fast variant of the second symbolic-manipulation algorithm. In terms of security, all the studied algorithms are equivalent to standard ECDSA* batch verification. These algorithms are practical only for small ($\leqslant 8$) batch sizes. The algorithms are also ported to the NIST Koblitz curves defined over fields of characteristic 2. This appears to be the first reported study on the batch verification of standard ECDSA signatures.

Keywords Digital Signatures · Elliptic Curves · ECDSA · Batch Verification · Symbolic Computation · Linearization

Mathematics Subject Classification (2000) 94A60 · 14H52 · 11T06 · 11T71 · 11Y16 · 68W30

CR Subject Classification Mathematics of computing ∼ Number-theoretic computations · Security and privacy ∼ Digital signatures · Theory of computation ∼ Cryptographic protocols · Computing methodologies ∼ Symbolic and algebraic algorithms · Proper nouns: People, technologies and companies ∼ National Institute of Standards and Technology

This paper was presented in part at AfricaCrypt 2012

Sabyasachi Karati · Abhijit Das · Dipanwita Roychowdhury
Computer Science and Engineering, Indian Institute of Technology Kharagpur, India
E-mail: skarati, abhij, drc@cse.iitkgp.ernet.in

Bhargav Bellur · Debojyoti Bhattacharya · Aravind Iyer
General Motors Technical Centre India, India Science Lab, Bangalore, India
E-mail: bhargav_bellur@yahoo.com, debojyoti.bhattacharya@gmail.com, aravindiyer@facebook.com

1 Introduction

Since the discoveries of Diffie and Hellman [7] and Rivest et al. [18], digital signatures have been widely used to achieve source authentication and data integrity. At present, most digital signature algorithms are based on two popular pubic-key algorithms:

- RSA signature scheme [18], the security of which is based on the difficulty of factoring large composite integers.
- ElGamal signature scheme [8], the security of which is based on the difficulty of computing discrete logarithms in certain groups, like multiplicative groups of finite fields and groups of rational points on elliptic curves over finite fields. Two variants of this scheme are accepted as standards: the digital signature algorithm (DSA) of NIST [16] and the elliptic curve digital signature algorithm (ECDSA) from Johnson and Menezes [11].

To verify an ElGamal-like signature, one requires at least two finite-field exponentiations (for DSA) or two elliptic-curve scalar multiplications (for ECDSA). Each such modular exponentiation or scalar multiplication is a time-consuming operation.

Batch verification is used to verify multiple digital signatures in time less than total individual verification time. The concept of batch verification is introduced by Naccache et al. [14]. They propose an interactive DSA batch-verification protocol. In this protocol, the signer generates t signatures through interaction with the verifier, and then the verifier validates all these t signatures simultaneously.

Harn, in 1998, proposes an efficient scheme for the batch verification of RSA signatures [9]. In this scheme (also see [10]), multiple signatures signed by the same private key can be verified simultaneously. Harn's scheme uses only one exponentiation for batches of any size t. There are some weaknesses in this scheme. For example, if batch verification fails, we cannot identify the faulty signature(s) without making individual verification. Moreover, Harn's scheme does not adapt to the case of signatures from multiple signers.

These protocols are not directly applicable to ECDSA signatures. ECDSA requires smaller key and signature sizes than DSA and RSA, so there has been a growing interest in ECDSA. ECDSA*, a modification of ECDSA introduced by Antipa et al. [1], permits an easy adaptation of Naccache et al's batch-verification protocol for DSA. Cheon and Yi [4] study batch verification of ECDSA* signatures, and report speedup factors of up to 7 for same signer and 4 for different signers. However, ECDSA* is not a standard, and is thus unacceptable, particularly in applications where interoperability is of important concern. More importantly, ECDSA* increases the signature size compared to ECDSA without any increase in the security. Consequently, batch verification of original ECDSA signatures turns out to be a practically important open research problem. To the best of our knowledge, no significant result in this area has ever been reported in the literature.

In this paper, we propose three algorithms to verify *standard* ECDSA signatures in batches. Our algorithms apply to all cases of ECDSA signatures sharing the same curve parameters, although we obtain good speedup figures when all the signatures in the batch come from the same signer. Our algorithms are effective only for small batch sizes (like $t \leqslant 8$). The first algorithm we introduce (henceforth denoted as Algorithm N) is based upon a naive approach of taking square roots in the underlying field. As the field size increases, square-root computations become quite costly. We modify Algorithm N by replacing square-root calculations by symbolic manipulations. We propose two ECDSA batch-verification algorithms, called S1 and S2, using symbolic manipulations. Algorithm S1 is not very practical, but is discussed in this paper, for it provides the theoretical and practical foundations for arriving at Algorithm S2. For a wide range of field and batch sizes, Algorithm S2 convinc-

ingly outperforms the naive Algorithm N. Both S1 and S2 are probabilistic algorithms in the Monte Carlo sense, that is, they may occasionally fail to verify correct signatures. We analytically establish that for randomly generated signatures, the failure probability is extremely low.

An application of our batch-verification algorithms is in secure vehicle-to-vehicle (V2V) communications in vehicular ad hoc networks (VANETs) (see [6] for a survey). Since signature generation and verification are time-consuming operations, and since vehicles have to verify signatures repeatedly, any algorithm that speeds up the authentication process is of great help in V2V communications. In a busy street where a vehicle needs to authenticate messages from multiple vehicles in real time, individual verification may result in practical bottlenecks. In this situation, it is also expected that multiple messages from the same vehicle get accumulated for being verified. This is precisely the case when our batch-verification algorithms produce the maximum benefits. Like other batch-verification schemes, this performance gain comes at a cost, namely, we forfeit the ability to identify individual faulty signatures. Our algorithm (like any batch-verification algorithm) turns out to be useful only when most signatures are authentic.

The rest of this paper is organized as follows. In Section 2, we identify the problems associated with ECDSA batch verification. In this process, we introduce the ECDSA and ECDSA* signature schemes, and set up the notations which we use throughout the paper. We also introduce the naive batch-verification algorithm N in this section. Section 3 elaborates our new algorithm S1 based upon symbolic manipulations. Section 4 presents an analytic study of Algorithm S1. We furnish details about the running time, the cases of failure, and the security of Algorithm S1. Some of the calculations are omitted in the main text and supplied in the appendix. The running time estimates for Algorithm S1 indicate that this algorithm is expected to perform poorly unless the batch size t is very small. In Section 5, we improve upon this algorithm to arrive at Algorithm S2. Analytic results for Algorithm S2 are provided in Section 6. A heuristic capable of significantly speeding up Algorithms S1 and S2 is presented in Section 7. In Section 8, we list our experimental results, and compare the performances of the three algorithms N, S1 and S2. We also study the performances of three faster variants N$'$, S1$'$ and S2$'$ of these algorithms. Since randomization of the batch-verfication process eliminates many attacks, we deal with the effects of randomization on our algorithms in Section 9. Although we have concentrated only upon the NIST curves over prime fields [15], our algorithms readily apply to other curves with cofactor 1. The curves over binary fields defined in the same standard are not immediately suited to our algorithms, since these curves correspond to cofactor values larger than 1. As mentioned in [1], cofactor values larger than 1 can be easily handled by appending only a few bits of extra information to standard ECDSA signatures. Assuming that these extra bits of information are available, we port our algorithms to the Koblitz curves over binary fields, defined in the NIST standard. The necessary modifications and the associated experimental results are presented in Section 10. Section 11 concludes this paper after highlighting some future research directions.

2 A Naive Approach

Throughout the rest of this paper, we plan to simultaneously verify t ECDSA signatures $(r_1, s_1), (r_2, s_2), \ldots, (r_t, s_t)$ on messages $M_1, M_2, \ldots, M_t$. We start with a description of the ECDSA signature scheme from [11].

2.1 ECDSA Domain Parameters

ECDSA is based upon some parameters common to all entities participating in a network.

$q =$ Order of the prime field $\mathbb{F}_q$.
$E =$ An elliptic curve $y^2 = x^3 + ax + b$ defined over the prime field $\mathbb{F}_q$.
$P =$ A random non-zero base point in $E(\mathbb{F}_q)$.
$n =$ The order of P, typically a prime.
$h =$ The cofactor $\dfrac{|E(\mathbb{F}_q)|}{n}$.

2.2 Assumptions

For the time being, we assume that $h = 1$, that is, $E(\mathbb{F}_q)$ is a cyclic group, and P is a generator of $E(\mathbb{F}_q)$. This is indeed the case for certain elliptic curves standardized by NIST. By Hasse's theorem, we have $|n - q - 1| \leqslant 2\sqrt{q}$. If $n \geqslant q$, an element of $\mathbb{Z}_n$ has a unique representation in $\mathbb{Z}_q$. On the other hand, if $n < q$, an element of $\mathbb{Z}_n$ has at most two representations in $\mathbb{Z}_q$. The density of elements of $\mathbb{Z}_n$ having two representations in $\mathbb{Z}_q$ is $\leqslant 2/\sqrt{q}$ which is close to zero for large values of q.

In an ECDSA signature (M, r, s), the values r and s are known modulo n. However, r corresponds to an elliptic-curve point and should be known modulo q. If r corresponds to a random point on E, it uniquely identifies an element of $\mathbb{F}_q$ with probability close to 1. In view of this, we ignore the effect of issues associated with the ambiguous representation stated above, in the rest of this article.

Note that the ambiguities arising out of $h > 1$ and/or $q > n$ can be practically solved by appending only a few extra bits to standard ECDSA signatures [1,4]. Consequently, our assumptions are neither too restrictive nor too impractical.

2.3 The ECDSA Algorithm

Algorithm 1 computes the public key Q and the private key d of a signer. The ECDSA signature (r, s) on a message M is generated by Algorithm 2. Algorithm 3 verifies the ECDSA signature (r, s) on a message M.

Algorithm 1 ECDSA Key-pair Generation

INPUT: Domain Parameters.
OUTPUT: Public key Q, private key d.

1. The private key $= d \in_R [1, n-1]$.
2. The public key $= Q = dP \in E(\mathbb{F}_q)$.

Algorithm 2 ECDSA Signature Generation

INPUT: Domain Parameters, message M and signer's private key d.
OUTPUT: ECDSA signature (r, s).

1. k = A randomly chosen element in the range $[1, n-1]$ (the session key).
2. $R = kP$.
3. $r = x(R)$ (the x-coordinate of R) reduced modulo n.
4. $s = k^{-1}(H(M) + dr) \pmod{n}$, where H is a cryptographic hash function like SHA-1 of [17].

Algorithm 3 ECDSA Signature Verification

INPUT: Domain Parameters, message M, signature (r, s) and signer's public key Q.
OUTPUT: Accept/Reject.

1. $w = s^{-1} \pmod{n}$.
2. $u = H(M)w \pmod{n}$.
3. $v = rw \pmod{n}$.
4. $R = uP + vQ \in E(\mathbb{F}_q)$. (1)
5. Accept the signature if and only if $x(R) = r \pmod{n}$.

2.4 A Naive Algorithm for ECDSA Batch Verification

For t signed messages (M_i, r_i, s_i), $i = 1, 2, \ldots, t$, we have

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

If all the signatures belong to the same signer, we have $Q_1 = Q_2 = \cdots = Q_t = Q$ (say), and the last equation simplifies to:

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

The basic idea is to compute the two sides of Eqn (2) or Eqn (3), and check for the equality. Use of these equations reduces the number of scalar multiplications from $2t$ to $[2, t+1]$, where 2 corresponds to the case where all the signatures belong to same signer, and $t+1$ corresponds to the case where the t signers are distinct from one another. However, only the x-coordinates of R_i are known from the signatures. In general, there are two y-coordinates corresponding to a given x-coordinate, but computing these y-coordinates requires taking square roots modulo q, a time-consuming operation. Moreover, there is nothing immediately available in the signatures to remove the resulting ambiguity in these two values of y. Finally, computing all R_i using Eqn (1) misses the basic idea of batch verification, since after this expensive computation, there is only an insignificant amount of effort left to complete individual verifications of all the t signatures.

ECDSA*, a modification of ECDSA introduced by Antipa et al. [1], adapts readily to the above batch-verification idea. Nonetheless, a naive algorithm for the batch verification of original ECDSA signatures can be conceived of, as illustrated in Algorithm 4. Of course, this is an obvious way of solving the ECDSA batch-verification problem, but we have not found any previous mention of this algorithm in the literature. There are (usually) 2^t choices of the square roots y_i of $r_i^3 + ar_i + b$ for all $i = 1, 2, \ldots, t$. If any of these combinations of square roots satisfies Eqn (2), we accept the batch of signatures. Step 6 turns out to be a

costly operation. Moreover, Step 7 needs to check (at most) $m = 2^t$ possible conditions for batch verification, and is also quite costly unless t is small.

Algorithm 4 ECDSA Batch-verification Algorithm N

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

1. Compute $w_i = s_i^{-1} \pmod n$ for all $i = 1, 2, \ldots, t$.
2. Compute $u_i = H(M_i)w_i \pmod n$ for all $i = 1, 2, \ldots, t$.
3. Compute $v_i = r_i w_i \pmod n$ for all $i = 1, 2, \ldots, t$.
4. Compute $R' = (\sum_{i=1}^{t} u_i)P + \sum_{i=1}^{t} v_i Q_i \in E(\mathbb{F}_q)$.
   Club together the points Q_i from same signers during the computation of R'. For example, if all the signatures belong to the same signer, compute R' as $(\sum_{i=1}^{t} u_i)P + (\sum_{i=1}^{t} v_i)Q$.
5. For each $i = 1, 2, \ldots, t$, if $r_i^3 + ar_i + b$ is neither zero nor a quadratic residue modulo q, reject the i-th signature, and remove it from the batch.
6. For $i = 1, 2, \ldots, t$, compute the square roots of $r_i^3 + ar_i + b$ modulo q.
7. For each square root y_i of $r_i^3 + ar_i + b$ for all $i = 1, 2, \ldots, t$, do the following:
   If $R' = \sum_{i=1}^{t} (r_i, y_i)$, accept all the signatures.
8. Reject all the signatures.

We highlight here that using a single extra bit of information in an ECDSA signature, one can unambiguously identify the *correct* square root of $r_i^3 + ar_i + b$, and thereby avoid the $\Theta(2^t)$ overhead associated with Algorithm N. In that case, Step 7 of Algorithm 4 needs to be updated appropriately. This updated (and efficient) version of the naive algorithm will henceforth be denoted by Algorithm N$'$. Despite this updating, there is apparently nothing present in ECDSA signatures, that provides a support for quickly *computing* the correct square root. The basic aim of this paper is to develop algorithms to reduce the overhead associated with square-root calculations. In effect, we are converting ECDSA signatures to ECDSA* signatures. In that sense, this paper is not competing with but complementary to the earlier works of [1,4] on ECDSA*.

2.5 ECDSA*

The ECDSA* signature (R, s) on a message M is computed by Algorithm 5. Algorithm 6 verifies the ECDSA* signature (R, s) on a message M.

Algorithm 5 ECDSA* Signature Generation

INPUT: Domain Parameters, message M and signer's private key d.
OUTPUT: ECDSA signature (R, s).

1. $k =$ A randomly chosen element in the range $[1, n-1]$ (the session key).
2. $R = kP$.
3. $r = x(R)$ (the x-coordinate of R) reduced modulo n.
4. $s = k^{-1}(H(M) + dr) \pmod n$ (where H is a hash function).

Batch verification of ECDSA* signatures is a straightforward adaptation of the procedure of [14]. One needs to check whether Eqn (2) (or Eqn (3)) holds. Since the entire points

Algorithm 6 ECDSA* Signature Verification

INPUT: Domain Parameters, message M, signature (R, s) and signer's public key Q.
OUTPUT: Accept/Reject.

1. $w = s^{-1} \pmod{n}$.
2. $u = H(M)w \pmod{n}$.
3. $v = rw \pmod{n}$, where $r = x(R)$.
4. $R' = uP + vQ \in E(\mathbb{F}_q)$.
5. Accept the signature if and only if $R' = R$.

R_i are present in the signatures, the left side of this equation can be computed efficiently and unambiguously. But ECDSA* is still not accepted as a standard and results in increased signature sizes, so an algorithm that efficiently performs batch verification in presence of only the partial information r_i (only the x-coordinates of R_i) turns out to be practically useful.

3 A New Batch-verification Algorithm for ECDSA

In this section, we present a new algorithm to convert Eqn (2) or Eqn (3) to a form which eliminates the problems associated with the lack of knowledge of the y-coordinates of R_i. We compute the right side of Eqn (2) or Eqn (3) as efficiently as possible. The left side is not computed explicitly, but symbolically in the unknown values $y_1, y_2, \ldots, y_t$ (the y-coordinates of $R_1, R_2, \ldots, R_t$). By solving a system of linear equations over $\mathbb{F}_q$, we obtain enough information to verify the t signatures simultaneously. This new algorithm, called Algorithm S1, turns out to be faster than Algorithm N for small batch sizes (typically for $t \leqslant 4$) and for large underlying fields.

3.1 Symbolic Computation of $R = \sum_{i=1}^{t} R_i$

Let $R_i = (x_i, y_i)$. The x-coordinates $x_i = x(R_i)$ are available from the signatures, namely, $x_i = r_i$ or $x_i = r_i + n$. The second case pertains to the condition $n < q$ and has a very low probability. So we plan to ignore this case, and take $x_i = x(R_i) = r_i$. It is indeed easy to detect when the reduced x-coordinate r_i has two representatives in $\mathbb{F}_q$, and if so, we repeat Algorithm S1 for both these values.

Although the y-coordinate $y_i = y(R_i)$ is unknown to us, we know the value

$$y_i^2 = r_i^3 + ar_i + b \pmod{q} \tag{4}$$

for all $i = 1, 2, \ldots, t$, since $R_i = (r_i, y_i)$ is a point on the curve E.

Let $P_1 = (h_1, k_1)$ and $P_2 = (h_2, k_2)$ be two non-zero points on E with $P_1 \neq \pm P_2$. The sum $P_1 + P_2$ is another point (h, k) on E computed as follows:

$$\lambda = (k_2 - k_1)/(h_2 - h_1), \tag{5}$$
$$h = \lambda^2 - h_1 - h_2, \tag{6}$$
$$k = \lambda(h_1 - h) - k_1. \tag{7}$$

Applying this formula repeatedly lets us have the following representation of the point $R = \sum_{i=1}^{t} R_i$:

$$R = \left(\frac{g_x(y_1, y_2, \ldots, y_t)}{h_x(y_1, y_2, \ldots, y_t)}, \frac{g_y(y_1, y_2, \ldots, y_t)}{h_y(y_1, y_2, \ldots, y_t)} \right), \tag{8}$$

where g_x, g_y, h_x, h_y are polynomials in $\mathbb{F}_q[y_1, y_2, \ldots, y_t]$. In view of Eqn (4), we may assume that these polynomials have y_i-degrees $\leqslant 1$ for all $i = 1, 2, \ldots, t$. This implies that the denominator $h_x(y)$ is of the form $u(y_2, y_3, \ldots, y_t)y_1 + v(y_2, y_3, \ldots, y_t)$. Multiplying both g_x and h_x by $u(y_2, y_3, \ldots, y_t)y_1 - v(y_2, y_3, \ldots, y_t)$ and making use of Eqn (4), we can eliminate y_1 from the denominator. Repeating this successively for $y_2, y_3, \ldots, y_t$ allows us to represent the point R as a pair of polynomial expressions:

$$R = (R_x(y_1, y_2, \ldots, y_t), R_y(y_1, y_2, \ldots, y_t)) \tag{9}$$

with the polynomials R_x and R_y linear individually with respect to all y_i. It is useful to clear the denominator after every symbolic addition instead of only once after the entire sum $R = \sum_{i=1}^{t} R_i$ is computed symbolically. A pseudocode for the symbolic-addition procedure is supplied as Algorithm 8 which uses Algorithm 7 for clearing the denominator of rational functions.

Algorithm 7 Denominator Clearing in a Rational Function

INPUT: Rational function f/g with $f, g \in \mathbb{F}_q[y_j, y_{j+1}, \ldots, y_k]$, and x-coordinates $r_j, r_{j+1}, \ldots, r_k$.

OUTPUT: A polynomial in $\mathbb{F}_q[y_j, y_{j+1}, \ldots, y_k]$ equal to f/g.

STEPS

For $i = j, j+1, \ldots, k$, repeat the following steps:

1. Express $g = uy_i + v$ with $u, v \in \mathbb{F}_q[y_{i+1}, y_{i+2}, \ldots, y_k]$.
2. If $u = 0$ (that is, if y_i does not appear in g), go to the top of the loop.
3. Set $g = u^2 \times (r_i^3 + ar_i + b) - v^2$.
4. For $i' = i+1, i+2, \ldots, k$, substitute $y_{i'}^2$ by $r_{i'}^3 + ar_{i'} + b$ in g.
5. Set $f = f \times (uy_i - v)$.
6. For $i' = j, j+1, \ldots, k$, substitute $y_{i'}^2$ by $r_{i'}^3 + ar_{i'} + b$ in f.

Output f/g. (After the above loop, $f \in \mathbb{F}_q[y_j, y_{j+1}, \ldots, y_k]$, and $g \in \mathbb{F}_q$.)

In Appendix A, we establish that R_x is a polynomial with each non-zero term having even total degree, whereas R_y is a polynomial with each non-zero term having odd total degree.

From the right side of Eqn (2) or Eqn (3), we compute the x- and y-coordinates of R as

$$R = (\alpha, \beta)$$

for some $\alpha, \beta \in \mathbb{F}_q$. This gives us two multivariate equations to start with:

$$R_x(y_1, y_2, \ldots, y_t) = \alpha, \tag{10}$$

$$R_y(y_1, y_2, \ldots, y_t) = \beta. \tag{11}$$

Algorithm 8 Symbolic Addition of Elliptic-curve Points

INPUT: x-coordinates $r_j, r_{j+1}, \ldots, r_k \in \mathbb{F}_q$.

OUTPUT: Symbolic sum $(R_x, R_y) = \sum_{i=j}^{k} (r_i, y_i)$ with $R_x, R_y \in \mathbb{F}_q[y_j, y_{j+1}, \ldots, y_k]$, $(r_i, y_i) \in E(\mathbb{F}_q)$.

STEPS

1. If $j = k$, return (r_j, y_j).
2. Set $\tau = \lfloor (j+k)/2 \rfloor$.
3. Recursively compute $(R_x^{(1)}, R_y^{(1)}) = \sum_{i=j}^{\tau} R_i$.
4. Recursively compute $(R_x^{(2)}, R_y^{(2)}) = \sum_{i=\tau+1}^{k} R_i$.
5. Compute $\lambda = (R_y^{(2)} - R_y^{(1)})/(R_x^{(2)} - R_x^{(1)})$ as a rational function in $y_j, y_{j+1}, \ldots, y_k$.
6. Apply denominator clearing (Algorithm 7) on λ.
7. Set $R_x = \lambda^2 - R_x^{(1)} - R_x^{(2)}$.
8. For $i = j, j+1, \ldots, k$, substitute y_i^2 by $r_i^3 + ar_i + b$ in R_x.
9. Set $R_y = \lambda \times (R_x^{(1)} - x) - R_y^{(1)}$.
10. For $i = j, j+1, \ldots, k$, substitute y_i^2 by $r_i^3 + ar_i + b$ in R_y.

3.2 Solving the Multivariate Equations

We treat Eqns (10) and (11) as linear equations in the monomials y_i, $y_i y_j$, $y_i y_j y_k$, and so on. R_x contains non-zero terms involving only the even-degree monomials, that is, $y_i y_j$, $y_i y_j y_k y_l$, and so on. Throughout the rest of this paper, we denote $m = 2^t$. There are exactly $\mu = 2^{t-1} - 1 = \frac{m}{2} - 1$ such monomials. We name these monomials as $z_1, z_2, \ldots, z_\mu$, and take out the constant term from R_x to rewrite Eqn (10) as

$$\rho_{1,1} z_1 + \rho_{1,2} z_2 + \cdots + \rho_{1,\mu} z_\mu = \alpha_1. \tag{12}$$

If we square both sides of this equation, and use Eqn (4) to eliminate all squares of variables, we obtain another linear equation:

$$\rho_{2,1} z_1 + \rho_{2,2} z_2 + \cdots + \rho_{2,\mu} z_\mu = \alpha_2. \tag{13}$$

By repeated squaring, we generate a total of μ linear equations in $z_1, z_2, \ldots, z_\mu$. We then solve the resulting system and obtain the values of $z_1, z_2, \ldots, z_\mu$.

If the system is not of full rank, we make use of Eqn (11) as follows. Each non-zero term in R_y has odd degree. However, the equation $R_y^2 = \beta^2$ (along with the substitution given by Eqn (4)) leads to a linear equation in the even-degree monomials $z_1, z_2, \ldots, z_\mu$ only. Repeated squaring of this equation continues to generate a second sequence of linear equations in $z_1, z_2, \ldots, z_\mu$.

We expect to be able to obtain μ linearly independent equations from these two sequences.

3.2.1 A Strategy for Faster Equation Generation

There are indeed other ways of generating new linear equations in $z_1, z_2, \ldots, z_\mu$. Let

$$\rho_1 z_1 + \rho_2 z_2 + \cdots + \rho_\mu z_\mu = \gamma \tag{14}$$

be an equation already generated, and let $f(z_1, z_2, \ldots, z_\mu)$ be any $\mathbb{F}_q$-linear combination of the monomials $z_1, z_2, \ldots, z_\mu$. Simplification of the equation

$$(\rho_1 z_1 + \rho_2 z_2 + \cdots + \rho_\mu z_\mu) f(z_1, z_2, \ldots, z_\mu) = \gamma f(z_1, z_2, \ldots, z_\mu)$$

using Eqn (4) again yields a linear equation in $z_1, z_2, \ldots, z_\mu$. The particular choice

$$f(z_1, z_2, \ldots, z_\mu) = z_i$$

with a small degree of z_i leads to a faster generation of a new equation than squaring Eqn (14). Our experiments indicate that we can generate a full-rank system by monomial multiplications and a few squaring operations. Moreover, only Eqn (10) suffices to generate a uniquely solvable linearized system.

3.2.2 Examples

First, consider $t = 2$. In this case, R_x contains only one unknown value y_1y_2, and the equation $R_x = \alpha$ can be immediately solved to obtain y_1y_2.

For $t = 3$, we have 3 unknown monomials y_1y_2, y_1y_3 and y_2y_3. Three independent linear equations are needed to solve for these values.

For $t = 4$, we have 7 unknown monomials y_1y_2, y_1y_3, y_1y_4, y_2y_3, y_2y_4, y_3y_4 and $y_1y_2y_3y_4$. We need seven linearly independent equations in these variables.

Table 1 List of monomials in R_x for batch sizes $t \leqslant 6$

Batch size (t)	**No. of monomials (μ)**	**Monomials** $(z_1, z_2, \ldots, z_\mu)$
2	1	y_1y_2
3	3	y_1y_2, y_1y_3, y_2y_3
4	7	y_1y_2, y_1y_3, y_1y_4, y_2y_3, y_2y_4, y_3y_4, $y_1y_2y_3y_4$
5	15	y_1y_2, y_1y_3, y_1y_4, y_1y_5, y_2y_3, y_2y_4, y_2y_5, y_3y_4, y_3y_5, y_4y_5, $y_1y_2y_3y_4$, $y_1y_2y_3y_5$, $y_1y_2y_4y_5$, $y_1y_3y_4y_5$, $y_2y_3y_4y_5$
6	31	y_1y_2, y_1y_3, y_1y_4, y_1y_5, y_1y_6, y_2y_3, y_2y_4, y_2y_5, y_2y_6, y_3y_4, y_3y_5, y_3y_6, y_4y_5, y_4y_6, y_5y_6, $y_1y_2y_3y_4$, $y_1y_2y_3y_5$, $y_1y_2y_3y_6$, $y_1y_2y_4y_5$, $y_1y_2y_4y_6$, $y_1y_2y_5y_6$, $y_1y_3y_4y_5$, $y_1y_3y_4y_6$, $y_1y_3y_5y_6$, $y_1y_4y_5y_6$, $y_2y_3y_4y_5$, $y_2y_3y_4y_6$, $y_2y_3y_5y_6$, $y_2y_4y_5y_6$, $y_3y_4y_5y_6$, $y_1y_2y_3y_4y_5y_6$

Table 1 lists the even-degree monomials for all the values of t in the range $2 \leqslant t \leqslant 6$. Table 2 proposes possible sequences of monomial multiplication and squaring for generating the linearized systems.

3.3 Retrieving the Unknown y-coordinates

The final step in Algorithm S1 involves the determination of the y-coordinates y_i of the points R_i. Multiplying both sides of Eqn (11) by y_1 gives an equation of the form

$$\beta y_1 = \epsilon_0 + \epsilon_1 z_1 + \epsilon_2 z_2 + \cdots + \epsilon_\mu z_\mu.$$

Substitution of the values of z_i available from the previous stage gives y_1 (provided that $\beta \neq 0$). Subsequently, the values y_i for $i = 2, 3, \ldots, t$ can be obtained by dividing the

known value of $y_1 y_j$ by y_1 provided that $y_1 \neq 0$. Even if $y_1 = 0$, we can multiply Eqn (11) by y_2 to solve for y_2. If $y_2 \neq 0$, we are allowed to compute $y_i = (y_2 y_i)/y_2$ for $i \geqslant 3$. If $y_2 = 0$ too, we compute y_3 by directly using Eqn (11), and so on. The only condition that is necessary to solve for all y_i values uniquely is $\beta \neq 0$, where β is the y-coordinate of the point on the right side of Eqn (2) (or Eqn (3)).

We finally check whether Eqn (4) is valid for all $i = 1, 2, \ldots, t$. If so, all the signatures are verified simultaneously. If one or more of these equations fail(s) to hold, batch verification fails.

3.4 Algorithm S1

Algorithm 9 describes our batch-verification Algorithm S1. In short, Algorithm S1 uniquely reconstructs the points R_i with $x(R_i) = r_i$. The computations do not involve taking modular square roots in $\mathbb{F}_q$. We also avoid computing the points $R'_i = u_i P + v_i Q_i$ needed in individual verification. The final check in Step 13 guarantees that the reconstructed points really lie on the curve. In the next section, we prove that the reconstruction process succeeds with very high probability. Moreover, for small batch sizes, the reconstruction process is efficient. The only cost we have to pay is a loss of our ability to identify individual faulty signatures. When the batch-verification condition of Step 13 fails, we have to repeat the algorithm on sub-batches or resort to individual verifications.

Algorithm 9 ECDSA Batch-Verification Algorithm S1

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

1. Compute $w_i = s_i^{-1} \pmod n$ for all $i = 1, 2, \ldots, t$.
2. Compute $u_i = H(M_i) w_i \pmod n$ for all $i = 1, 2, \ldots, t$.
3. Compute $v_i = r_i w_i \pmod n$ for all $i = 1, 2, \ldots, t$.
4. Compute $R' = (\sum_{i=1}^{t} u_i) P + \sum_{i=1}^{t} v_i Q_i \in E(\mathbb{F}_q)$.
   Club together the points Q_i from same signers during the computation of R'. For example, if all the signatures belong to the same signer, compute R' as $(\sum_{i=1}^{t} u_i) P + (\sum_{i=1}^{t} v_i) Q$.
5. If R' is the point at infinity or if $y(R') = 0$, resort to individual verification, else proceed as follows.
6. Let $R_i = (r_i, y_i)$ with variables y_i for all $i = 1, 2, \ldots, t$.
7. Compute $R = (R_x, R_y) = \sum_{i=1}^{t} R_i$ symbolically using the substitutions $y_i^2 = r_i^3 + a r_i + b$ whenever necessary. The polynomials R_x and R_y are linear with respect to each y_i.
8. Generate equations $R_x = \alpha$ and $R_y = \beta$, where $R' = (\alpha, \beta)$.
9. Create $\mu - 1 = 2^{t-1} - 2$ other equations by repeated squaring of $R_x = \alpha$ or by repeatedly multiplying this equation by even-degree monomials in $y_1, y_2, \ldots, y_t$. Make the substitutions $y_i^2 = r_i^3 + a r_i + b$ whenever needed.
10. Solve the linearized system to find out the values of all even-degree monomials of $y_1, y_2, \ldots, y_t$. If the system is not uniquely solvable, resort to individual verification.
11. Multiply both sides of $R_y = \beta$ by y_1 and solve for y_1.
12. Compute $y_i = y_1 y_i / y_1$ from the monomials $y_1 y_i$ for all $i = 2, 3, \ldots, t$. (If $y_1 = 0$, solve for $y_2, y_3, \ldots, y_t$ as in Step 11.)
13. Accept all the signatures if and only if $y_i^2 = r_i^3 + a r_i + b \pmod q$ for all $i = 1, 2, \ldots, t$.

4 Analysis of Algorithm S1

4.1 Time and Space Complexity

The number of monomials handled during the equation-generation and equation-solving stages is $\mu = 2^{t-1} - 1 = \frac{m}{2} - 1$ which grows exponentially with t. Determination of the Eqns (10) and (11) needs $t-1$ symbolic additions involving rational functions with at most $\Theta(m)$ non-zero terms. Each symbolic addition is followed by at most t uses of Eqn (4). Therefore, the symbolic derivation of R requires $\mathrm{O}(mt^2)$ operations in the field $\mathbb{F}_q$. The subsequent generation of the $\mu \times \mu$ linearized system requires $\mathrm{O}(m^2 t)$ field operations. Finally, Gaussian elimination on an $\mu \times \mu$ system demands $\Theta(m^3)$ field operations. Retrieving individual y_i values calls for $\mathrm{O}(mt^2)$ (usually $\mathrm{O}(mt)$) field operations. The running time of Algorithm S1 is dominated by the linear system-solving stage. Evidently, Algorithm S1 becomes impractical except only for small values of t.

It is worthwhile to investigate the running time of the naive Algorithm N. First, this algorithm needs to compute t modular square roots in the field $\mathbb{F}_q$. Each such square-root computation (for example, by the Tonelli-Shanks algorithm [19]) involves an exponentiation in $\mathbb{F}_q$. Subsequently, one needs to check at most $m = 2^t = 2(\mu + 1)$ conditions, with each check involving the computation of the sum of t points on the curve. Therefore, the total running time of Algorithm N is $\mathrm{O}((\sigma + m)t)$, where σ is the time for computing one square root in $\mathbb{F}_q$. Thus, Algorithm S1 outperforms Algorithm N only in situations where σ is rather large compared to m. This happens typically when the batch size t is small and the field size q is large.

The major memory requirement for Algorithm S1 is the storage of the coefficient matrix of the linearized system. This is a $\mu \times \mu$ matrix, that is, $\Theta(\mu^2) = \Theta(2^{2t})$ field elements need to be stored. The naive method, on the other hand, needs the storage of $\Theta(t)$ square roots (field elements) only.

4.2 Unique Solvability of the Linearized System

In Step 10 of Algorithm 9, we solve a linearized $\mu \times \mu$ system in order to obtain the values of the even-degree monomials in the unknown y-coordinates $y_1, y_2, \ldots, y_t$. Let us call these monomials $z_1, z_2, \ldots, z_\mu$ and the coefficient matrix M. In order that the linearized system is uniquely solvable, we require $\det M \neq 0$. We now investigate how often this condition is satisfied, and also how we can force this condition to hold in most cases.

For a moment, let us treat the x-coordinates $r_1, r_2, \ldots, r_t$ as symbols. But then the failure condition $\det M = 0$ can be rephrased in terms of a multivariate polynomial equation in $r_1, r_2, \ldots, r_t$. Let us denote this equation as $D(r_1, r_2, \ldots, r_t) = 0$. If D is identically zero, then any values of $r_1, r_2, \ldots, r_t$ constitute a root of D. We explain shortly how this situation can be avoided.

Assume that D is not identically zero. Let δ be the maximum degree of each individual r_i in D. In Appendix B, we derive that

$$\delta \leqslant \left(2^{2t+3\lceil \log_2 t \rceil + 2} + 3\right)\left(2^{2^{t-1}-1} - 1\right) \approx 2^{2^{t-1}+2t+3\lceil \log_2 t \rceil + 1}.$$

If we restrict our attention to the values $t \leqslant 6$, we have $\delta \leqslant 2^{54}$. In Appendix C, we show that the maximum number of roots of D is bounded below $t\delta q^{t-1}$. The total number of t-tuples $(r_1, r_2, \ldots, r_t)$ over $\mathbb{F}_q$ is q^t. Therefore, a randomly chosen tuple $(r_1, r_2, \ldots, r_t)$ is a

root of D with probability $\leqslant t\delta q^{t-1}/q^t = t\delta/q$. If we use the inequalities $t \leqslant 6$, $\delta \leqslant 2^{54}$ and $q \geqslant 2^{160}$, we conclude that this probability is less than 2^{-103}. Therefore, if D is not the zero polynomial, we can solve for $z_1, z_2, \ldots, z_\mu$ uniquely with very high probability.

What remains is to propose a way to avoid the condition $D = 0$. We start with any t randomly chosen ECDSA signatures with r-values $r_1, r_2, \ldots, r_t$. We then choose any sequence of squaring and multiplication by z_i (Step 9 in Algorithm 9) in order to arrive at a linear system in $z_1, z_2, \ldots, z_\mu$. If the corresponding coefficient matrix M is not invertible, we discard the chosen sequence of squaring and multiplication. This is because $\det M = 0$ implies that either D is the zero polynomial or the chosen $r_1, r_2, \ldots, r_t$ constitute a root of a non-zero D. The second case is extremely unlikely. With high probability, we, therefore, conclude that the chosen sequence of squaring and multiplication gives $D = 0$ identically. We change the sequence, and repeat the above process until we come across the situation where $r_1, r_2, \ldots, r_t$ do not constitute a root of the non-zero polynomial equation $D(r_1, r_2, \ldots, r_t) = 0$. This implies that D is not identically zero, and randomly chosen $r_1, r_2, \ldots, r_t$ satisfy $D(r_1, r_2, \ldots, r_t) = 0$ with very low probability. We keep this sequence for all future invocations of our batch-verification algorithm, since this will work almost always.

Some sequences of squaring and multiplication, that work for all NIST prime curves are listed in Table 2. In the table, S stands for a squaring step, whereas a monomial (like y_2y_4) stands for multiplication by that monomial. In all these cases, we use only Eqn (10), whereas Eqn (11) is used only for the unique determination of individual y_i values. These sequences depend upon t alone, but not on the NIST curves. For curves other than the NIST curves, this method is expected to work equally well. Indeed, we may consider $D(r_1, r_2, \ldots, r_t)$ as a polynomial in $\mathbb{Z}[r_1, r_2, \ldots, r_t]$. If D is not identically zero, then it is identically zero modulo only a finite number of primes (the common prime divisors of the coefficients of D).

Table 2 Sequences to generate linearized systems for NIST prime curves

t	**Sequence in Step 9 of Algorithm 9**
2	No squaring or multiplication needed
3	y_1y_2, y_1y_3
4	$y_1y_2, y_1y_3, y_1y_4, y_2y_3, y_3y_4, y_1y_4$
5	$y_1y_2, y_1y_3, y_1y_4, y_1y_5, y_2y_3, y_2y_4, y_4y_5, y_1y_2, y_1y_3, y_1y_4, y_1y_5, y_1y_2, y_2y_4, y_2y_3$
6	$y_1y_2, y_1y_3, y_1y_4, y_1y_5, y_1y_6, y_2y_3, y_2y_4, y_2y_5, y_1y_2, y_3y_4, y_3y_5, y_1y_5, y_1y_6, y_1y_2y_3y_6, y_1y_5,$ $y_1y_4, y_1y_3, y_1y_2y_3y_6, y_1y_2, y_1y_3, y_1y_4, y_1y_5, y_2y_5, y_2y_3, S, y_2y_6, y_4y_6, y_3y_6, y_5y_6, y_1y_5$

4.3 Security Analysis

In Algorithm S1, we reconstruct the points R_i with x-coordinates $x(R_i) = r_i$ by forcing the condition $R = \sum_{i=1}^{t} R_i = \sum_{i=1}^{t} R'_i = R'$, where $R'_i = u_iP + v_iQ_i$. Suppose that an adversary too can force the condition $R = R'$. The adversary must also reveal the x-coordinates $r_1, r_2, \ldots, r_t$ as parts of ECDSA signatures. Given these x-coordinates and the condition $R = R'$, there exists (with high probability) a unique solution for the corresponding y-coordinates $y_1, y_2, \ldots, y_t$ of $R_1, R_2, \ldots, R_t$. This solution can be computed by the adversary, for example, using Algorithm S1 (or by taking modular square roots in $\mathbb{F}_q$ as in Algorithm N). So long as t is restricted to small constant values (like $t \leqslant 6$), the adversary

requires only moderate computing resources for determining $y_1, y_2, \ldots, y_t$ uniquely. This implies that although the adversary needs to reveal only the x-coordinates r_i, (s)he essentially *knows* the full points R_i. But these points $R_1, R_2, \ldots, R_t$ satisfy the standard batch-verification condition for ECDSA*. That is, if the adversary can fool Algorithm S1, (s)he can fool the standard ECDSA* batch-verification algorithm too. It, therefore, follows that Algorithm S1 is no less secure than the standard batch-verification algorithm for ECDSA*. Conversely, if an adversary can fool any ECDSA* batch-verification algorithm, (s)he can always fool any ECDSA batch-verification algorithm, since ECDSA signatures are only parts of corresponding ECDSA* signatures. To sum up, Algorithm S1 is as secure as ECDSA* batch verification. For the security of the standard batch-verification algorithm for ECDSA* (or DSA), we refer the reader to [14].

The above argument is based upon the unique solvability of $y_1, y_2, \ldots, y_t$. If this is not the case, our algorithm resorts to individual verification. Moreover, we have argued that this unwelcome situation can be made extremely improbable.

An analysis of the security of Algorithm N is also worth including here. Suppose that an adversary can pass one of the $m = 2^t$ checks in Algorithm N along with disclosing $r_1, r_2, \ldots, r_t$. The correct choices y_i of the square roots of $r_i^3 + ar_i + b$ (that is, those choices corresponding to the successful check) constitute a case of fulfillment of the ECDSA* batch-verification criterion. Consequently, Algorithm N too is as secure as standard ECDSA* batch verification.

4.4 Cases of Failure for Algorithm S1

Our Monte Carlo batch-verification Algorithm S1 may fail for a few reasons. We now argue that these cases of failure are probabilistically very rare.

1. Taking $x_i = r_i$ blindly is a possible cause of failure for Algorithm S1. As discussed earlier, this situation has a very low probability. Furthermore, it is easy to identify when this situation occurs. In case of ambiguity in the values of x_i, we can repeat Algorithm S1 for all possible candidate tuples $(x_1, x_2, \ldots, x_t)$. If the points R_i are randomly chosen in $E(\mathbb{F}_q)$, most of these x_i values are unambiguously available to us, and there should not be many repeated runs (if any) of Algorithm S1. Repeated runs, if necessary, may be avoided, because doing so goes against the expected benefits achievable by batch verification.
2. Although we are able to identify good sequences of squaring and multiplication in Step 9 in order to force the determinant polynomial $D(r_1, r_2, \ldots, r_t)$ to be not identically zero, roots of this polynomial may appear in some cases of ECDSA signatures. We have seen that if $r_1, r_2, \ldots, r_t$ are randomly chosen, the probability of this situation is no more than 2^{-103}. However, an adversary may supply roots of D as $r_1, r_2, \ldots, r_t$, thereby forcing our algorithm to resort to individual verification. Clearly, the adversary gains nothing in this case, but our algorithm forfeits the desired speedup.
3. The derivation of Eqn (8) is based upon the point-addition formula on the curve E. The doubling formula (corresponding to the case $P_1 = P_2$) has a different value for λ. Point additions of the form $U + (-U)$ cannot also be handled by the addition formula. So long as we work symbolically using the unknown quantities $y_1, y_2, \ldots, y_t$, it is impossible to predict when the two points being added turn out to be equal or opposite. If $R_1, R_2, \ldots, R_t$ are randomly chosen from $E(\mathbb{F}_q)$, the probability of this occurrence is extremely low. However, an attacker may force the case of such degenerate sums as suggested by Bernstein et al. [3]. Failure of batch verification on a collection of signatures

anyway calls for a treatment of the signatures in smaller groups and/or individually. In doing so, one detects the error associated with Algorithm S1. Another alternative is to randomize the batch-verification process (see Section 9).

4. Algorithm S1 fails if R' is the point at infinity or lies on the x-axis ($\beta = 0$). In that case, one should resort to individual verification. For randomly chosen session keys, this case occurs with a very small probability (nearly $4/q$).

5 A More Efficient Batch-Verification Algorithm

The linearization stage in Algorithm S1 (requiring $\mathrm{O}(m^2t)$ field operations) and the subsequent Gaussian-elimination stage (requiring $\mathrm{O}(m^3)$ field operations) are rather costly, m being already an exponential function of the batch size t. Our second symbolic-manipulation algorithm S2 avoids these two stages altogether.

Algorithm S1 uniquely solves for the monomials $z_1, z_2, \ldots, z_\mu$ using the equation $R_x = \alpha$ only. At this point, there are only two possible solutions for the y_i values: $(y_1, y_2, \ldots, y_t)$ and $(-y_1, -y_2, \ldots, -y_t)$. This *sign* ambiguity is eliminated by using the other equation $R_y = \beta$. As mentioned in connection with the security analysis of Algorithm N, the exact determination of these signs is not important. In other words, we would be happy even if we can determine each y_i correctly up to multiplication by ± 1. This, in turn, implies that if we have any multivariate equation (linear in y_i) of the form $uy_i + v = 0$ (where u, v are polynomials in $y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_t$), we do not mind multiplying this equation by $uy_i - v$ so that $\pm y_i$ satisfy $u^2y_i^2 - v^2 = 0$. But $y_i^2 = r_i^3 + ar_i + b$, so we have $u_i^2(r_i^3 + ar_i + b) - v_i^2 = 0$, an equation in which y_i is *eliminated*. This observation leads to Algorithm S2.

5.1 Algorithm S2

Like Algorithm S1, we first symbolically compute $R = \sum_{i=1}^{t} R_i$, and arrive at Eqns (10) and (11). Then, we consider only the multivariate equation $R_x - \alpha = 0$ linear individually in each y_i. We first eliminate y_1, and with substitutions given by Eqn (4) for $i = 2, 3, \ldots, t$, we arrive at a multivariate equation in $y_2, y_3, \ldots, y_t$, again linear in each of these variables. We eliminate y_2 from this equation, and arrive at a multivariate equation in $y_3, y_4, \ldots, y_t$. We repeat this process until all variables $y_1, y_2, \ldots, y_t$ are eliminated. If the polynomial after all these eliminations reduces to zero, we know that the original equation $R_x = \alpha$ is consistent with respect to $y_i^2 = r_i^3 + ar_i + b$ for all $i = 1, 2, \ldots, t$.

We may likewise eliminate $y_1, y_2, \ldots, y_t$ from $R_y - \beta = 0$ too, but this is not necessary, because it suffices to know y_i uniquely up to multiplication by ± 1.

Algorithm 10 summarizes the steps of this improved Algorithm S2.

5.2 Implementation Issues

Some comments on efficient implementations of the elimination stage (Step 10) are now in order. First, we are not using Eqn (11) at all in Algorithm S2. Consequently, it is not necessary to compute the polynomial R_y. However, in the symbolic-computation stage (Step 8), we need to compute all intermediate y-coordinates, since they are needed in the final value of R_x. The computation of only the last y-coordinate R_y may be avoided. Still, this saves quite some amount of effort ($\mathrm{O}(mt)$ field operations, to be precise). This saving does not

Algorithm 10 ECDSA Batch-verification Algorithm S2

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

1. Compute $w_i = s_i^{-1} \pmod n$ for all $i = 1, 2, \ldots, t$.
2. Compute $u_i = H(M_i)w_i \pmod n$ for all $i = 1, 2, \ldots, t$.
3. Compute $v_i = r_i w_i \pmod n$ for all $i = 1, 2, \ldots, t$.
4. Compute $R' = (\sum_{i=1}^{t} u_i)P + \sum_{i=1}^{t} v_i Q_i \in E(\mathbb{F}_q)$.
   Club together the points Q_i from same signers during the computation of R'. For example, if all the signatures belong to the same signer, compute R' as $(\sum_{i=1}^{t} u_i)P + (\sum_{i=1}^{t} v_i)Q$.
5. If R' is the point at infinity or if $y(R') = 0$, resort to individual verification, else proceed as follows.
6. For each $i = 1, 2, \ldots, t$, if $r_i^3 + ar_i + b$ is neither zero nor a quadratic residue modulo q, reject the i-th signature, and remove it from the batch.
7. Let $R_i = (r_i, y_i)$ with variables y_i for all $i = 1, 2, \ldots, t$.
8. Compute $R = (R_x, R_y) = \sum_{i=1}^{t} R_i$ symbolically using the substitutions $y_i^2 = r_i^3 + ar_i + b$ whenever necessary.
9. Let $\phi = R_x - \alpha$.
10. For $i = 1, 2, 3, \ldots, t-1$, eliminate y_i from ϕ as in the following steps:
    If ϕ is identically zero, resort to individual verification.
    Otherwise, write $\phi = uy_i + v$ where u, v are polynomials in $y_{i+1}, \ldots, y_t$.
    If u is the zero polynomial, resort to individual verification.
    Set $\phi = u^2(r_i^3 + ar_i + b) - v^2$.
    Substitute y_j^2 in ϕ by $r_j^3 + ar_j + b$ for $j = i+1, \ldots, t$.
11. Accept all the signatures if and only if $\phi = 0$.

affect the theoretical complexity of Step 8 in the big-Oh notation, but its practical effects are noticeable.

The second issue is that the polynomials u and v computed in Step 10 have some nice properties. Throughout this step, ϕ and v are polynomials with each non-zero term having even degree, whereas u is a polynomial with each non-zero term having odd degree. In particular, when the first $t-2$ y-coordinates are eliminated, we have $\phi = uy_{t-1}y_t + v$ with $u, v \in \mathbb{F}_q$. Elimination of y_{t-1} eliminates y_t too, so an explicit elimination of y_t is not necessary, that is, it suffices to let the loop of Step 10 run for $i = 1, 2, \ldots, t-1$ only.

5.3 Reconstruction of $y_1, y_2, \ldots, y_t$

This step is unnecessary (and is not included) in Algorithm S2, but is explained here for theoretical interest only. Suppose that a set of t ECDSA signatures with the x-coordinates $r_1, r_2, \ldots, r_t$ passes Algorithm S2. We remember all ϕ values of the form $\phi = uy_i + v$ in Step 10. We then work back to obtain two solutions for the y-coordinates $y_1, y_2, \ldots, y_t$. Just before y_{t-1} is eliminated, we have $\phi = uy_{t-1}y_t + v$ with $u, v \in \mathbb{F}_q$. Since ϕ will eventually reduce to zero, some square roots of $r_{t-1}^3 + ar_{t-1} + b$ and $r_t^3 + ar_t + b$ satisfy $\phi = 0$. If y_{t-1}, y_t are two such square roots (computed by the Tonelli-Shanks algorithm [19]), the other solution of $\phi = 0$ at this point is $-y_{t-1}, -y_t$.

Now, let us assume that we have computed two solutions $y_{i+1}, y_{i+2}, \ldots, y_t$ and $-y_{i+1}, -y_{i+2}, \ldots, -y_t$. We now look at the elimination step for y_i, that is, $\phi = uy_i + v$ with $u, v \in \mathbb{F}_q[y_{i+1}, y_{i+2}, \ldots, y_t]$. Since $\phi = 0$ gives $y_i = -v/u$ with v a polynomial with non-zero terms of even degrees and with u a polynomial with non-zero terms of odd degrees, the substitution of the two sets of values of $y_{i+1}, \ldots, y_t$ gives exactly two values $\pm y_i$.

Here, we have assumed that u evaluates to a non-zero value for the computed values of $y_{i+1}, \ldots, y_t$. If, however, we have $u(y_{i+1}, \ldots, y_t) = 0$, we must also have $v(y_{i+1}, \ldots, y_t) = 0$ too, since otherwise we cannot reduce ϕ to zero after the remaining eliminations of $y_{i+1}, \ldots, y_t$. In this case, plugging in any value of y_i^2 allows the algorithm to succeed, that is, the information $y_i^2 = r_i^3 + ar_i + b$ is not utilized during the elimination of y_i. Therefore, if this situation occurs, we resort to individual verification. Since u and v are polynomials of degrees much smaller than q, this situation occurs with vanishingly small probabilities.

Eventually, we compute two solutions $y_1, y_2, \ldots, y_t$ and $-y_1, -y_2, \ldots, -y_t$ of $R_x = \alpha$. The sign ambiguity is eliminated by using the other batch-verification condition $R_y = \beta$, provided that $\beta \neq 0$. The case $\beta = 0$ is handled by Step 5 of Algorithm S2.

The running time of this reconstruction procedure is $\mathrm{O}(mt^2 + \sigma)$. Here, σ is the time for computing a square root in $\mathbb{F}_q$. It follows that this reconstruction procedure is practical in all cases where running the algorithms N and S1 is practical.

The condition $\det M = 0$ (Step 10 of Algorithm 9) was necessary for Algorithm S1 to work. Such a condition is not needed for Algorithms N and S2. Moreover, Steps 11 and 12 of Algorithm S1 (determination of individual y_i values) are cryptographically unimportant, since $R_x = \alpha$ already identifies exactly two solutions for the reconstructed points. If these steps are omitted, the batch-acceptance criterion of Step 13 has to be changed. We compute z_i^2 for all $i = 1, 2, \ldots, \mu$, and match these values against appropriate products of $r_j^3 + ar_j + b$. In fact, it suffices to consider only the monomials z_i of degree 2. Note, however, that the unique determination of all y_i values takes only an insignificant amount of time compared to Steps 7–9 in Algorithm S1. Therefore, it does not practically matter to make a choice between whether we carry out these steps or not.

6 Analysis of Algorithm S2

6.1 Time and Space Complexity

The symbolic computation of (R_x, R_y) involves $\mathrm{O}(mt^2)$ field operations (as in Algorithm S1). Subsequently, we start with the polynomial $\phi = R_x - \alpha$ with at most $\mu + 1 = \frac{m}{2} + 1$ non-zero terms. Elimination of y_i requires computing the squares u^2 and v^2, carrying out the polynomial arithmetic $u^2(r_i^3 + ar_i + b) - v^2$, and $t - i$ substitutions of y_j^2 by $r_j^3 + ar_j + b$. Therefore, the reduction of ϕ too requires $\mathrm{O}(mt^2)$ field operations. This is significantly better than the $\mathrm{O}(m^3)$ operations needed by Algorithm S1. Moreover, Algorithm S2 outperforms Algorithm N for a wide range of t and q, since the condition $(\sigma + m)t \gg mt^2$ is more often satisfied than the condition $(\sigma + m)t \gg m^3$.

For Algorithm S2, the major storage requirement is that for the polynomial ϕ. This multivariate polynomial has $m/2$ non-zero terms, so the space complexity is that of $\Theta(m) = \Theta(2^t)$ field elements.

6.2 Security Analysis

We establish the equivalence between the security of Algorithm S2 and the security of standard ECDSA* batch verification, as we have done for the earlier algorithms (N and S1). Suppose that an adversary reveals the x-coordinates $r_1, r_2, \ldots, r_t$ in ECDSA signatures which pass the batch-verification procedure of Algorithm S2. The procedure described in

Section 5.3 indicates that there are exactly two solutions

$$(y_1, y_2, \ldots, y_t) \text{ and } (-y_1, -y_2, \ldots, -y_t)$$

consistent with $\phi = 0$ (Step 9 of Algorithm S2) and $y_i^2 = r_i^3 + ar_i + b$ for $i = 1, 2, \ldots, t$. One of these solutions corresponds to the ECDSA* signatures based upon the disclosed values $r_1, r_2, \ldots, r_t$. It is that solution that would pass $R_y = \beta$. To sum up, the adversary can forge the standard ECDSA* batch-verification algorithm. Moreover, this forging procedure which essentially involves the unique reconstruction of the points $R_i = (r_i, y_i)$ is practical for any adversary with only a moderate amount of computing resources, so long as t is restricted to small values (the only cases where we can apply S2).

6.3 Cases of Failure for Algorithm S2

Algorithm S2 may fail for a variety of reasons. Most of these reasons are identical to those associated with Algorithm S1 (see points 1, 3 and 4 in Section 4.4). However, unlike Algorithm S1, Algorithm S2 does not generate or solve the linearized system. It instead uses a separate elimination idea. Here, we require ϕ to be never identically equal to zero before all variables $y_1, y_2, \ldots, y_t$ are eliminated. This is motivated by that we require all of the t y-coordinates to play active roles in the elimination phase. If p_i denotes the probability that ϕ becomes identically zero during the elimination of y_i, then the probability of failure of Algorithm S2 inside the loop of Step 10 is

$$\begin{aligned} & p_1 + (1-p_1)p_2 + (1-p_1)(1-p_2)p_3 + \cdots + (1-p_1)(1-p_2)\cdots(1-p_{t-2})p_{t-1} \\ \approx\ & p_1 + p_2 + \cdots + p_{t-1} \\ \approx\ & p_{t-1} \end{aligned}$$

(recall that y_{t-1} and y_t are eliminated together). In Appendix D, we show that for $q \geqslant 2^{160}$ and $t \leqslant 8$, this failure probability is $\leqslant 2^{-246}$. Algorithm S2 can be randomized exactly like Algorithm S1 (see Section 9).

7 Efficient Variants of S1 and S2

In Algorithm S1, we generate a system of linearized equations in $\frac{m}{2} - 1 = 2^{t-1} - 1$ monomials (Steps 7–9). The resulting equation is solved in Step 10 which turns out to be the costliest step of Algorithm S1, demanding $\Theta(m^3)$ field operations.

In Algorithm S2, on the other hand, Step 8 turns out to be the most time-consuming step. This step calls for $\Theta(mt^2)$ field operations. Step 10 too calls for $\Theta(mt^2)$ operations. Any improvement in these steps speeds up Algorithm S2.

In this section, we explain a strategy to reduce the number of monomials in Algorithms S1 and S2. So far, we have been symbolically computing the point $R = \sum_{i=1}^{t} R_i$, and equating the symbolic sum to $R' = (\alpha, \beta)$. This results in polynomial expressions with $\Theta(2^{t-1})$ (that is, $\Theta(m)$) non-zero terms.

Now, let $\tau = \lceil t/2 \rceil$. We symbolically compute the two sums:

$$R^{(1)} = \sum_{i=1}^{\tau} R_i \ \text{ and } \ R^{(2)} = R' - \sum_{i=\tau+1}^{t} R_i. \tag{15}$$

The polynomial expressions involved in $R^{(1)}$ and $R^{(2)}$ contain only $\Theta(2^\tau)$, that is, $\Theta(\sqrt{m})$ non-zero terms. Computing these two symbolic sums, therefore, needs $\Theta(2^\tau \tau^2)$, that is, $\Theta(\sqrt{m}t^2)$ field operations which is significantly smaller than the $\Theta(mt^2)$ operations associated with the symbolic computation of the complete sum $\sum_{i=1}^{t} R_i$. The condition $R = R'$ is equivalent to the condition $R^{(1)} = R^{(2)}$. Using this new condition helps us in speeding up the subsequent steps too.

The symbolic sum $R^{(1)}$ can be computed using Algorithm 8. For computing $R^{(2)}$, one may first compute the symbolic sum $\sum_{i=\tau+1}^{t} R_i$ by Algorithm 8, and subsequently add the opposite of this point to the explicit point $R = (\alpha, \beta) \in E(\mathbb{F}_q)$. This symbolic addition involves denominator clearing and substitutions (4) for $i = \tau+1, \ldots, t$. A faster approach is to compute $R^{(2)}$ as $R' + \sum_{i=\tau+1}^{t}(-R_i) = R' + \sum_{i=\tau+1}^{t}(r_i, -y_i)$. This situation is similar to the case of Algorithm 8 with two modifications. For the first summand, the y-coordinate is explicitly available as an element of $\mathbb{F}_q$. Consequently, this y-coordinate is not considered in the denominator-clearing procedure of Algorithm 7. For the other summands, the elliptic-curve point being added is $(r_i, -y_i)$ instead of (r_i, y_i). Therefore, Step 1 of Algorithm 8 should return $(r_j, -y_j)$.

7.1 Algorithm S1′

Step 7 of Algorithm S1 can be replaced by the two symbolic additions given by Eqn (15). In that case, we replace Step 8 by the initial generation of two equations $x(R^{(1)}) = x(R^{(2)})$ and $y(R^{(1)}) = y(R^{(2)})$. It is easy to argue that $x(R^{(1)})$ is a polynomial in $y_1, y_2, \ldots, y_\tau$ with each non-zero term having even degree, whereas $y(R^{(1)})$ is a polynomial in $y_1, y_2, \ldots, y_\tau$ with each non-zero term having odd degree. That is, the number of non-zero terms in these two expressions is $2^{\tau-1} = \frac{\sqrt{m}}{2}$. However, the presence of $R' = (\alpha, \beta)$ on the right side of the expression for $R^{(2)}$ (Eqn 15) lets both $x(R^{(2)})$ and $y(R^{(2)})$ contain all (square-free) monomials in $y_{\tau+1}, y_{\tau+2}, \ldots, y_t$ (both even and odd degrees). There are exactly $2^{\lfloor t/2 \rfloor} - 1 \leqslant \sqrt{m} - 1$ monomials in these two expressions. In the linearized system that we subsequently generate, we consider, as variables, only the even-degree monomials in $y_1, y_2, \ldots, y_\tau$ and all monomials in $y_{\tau+1}, y_{\tau+2}, \ldots, y_t$. To start with, we have one equation $x(R^{(1)}) = x(R^{(2)})$ in these $\Theta(\sqrt{m})$ monomials.

Subsequently, we keep on squaring the equation $x(R^{(1)}) = x(R^{(2)})$ (and substituting values of y_i^2 wherever necessary). This sequence does not increase the number of monomials in the linearized equations. More precisely, for any $j \geqslant 0$, the equation $x(R^{(1)})^{2^j} = x(R^{(2)})^{2^j}$ contains only the $\Theta(\sqrt{m})$ monomials with which we start. If we fail to obtain a linearized system of full rank, we start squaring the other initial equation $y(R^{(1)}) = y(R^{(2)})$. For any $j \geqslant 1$, the equation $y(R^{(1)})^{2^j} = y(R^{(2)})^{2^j}$ again contains only the monomials with which we start. In all the cases studied, we have been able to obtain a full-rank linearized system by squaring the two initial equations. Since the number of linearized variables in $\Theta(\sqrt{m})$, the total cost of Step 9 of Algorithm S1 now reduces to $\mathrm{O}(mt)$ field operations. Finally, in Step 10, we solve a system with $\Theta(\sqrt{m})$ variables. This step calls for $\Theta(m^{3/2})$ field operations.

To sum up, using the trick introduced in this section decreases the number of field operations from $\Theta(m^3)$ to $\Theta(m^{3/2})$. Let us plan to call this efficient variant of S1 as S1′. Fundamentally, S1′ is not a different algorithm from S1. In particular, the security of S1′ is the same as the security of S1 (in fact, little better, because fewer linearized equations are

involved). However, the reduction in the running time is very significant, both theoretically and practically. The space complexity too improves from $\Theta(m^2)$ to $\Theta(m)$.

7.2 Algorithm S2′

Instead of starting with $\phi = R_x - \alpha$ (Step 9 of Algorithm S2), we start with the initial expression

$$\phi = x(R^{(1)}) - x(R^{(2)}). \tag{16}$$

We then repeatedly eliminate $y_1, y_2, \ldots, y_t$ as in Step 10. Although the initial expression of ϕ contains much less number of monomials than in the original Algorithm S2, elimination of y_1 itself introduces many new monomials in ϕ, that is, soon ϕ becomes almost *full*. Consequently, Step 10 continues to make $\Theta(mt^2)$ field operations as before, that is, the theoretical running time of S2′ is the same as that of S2. The space complexity also remains the same as S2, namely, $\Theta(m)$ field elements. Still, the effects of our heuristic are clearly noticeable in practical implementations.

As described in Section 5.2, the y-coordinates $y(R^{(1)})$ and $y(R^{(2)})$ need not be computed. It is, however, necessary to symbolically compute the y-coordinates of all intermediate sums.

This variant of Algorithm S2, in which Step 8 is replaced by the computations given by Eqn (15), and Step 9 is replaced by the initialization given by Eqn (16), is denoted as S2′. Notice that S2′ is not fundamentally different from S2. For example, the security of S2′ is identical to that of S2.

7.3 Cases of Failure for Algorithms S1′ and S2′

From the viewpoint of failure analysis, the variants S1′ and S2′ are identical to Algorithms S1 and S2, respectively. The coefficients in the monomials can be treated as polynomials in $r_1, r_2, \ldots, r_t$. For S1′ and S2′, these polynomials have smaller total degrees in r_j's than S1 and S2. Consequently, the failure probabilities arising out of $\det M$ being zero (for S1 and S1′) or of ϕ becoming identically zero before all elimination rounds are smaller for S1′ and S2′ than for S1 and S2. A more precise upper bound on these failure probabilities can be computed for S1′ and S2′. However, since S1 and S2 already enjoy negligibly small failure probabilities, an exact determination of these probabilities for S1′ and S2′ would be of theoretical interests only, and is omitted here.

8 Experimental Results

Our batch-verification algorithms are implemented using the GP/PARI calculator [5]. Our choice of this implementation platform is dictated by the symbolic-computation facilities and an easy user interface provided by the calculator. Simplification of polynomial expressions and rational functions using Eqn (4) has been carried out by our customized functions. Although GP/PARI provides the built-in function `substpol()` for this purpose, using this function leads to a slow implementation, particularly when we are simplifying rational functions. All experiments are carried out in a 2.33 GHz Xeon server running Mandriva Linux Version 2010.1. The GNU C compiler 4.4.3 is used for compiling the GP/PARI calculator. Our version of GP/PARI is 2.3.5.

In Table 3, we have listed the average times for carrying out single scalar multiplications in the NIST elliptic curves over prime fields. When a batch of t signatures is of concern, individual verification calls for $2t$ such scalar multiplications, whereas batch verification involves between 2 and $t + 1$ (both inclusive) scalar multiplications, irrespective of which batch-verification algorithm we use. Table 3 also lists the times for single square-root calculations in the underlying fields.

Table 4 lists the worst-case overheads associated with the three algorithms N, S1 and S2. These overhead figures do not include the scalar-multiplication times. Indeed, Steps 1–4 are common to all the three batch-verification algorithms. Two variants of the naive algorithm N are experimented with. Algorithm 4 is specified as N in the table. If we remove the checking of $m = 2^t$ conditions (at the expense of appending one bit to each ECDSA signature), we arrive at the algorithm indicated as N$'$. The faster variants S1$'$ and S2$'$ of the symbolic-computation algorithms are also implemented. The algorithms S1, S1$'$ and S2 become impractical for batch sizes $t > 6$, so these algorithms are not implemented for $t = 7$ and $t = 8$. Use of randomizers degrades the performance of all the algorithms. However, since the use of randomizers is external to the batch-verification algorithms (more precisely, one needs to compute $x(\xi_i R_i)$ from $x(R_i)$ and ξ_i before invoking any batch-verification algorithm at all), we do not consider the effects of randomizers while comparing different batch-verification algorithms.

Table 4 indicates that Algorithm S1 is not very practical, since the overhead increases rapidly with the batch size. Only for $t \leqslant 4$ and for large fields, Algorithm S1 is more efficient than Algorithm N. However, its improved variant S1$'$ significantly outperforms S1 in all cases of primes and batch sizes. Indeed, S1$'$ outperforms even N in most of these studied cases.

Algorithm S2$'$ is always faster than Algorithm S2, and is more efficient than Algorithm N in all reported cases, whereas Algorithm S2 is more efficient than Algorithm N in all cases for $t \leqslant 4$ and in some cases (large fields) for $t = 5$ and $t = 6$ too. The superiority between Algorithms N and S2 is determined by the relative cost of square-root computations in $\mathbb{F}_q$ and symbolic manipulations in $\mathbb{F}_q[y_1, y_2, \ldots, y_t]$.

All of the batch-verification algorithms N, S1, S1$'$, S2, S2$'$ become impractical beyond some small values of t (since their running times are exponential in t). So, we have restricted our experiments only to the values $2 \leqslant t \leqslant 8$. Algorithm N$'$, on the other hand, does not carry out any effort exponential in the batch size t. Consequently, Algorithm N$'$ eventually outperforms all the other five algorithms for large values of t. Our experiments reveal that for small batch sizes, symbolic computation provides faster alternatives. It is also worthwhile to note here that Algorithm N$'$ makes use of *extra* information. The other five algorithms, on the other hand, are fully compliant with standardized ECDSA signatures.

Table 5 records the speedup values achieved by the six algorithms N, N$'$, S1, S1$'$, S2 and S2$'$. Here, the speedup is computed with respect to individual verification, and incorporates both scalar-multiplication times and batch-verification overheads. The maximum achievable speedup values (t in the case of same signer, and $2t/(t + 1)$ in the case of different signers) are also listed in Table 5, in order to indicate how our batch-verification algorithms compare with the ideal cases. The maximum speedup achieved by our fully ECDSA-compliant algorithms is 6.20 in the case of same signer, and 1.70 in the case of different signers. Both these records are achieved by Algorithm S2$'$ for the curve P-521 and for the batch size $t = 7$.

From Table 5, it is evident that one should use Algorithm S2$'$ if extra information (a bit identifying the correct square root of each $r_i^3 + ar_i + b$) is not available. In this case, the optimal batch size is $t = 7$ (or $t = 6$ if the underlying field is small). If, on the other hand,

Table 3 Timings (ms) for NIST prime curves

	P-192	P-224	P-256	P-384	P-521
Time for Scalar Multiplication (in $E(\mathbb{F}_q)$)	1.82	2.50	3.14	7.33	14.38
Time for Square-root (in $\mathbb{F}_q$)	0.06	0.35	0.09	0.26	0.67

Table 4 Overheads (ms) for different batch-verification algorithms

	Naive (N)							Naive (N′)						
	t							t						
Curve	2	3	4	5	6	7	8	2	3	4	5	6	7	8
P-192	0.18	0.39	0.76	1.57	3.40	7.71	17.00	0.13	0.19	0.26	0.33	0.39	0.46	0.52
P-224	0.81	1.34	2.04	3.29	5.63	10.60	21.50	0.71	1.06	1.42	1.78	2.14	2.49	2.85
P-256	0.24	0.49	0.97	1.95	4.18	9.27	20.85	0.19	0.29	0.38	0.48	0.58	0.68	0.78
P-384	0.66	1.15	1.95	3.51	6.76	13.80	29.90	0.53	0.81	1.08	1.35	1.62	1.90	2.17
P-521	1.66	2.70	4.21	6.73	11.63	21.00	43.10	1.36	2.05	2.74	3.42	4.11	4.80	5.49

	Symbolic (S1)					Symbolic (S1′)				
	t					t				
Curve	2	3	4	5	6	2	3	4	5	6
P-192	0.14	0.57	2.01	8.66	40.50	0.07	0.20	0.70	1.60	4.40
P-224	0.15	0.60	2.10	9.50	45.60	0.07	0.20	0.80	1.80	4.70
P-256	0.16	0.61	2.17	9.78	46.30	0.08	0.21	0.82	1.90	4.90
P-384	0.18	0.74	2.71	12.56	62.10	0.08	0.30	0.90	2.20	6.10
P-521	0.22	0.90	3.45	16.80	88.40	0.12	0.40	1.30	2.90	8.00

	Symbolic (S2)					Symbolic (S2′)						
	t					t						
Curve	2	3	4	5	6	2	3	4	5	6	7	8
P-192	0.07	0.30	0.76	2.39	6.65	0.07	0.11	0.32	0.61	1.14	2.36	5.46
P-224	0.07	0.32	0.84	2.53	7.11	0.07	0.12	0.33	0.64	1.21	2.51	5.91
P-256	0.08	0.32	0.80	2.51	7.08	0.08	0.12	0.33	0.64	1.22	2.52	5.88
P-384	0.09	0.37	0.91	2.85	8.15	0.09	0.14	0.38	0.72	1.41	2.95	7.12
P-521	0.11	0.44	1.07	3.45	10.02	0.11	0.18	0.42	0.95	1.76	3.72	9.26

disambiguating extra bits are appended to ECDSA signatures, one should use S2′ for $t \leqslant 4$ for (curves over) small fields and for $t \leqslant 6$ (or $t \leqslant 7$) for large fields. If the batch size increases beyond these bounds, it is preferable to use Algorithm N′.

9 Randomizing Batch Verification

The occurrence of degenerate sums caused accidentally or by an attacker can be largely eliminated if the batch-verification algorithm is randomized (see [2,3,14]). In the first attack of [3], 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

Table 5 Speedup obtained by different batch-verification algorithms

(Experiment not carried out for cells marked as –)

		Same signer							Different signers						
Curve	t	Ideal	N	N′	S1	S1′	S2	S2′	Ideal	N	N′	S1	S1′	S2	S2′
P-192	2	2.00	1.91	1.94	1.93	1.96	1.96	1.96	1.33	1.29	1.30	1.30	1.32	1.32	1.32
	3	3.00	2.71	2.86	2.59	2.84	2.77	2.91	1.50	1.42	1.46	1.39	1.46	1.44	1.48
	4	4.00	3.31	3.75	2.58	3.35	3.31	3.68	1.60	1.48	1.56	1.31	1.49	1.48	1.55
	5	5.00	3.49	4.62	1.48	3.47	3.02	4.28	1.67	1.46	1.62	0.93	1.45	1.37	1.58
	6	6.00	3.10	5.46	0.49	2.72	2.12	4.57	1.71	1.35	1.67	0.41	1.27	1.13	1.57
	7	7.00	2.24	6.28	–	–	–	4.25	1.75	1.14	1.70	–	–	–	1.51
	8	8.00	1.41	7.07	–	–	–	3.20	1.78	0.87	1.73	–	–	–	1.33
P-224	2	2.00	1.72	1.75	1.94	1.97	1.97	1.97	1.33	1.20	1.22	1.31	1.32	1.32	1.32
	3	3.00	2.37	2.48	2.68	2.88	2.82	2.93	1.50	1.32	1.36	1.42	1.47	1.45	1.48
	4	4.00	2.84	3.12	2.82	3.45	3.42	3.75	1.60	1.38	1.44	1.37	1.50	1.50	1.56
	5	5.00	3.02	3.70	1.72	3.68	3.32	4.43	1.67	1.37	1.49	1.02	1.49	1.43	1.60
	6	6.00	2.82	4.23	0.59	3.09	2.48	4.83	1.71	1.30	1.53	0.48	1.35	1.22	1.60
	7	7.00	2.24	4.70	–	–	–	4.66	1.75	1.14	1.56	–	–	–	1.55
	8	8.00	1.51	5.13	–	–	–	3.67	1.78	0.91	1.58	–	–	–	1.41
P-256	2	2.00	1.93	1.94	1.95	1.97	1.97	1.97	1.33	1.30	1.31	1.31	1.32	1.32	1.32
	3	3.00	2.78	2.88	2.73	2.90	2.85	2.94	1.50	1.44	1.47	1.43	1.48	1.46	1.49
	4	4.00	3.46	3.78	2.97	3.54	3.55	3.80	1.60	1.51	1.56	1.41	1.52	1.52	1.57
	5	5.00	3.82	4.67	1.96	3.84	3.57	4.54	1.67	1.51	1.63	1.10	1.51	1.47	1.61
	6	6.00	3.60	5.52	0.72	3.37	2.82	5.02	1.71	1.44	1.67	0.55	1.40	1.30	1.62
	7	7.00	2.83	6.36	–	–	–	5.00	1.75	1.28	1.71	–	–	–	1.59
	8	8.00	1.85	7.18	–	–	–	4.13	1.78	1.02	1.73	–	–	–	1.47
P-384	2	2.00	1.91	1.93	1.98	1.99	1.99	1.99	1.33	1.29	1.30	1.32	1.33	1.33	1.33
	3	3.00	2.78	2.85	2.86	2.94	2.93	2.97	1.50	1.44	1.46	1.46	1.48	1.48	1.49
	4	4.00	3.53	3.74	3.38	3.77	3.77	3.90	1.60	1.52	1.56	1.49	1.56	1.56	1.58
	5	5.00	4.03	4.59	2.69	4.35	4.19	4.77	1.67	1.54	1.62	1.30	1.59	1.57	1.64
	6	6.00	4.11	5.42	1.15	4.24	3.86	5.47	1.71	1.51	1.66	0.78	1.53	1.48	1.67
	7	7.00	3.61	6.23	–	–	–	5.83	1.75	1.42	1.70	–	–	–	1.67
	8	8.00	2.63	7.01	–	–	–	5.38	1.78	1.22	1.72	–	–	–	1.60
P-521	2	2.00	1.89	1.91	1.98	1.99	1.99	1.99	1.33	1.28	1.29	1.33	1.33	1.33	1.33
	3	3.00	2.74	2.80	2.91	2.96	2.95	2.98	1.50	1.43	1.45	1.48	1.49	1.49	1.50
	4	4.00	3.49	3.66	3.57	3.83	3.86	3.94	1.60	1.51	1.54	1.53	1.57	1.58	1.59
	5	5.00	4.05	4.48	3.16	4.54	4.46	4.84	1.67	1.55	1.60	1.40	1.61	1.60	1.65
	6	6.00	4.27	5.26	1.47	4.69	4.45	5.65	1.71	1.54	1.65	0.91	1.59	1.56	1.68
	7	7.00	4.05	6.02	–	–	–	6.20	1.75	1.48	1.68	–	–	–	1.70
	8	8.00	3.20	6.74	–	–	–	6.05	1.78	1.33	1.71	–	–	–	1.66

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.

Let $\xi_1, \xi_2, \ldots, \xi_t$ be randomly chosen multipliers. Instead of symbolically manipulating Eqns (2) and (3), we now need to manipulate the weighted sums

$$\sum_{i=1}^{t} \xi_i R_i = \left(\sum_{i=1}^{t} \xi_i u_i \right) P + \sum_{i=1}^{t} \xi_i v_i Q_i \tag{17}$$

or

$$\sum_{i=1}^{t} \xi_i R_i = \left(\sum_{i=1}^{t} \xi_i u_i \right) P + \left(\sum_{i=1}^{t} \xi_i v_i \right) Q. \tag{18}$$

The right sides of these equations do not pose any problem. But only the x-coordinates of R_i are available. However, this information is enough to compute the x-coordinates of $\xi_i R_i$

uniquely. If the x-coordinates of $\xi_i R_i$ are available, we can apply our symbolic-computation algorithms exactly as we have done for the unweighted sum of the points R_i. Two ways of computing $x(\xi_i R_i)$ from $x(R_i)$ are described in [12]. One of these techniques uses Montgomery ladders, and the other a form of (semi)numeric computation. In this paper, we do not make a detailed discussion of these randomization algorithms. We instead concentrate on the experimental effects of randomization on the performance of our fastest batch-verification algorithm S2$'$.

It is easy to see that for l-bit randomizers, the probability of successful attacks against batch verification is 2^{-l}. The length l is chosen to make a tradeoff between security and performance. 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 [2], 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.

We have implemented windowed, w-NAF and frac-w-NAF variants of the seminumeric method of [12]. We have used affine and projective (standard or Jacobian) coordinates. We report the results obtained by the fastest randomization methods.

Table 6 Speedup for NIST Prime Curves

Batch	P-192		P-224		P-256	
Size(t)	None*	$l = 96$	None	$l = 112$	None	$l = 128$
3	2.95	1.39	2.95	1.45	2.95	1.40
4	3.82	1.56	3.84	1.63	3.85	1.57
5	4.53	1.67	4.59	1.76	4.62	1.69
6	5.04	1.73	5.16	1.83	5.23	1.76
7	4.95	1.72	5.15	1.83	5.33	1.77
8	4.02	1.59	4.30	1.71	4.57	1.68

*Without randomization

Batch	P-384			P-521		
Size(t)	None*	$l = 128$	$l = 192$	None	$l = 128$	$l = 256$
3	2.96	1.71	1.43	2.96	1.96	1.45
4	3.89	1.98	1.61	3.91	2.33	1.64
5	4.73	2.18	1.74	4.80	2.62	1.78
6	5.44	2.32	1.83	5.58	2.83	1.88
7	5.76	2.38	1.86	6.06	2.95	1.93
8	5.28	2.29	1.81	5.80	2.89	1.90

*Without randomization

Table 6 illustrates the performance degradation of the batch-verification Algorithm S2$'$ 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. We have taken two cryptographically meaningful bit-lengths l of randomizers (half-length and 128). Although the increased security provided by randomization incurs reasonable overhead, we still have sizable speedup over individual verification. The algorithms S1, S2 and S1$'$ would provide little performance gain, if any at all, when used in conjunction with randomization. Likewise, the case of multiple signers too is severely affected by randomization. Therefore, we do not present experimental data for these cases.

10 Batch Verification on Koblitz Curves

So far, we have concentrated on elliptic-curve parameters with cofactor $h = 1$. If $h > 1$, the disclosure of r in an ECDSA signature leaves multiple possibilities for the x-coordinate of the point $R = kP$ (See Algorithm 2). Running a batch-verification algorithm for all these possibilities is not a practical solution. However, if h is not too large, one can append a few bits to a standard ECDSA signature in order to identify the correct x-coordinate from the published value of r. This leads to a slight loss of conformity with the ECDSA standard. However, under the assumption that we *know* the exact x-coordinates of the points R, we can apply our batch-verification algorithms *mutatis mutandis* to these curves as well. Henceforth, we assume, with a slight abuse of notation, that r itself stands for the x-coordinate of the point R (before conversion to an integer and reduction modulo n). In this section, we report our experience with the NIST Koblitz curves [15].

10.1 Koblitz Curves

A Koblitz curve is defined over a binary field $\mathbb{F}_{2^l}$ by the equation

$$y^2 + xy = x^3 + ax^2 + 1. \tag{19}$$

Here, a is either 0 or 1. If $a = 0$, the cofactor is $h = 4$, whereas $h = 2$ if $a = 1$. The NIST standard lists five such curves K-163, K-233, K-283, K-409 and K-571, where the number after K- indicates the size of the field elements (that is, the extension degree l). The standard also specifies how the arithmetic of each such field $\mathbb{F}_{2^l}$ is to be implemented.

The Koblitz curves, being defined over fields of characteristic two, are subtly different from the prime curves. We can no longer use the simplified Weierstrass equation used for curves defined over large prime fields. Indeed, the two y-coordinates of points with a given x-coordinate r now satisfy the equation

$$y^2 + ry + (r^3 + ar^2 + 1) = 0. \tag{20}$$

The sum of the two roots is r, and their product is $r^3 + ar^2 + 1$. This calls for suitable changes in the batch-verification algorithms described so far, since they are based upon the simplified Weierstrass equation.

The NIST standard also specifies a set of random elliptic curves defined over the above five binary fields. They all correspond to the cofactor value $h = 2$. Adaptations of our batch-verification algorithms to these curves are straightforward, given those for Koblitz curves. Consequently, we do not discuss this random family in the rest of this paper. Indeed, following the adaptations to Koblitz curves, one can easily modify our algorithms to work for any non-supersingular elliptic curve defined over $\mathbb{F}_{2^l}$.

10.2 Adaptation of the Naive Algorithms

The basic computational task involved in the algorithms N and N′ was the computation of square roots in the underlying field. In the case of binary fields, we need to solve the quadratic equation (20) both the roots of which lie in the field. We can use a root-finding or polynomial-factoring algorithm for computing these two roots, such as Berlekamp's trace

algorithm or its randomized variant as described in Menezes et al. [13]. For quadratic polynomials over $\mathbb{F}_{2^l}$, these algorithms take $O(l^3)$ expected running time—theoretically the same as the Tonelli-Shanks algorithm. In practice, however, these algorithms are significantly slower than modular square-root computations in prime fields of comparable sizes. The implication of this is that the naive algorithms are rather inefficient for fields of characteristic two, and consequently, our symbolic-computation algorithms exhibit noticeably better performances than the naive algorithms.

10.3 Adaptation of the Symbolic-computation Algorithms

The symbolic-computation algorithms S1, S1′, S2 and S2′ require modifications in view of the changed equation for y in terms of r. In all these algorithms, we can keep each y_i-degree below 2 by making repeated substitutions $y_i^2 = r_i y_i + (r_i^3 + ar_i^2 + 1)$. The introduction of the term $r_i y_i$ creates some trouble with the parity of the degrees of the terms. The sequence of linearized equations generated in Steps 7–9 of Algorithm S1 now involves also the odd-degree monomials in $y_1, y_2, \ldots, y_t$, that is, the count of linearized variables increases from $2^{t-1} - 1$ to $2^t - 1$. This results in a degradation of the performance of S1.

In both Algorithms S1 (Step 7) and S2 (Steps 8 and 10), we need to eliminate y_i from a multivariate polynomial linear with respect to y_i. Let us write such a polynomial as

$$\phi = uy_i + v,$$

where u and v are multivariate polynomials not containing y_i. Multiplying ϕ by $u(y_i+r_i)+v$ gives

$$\begin{aligned}(u(y_i + r_i) + v)\phi \;&=\; (u(y_i + r_i) + v)(uy_i + v) \;=\; u^2(y_i^2 + r_i y_i) + uvr_i + v^2 \\ &= u^2(r_i^3 + ar_i^2 + 1) + uvr_i + v^2.\end{aligned}$$

The last expression is free from y_i. The polynomials u^2, uv, v^2 may contain y_j^2 for $j \neq i$. Each such occurrence of y_j^2 is to be substituted by $r_j y_j + (r_j^3 + ar_j^2 + 1)$ in order to reduce the y_j-degree of the expression to below 2. Algorithms S2 and S2′ do not deal with linearized systems, but still experience an increase in the count of non-zero terms from a maximum of 2^{t-1} to a maximum of 2^t. This reduces the performance benefits of S2 and S2′, but this degradation is much more graceful than for S1. The variant S1′ is also less affected than S1.

10.4 Experimental Results

We continued our experiments with GP/PARI 2.3.5. It seems that the GP/PARI routines for elliptic curves over binary fields $\mathbb{F}_{2^l}$ are not very optimized. Table 7 indicates that elliptic-curve scalar-multiplication times for Koblitz curves are over three orders of magnitude slower than those for prime curves (Table 3) for fields of comparable sizes. A more critical difference is in the times for solving quadratic equations over the underlying fields. For prime curves, the modular square-root algorithm is over 20 times faster than individual scalar multiplication. For Koblitz curves, we use the GP/PARI function `factorff` which requires comparable (even larger) running times relative to individual scalar multiplication.

Table 8 lists the worst-case overheads (excluding the computation of the point $R = (\alpha, \beta)$) associated with the five batch-verification algorithms studied. Both the variants of

Table 7 Timings (sec) for NIST Koblitz curves

	K-163	K-233	K-283	K-409	K-571
Time for Scalar Multiplication (in $E(\mathbb{F}_{2^l})$)	2.57	6.97	12.00	36.16	94.17
Time for `factorff` (over $\mathbb{F}_{2^l}$)	1.78	6.41	12.96	49.30	165.67

Table 8 Overheads (sec) for different batch-verification algorithms

	Naive (N)						Naive (N′)					
	t						t					
Curve	2	3	4	5	6	7	2	3	4	5	6	7
K-163	3.57	5.47	7.60	10.17	13.85	20.07	3.56	5.33	7.12	8.90	10.74	12.48
K-233	12.89	19.51	26.55	34.47	44.61	59.45	12.82	19.26	25.72	32.11	38.52	45.14
K-283	26.21	39.85	53.17	68.42	86.64	111.85	26.17	38.97	52.00	65.08	77.88	91.13
K-409	99.27	149.31	199.78	253.62	313.77	388.08	98.56	148.41	197.75	248.54	297.92	345.14
K-571	333.4	500.8	668.1	842.7	1028.9	1241.8	332.0	500.5	664.5	830.0	1017.5	1162.6

	Symbolic (S1′)					Symbolic (S2)					Symbolic (S2′)					
	t					t					t					
Curve	2	3	4	5	6	2	3	4	5	6	2	3	4	5	6	7
K-163	0.04	0.26	0.88	3.64	8.52	0.03	0.28	1.20	4.64	16.95	0.03	0.07	0.21	0.66	1.65	5.73
K-233	0.08	0.48	1.59	6.77	15.51	0.06	0.48	2.09	8.23	30.97	0.06	0.12	0.37	1.17	2.94	10.22
K-283	0.11	0.68	2.28	9.36	21.58	0.08	0.66	2.88	11.25	42.92	0.08	0.17	0.52	1.59	4.05	14.17
K-409	0.23	1.29	4.28	18.01	40.83	0.14	1.20	5.16	20.21	73.33	0.14	0.31	0.95	2.90	7.23	25.71
K-571	0.40	2.36	7.82	31.75	72.50	0.25	2.04	8.80	34.88	127.1	0.25	0.57	1.65	4.93	12.28	44.21

Table 9 Speedup obtained by different batch-verification algorithms

		Same signer						Different signers					
Curve	t	Ideal	N	N′	S1′	S2	S2′	Ideal	N	N′	S1′	S2	S2′
K-163	2	2.00	1.18	1.18	1.98	1.99	1.99	1.33	0.91	0.91	1.33	1.33	1.33
	3	3.00	1.45	1.47	2.86	2.85	2.96	1.50	0.98	0.99	1.46	1.46	1.49
	4	4.00	1.61	1.68	3.42	3.24	3.84	1.60	1.01	1.03	1.50	1.46	1.57
	5	5.00	1.68	1.83	2.93	2.63	4.43	1.67	1.00	1.06	1.35	1.28	1.60
	6	6.00	1.62	1.94	2.26	1.40	4.54	1.71	0.97	1.07	1.16	0.88	1.57
	7	7.00	1.43	2.04	–	–	3.31	1.75	0.89	1.09	–	–	1.37
K-233	2	2.00	1.04	1.04	1.99	1.99	1.99	1.33	0.82	0.83	1.33	1.33	1.33
	3	3.00	1.25	1.26	2.90	2.90	2.97	1.50	0.88	0.89	1.47	1.47	1.49
	4	4.00	1.38	1.41	3.59	3.48	3.90	1.60	0.91	0.92	1.53	1.51	1.58
	5	5.00	1.44	1.51	3.37	3.14	4.61	1.67	0.91	0.94	1.43	1.39	1.62
	6	6.00	1.43	1.59	2.84	1.86	4.95	1.71	0.90	0.96	1.30	1.05	1.62
	7	7.00	1.33	1.65	–	–	4.04	1.75	0.85	0.97	–	–	1.48
K-283	2	2.00	0.96	0.96	1.99	1.99	1.99	1.33	0.77	0.77	1.33	1.33	1.33
	3	3.00	1.13	1.14	2.92	2.92	2.98	1.50	0.82	0.83	1.48	1.48	1.49
	4	4.00	1.24	1.26	3.65	3.57	3.92	1.60	0.85	0.86	1.54	1.53	1.59
	5	5.00	1.30	1.35	3.60	3.40	4.69	1.67	0.85	0.88	1.47	1.44	1.63
	6	6.00	1.30	1.41	3.16	2.15	5.13	1.71	0.84	0.89	1.36	1.13	1.64
	7	7.00	1.24	1.46	–	–	4.40	1.75	0.81	0.90	–	–	1.52
K-409	2	2.00	0.84	0.85	1.99	2.00	2.00	1.33	0.70	0.70	1.33	1.33	1.33
	3	3.00	0.98	0.98	2.95	2.95	2.99	1.50	0.74	0.74	1.49	1.49	1.50
	4	4.00	1.06	1.07	3.78	3.73	3.95	1.60	0.76	0.76	1.56	1.56	1.59
	5	5.00	1.11	1.13	4.00	3.91	4.81	1.67	0.77	0.78	1.54	1.52	1.64
	6	6.00	1.12	1.17	3.83	2.98	5.45	1.71	0.77	0.79	1.48	1.33	1.67
	7	7.00	1.10	1.21	–	–	5.16	1.75	0.75	0.80	–	–	1.61
K-571	2	2.00	0.72	0.72	2.00	2.00	2.00	1.33	0.61	0.61	1.33	1.33	1.33
	3	3.00	0.82	0.82	2.96	2.97	2.99	1.50	0.64	0.64	1.49	1.49	1.50
	4	4.00	0.88	0.88	3.84	3.82	3.97	1.60	0.66	0.66	1.57	1.57	1.59
	5	5.00	0.91	0.92	4.28	4.22	4.87	1.67	0.67	0.68	1.58	1.57	1.65
	6	6.00	0.93	0.94	4.33	3.58	5.63	1.71	0.67	0.67	1.54	1.44	1.68
	7	7.00	0.92	0.98	–	–	5.67	1.75	0.66	0.69	–	–	1.65

the naive algorithm are crippled by huge running times taken by `factorff`. The symbolic-computation algorithms do not involve this function and perform significantly better than the naive algorithms. Since Algorithm S1 involves generating and solving large linear systems, we have not implemented it. Its efficient variant S1$'$ is only implemented. Once again, randomizers are not considered in these running times, because they do not show up in the relative performance of the different batch-verification algorithms.

Speedup figures over individual verification are listed in Table 9. The maximum speedup achieved by the naive algorithms is 2.04 for the case of the same signer and 1.09 for the case of different signers, indicating that these algorithms are not effective at all for Koblitz curves. The symbolic-computation algorithms, on the other hand, exhibit a similar pattern for Koblitz curves, as they have done for prime curves. The maximum recorded speedup is achieved always by S2$'$. For the curve K-571 and for the batch size $t = 7$, this is 5.67 in the case of the same signer, and 1.65 in the case of different signers. Even for small fields, S2 and S2$'$ exhibit decent performance. The performance of S1$'$ is between those of S2 and S2$'$.

10.5 Randomization for Koblitz Curves

As mentioned in Section 9, we have chosen the bit-lengths l of randomizes to be 128 and $d/2$. The seminumeric algorithm is used to compute the x-coordinate of ξR from the randomizer ξ and the the x-coordinate of the point R. For Koblitz curves, the τ-NAF variant of the seminumeric method of [12] is the fastest randomization method. In the τ-NAF method, the lengths of the addition chains are nearly equal for half- and full-length randomizers. This causes a substantial overhead in the randomization procedure, and the resulting performance gains are much lower than those for prime fields. Table 10 lists all the speedup figures obtained using Algorithm S2$'$ in the case of the same signer.

Table 10 Speedup for NIST Koblitz Curves

Batch	K-163		K-233		K-283	
Size(t)	None*	$l = 80$	None	$l = 112$	None	$l = 128$
2	1.95	0.86	1.96	0.86	1.97	0.89
3	2.82	0.99	2.87	1.00	2.89	1.04
4	3.48	1.06	3.59	1.08	3.66	1.13
5	3.31	1.05	3.61	1.08	3.78	1.14
6	2.81	0.99	3.22	1.04	3.48	1.11
7	1.50	0.76	1.84	0.84	2.09	0.92

*Without randomization

Batch	K-409			k-571		
Size(t)	None*	$l = 128$	$l = 192$	None	$l = 128$	$l = 256$
2	1.98	1.05	0.86	1.98	1.19	0.87
3	2.92	1.27	1.01	2.94	1.48	1.02
4	3.75	1.41	1.09	3.80	1.67	1.11
5	4.07	1.45	1.11	4.25	1.75	1.14
6	3.96	1.43	1.11	4.28	1.76	1.14
7	2.61	1.21	0.97	3.05	1.51	1.03

*Without randomization

11 Conclusion

In this paper, we have proposed six algorithms for the batch verification of ECDSA signatures. To the best of our knowledge, these are the first batch-verification algorithms ever proposed for ECDSA. In particular, development of algorithms based upon symbolic manipulations appears to be a novel approach in the history of batch-verification algorithms. There are several ways to extend our study, some of which are listed below.

- Section 7 describes a way to reduce the running time of Step 8 of Algorithm S2 from $\mathrm{O}(mt^2)$ to $\mathrm{O}(\sqrt{m}t^2)$. An analogous speedup for Step 10 would be very useful.
- Our best symbolic-computation algorithm runs in $\mathrm{O}(mt^2)$ time. Removal of a factor of t (that is, designing an $\mathrm{O}(mt)$-time algorithm) would be useful to achieve higher speedup values.
- It is also of good research interest to study our algorithms in conjunction with the earlier works [1,4] on ECDSA*.

References

1. Antipa, A., Brown, D., Gallant, R., Lambert, R., Struik, R., Vanstone, S.: Accelerated verification of ECDSA signatures. In: SAC. Lecture Notes in Computer Science, vol. 3897, pp. 307–318. Springer (2006)
2. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 1403, pp. 236–250. Springer (1998)
3. Bernstein, D.J., Doumen, J., Lange, T., Oosterwijk, J.J.: Faster batch forgery identification. In: INDOCRYPT. Lecture Notes in Computer Science, vol. 7668, pp. 454–473. Springer (2012)
4. Cheon, J.H., Yi, J.H.: Fast batch verification of multiple signatures. In: PKC. Lecture Notes in Computer Science, vol. 4450, pp. 442–457. Springer (2007)
5. Cohen, H., Belabas, K.: PARI/GP. `http://pari.math.u-bordeaux.fr/` (2003–2013)
6. Das, A., Choudhury, D.R., Bhattacharya, D., Rajavelu, S., Shorey, R., Thomas, T.: Authentication schemes for VANETs: A survey. International Journal of Vehicle Information and Communication Systems 3(1), 1–27 (2013)
7. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (November 1976)
8. ElGamal, T.: A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31, 469–472 (1985)
9. Harn, L.: Batch verifying multiple RSA digital signatures. Electronics Letters 34(12), 1219–1220 (1998)
10. Hwang, M.S., Lin, I.C., Hwang, K.F.: Cryptanalysis of the batch verifying multiple RSA digital signatures. Informatica 11(1), 15–19 (2000)
11. Johnson, D., Menezes, A.: The elliptic curve digital signature algorithm (ECDSA). Journal on Information Security 1, 36–63 (2001)
12. Karati, S., Das, A., Roychowdhury, D.: Using randomizers for batch verification of ECDSA signatures. Tech. rep., Cryptology ePrint Archive: Report 2012/582 (2012)
13. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Some computational aspects of root finding in GF(q^m). In: ISSAC. Lecture Notes in Computer Science, vol. 358, pp. 259–270. Springer (1989)
14. Naccache, D., M'Raihi, D., Rapheali, D., Vaudenay, S.: Can D.S.A. be improved? Complexity trade-offs with the digital signature standard. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 950, pp. 77–85. Springer (1994)
15. NIST: Recommended elliptic curves for federal government use. `http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf` (1999)
16. NIST: Digital Signature Standard (DSS). `http://csrc.nist.gov/publications/drafts/fips_186-3/Draft-FIPS-186-3%20_March2006.pdf` (2006)
17. NIST: Secure Hash Standard (SHS). `http://csrc.nist.gov/publications/drafts/fips180-3/draft_fips_180-3_June-08-2007.pdf` (2007)
18. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Comm. ACM 21(2), 120–126 (1978)
19. Shanks, D.: Five number theoretic algorithms. In: Proceedings of the Second Manitoba Conference on Numerical Mathematics. pp. 51–70 (1973)

A Properties of R_x and R_y

Theorem 1 *R_x consists of only even-degree monomials, and R_y consists of only odd-degree monomials in the variables $y_1, y_2, \ldots, y_t$.*

Proof: We proceed by induction on the batch size $t \geqslant 1$. If $t = 1$ (this amounts to individual verification), we have $R_x = r_1$ and $R_y = y_1$, for which the theorem evidently holds.

So assume that $t \geqslant 2$. We compute $R = \sum_{i=1}^{t} R_i$ as $R' + R''$ with $R' = \sum_{i=1}^{\tau} R_i$ and $R'' = \sum_{i=\tau+1}^{t} R_i$ for some τ in the range $1 \leqslant \tau \leqslant t-1$. Let $R' = (R'_x, R'_y)$ and $R'' = (R''_x, R''_y)$. The inductive assumption is that all non-zero terms of R'_x and R''_x are of even degrees (in $y_1, \ldots, y_\tau$ and $y_{\tau+1}, \ldots, y_t$, respectively), and all non-zero terms of R'_y and R''_y are of odd degrees.

We first symbolically compute $\lambda = (R''_y - R'_y)/(R''_x - R'_x)$ as a rational function. Clearing the variables y_i from the denominator multiplies both the numerator and the denominator of λ by polynomials of non-zero terms having even degrees. Every substitution of y_i^2 by the field element $r_i^3 + ar_i + b$ reduces the y_i-degree of certain terms by 2, so the parity of the degrees in these terms is not altered. Finally, λ becomes a polynomial with each non-zero term having odd degree. But then, $R_x = \lambda^2 - R'_x - R''_x$ is a polynomial with each non-zero term having even degree, whereas $R_y = \lambda(R'_x - R_x) - R'_y$ is a polynomial with each non-zero term having odd degree. Further substitutions of y_i^2 by $r_i^3 + ar_i + b$ to simplify R_x and R_y preserve these degree properties. •

B Derivation of δ

In order to compute the number of roots $(r_1, r_2, \ldots, r_t)$ of $\det M = 0$, we treat $r_1, r_2, \ldots, r_t$ as symbols, and need to calculate an upper bound on the degree δ of each individual r_i. Without loss of generality, we compute an upper bound on the degree δ of r_1 in $\det M = 0$. To this effect, we first look at the expressions for R_x and R_y which are elements of $\mathbb{F}_q(r_1, r_2, \ldots, r_t)[y_1, y_2, \ldots, y_t]$. We can write $R_x = g_x/h$ and $R_y = g_y/h$, where g_x, g_y are polynomials in $\mathbb{F}_q[r_1, r_2, \ldots, r_t, y_1, y_2, \ldots, y_t]$, and the common denominator h is a polynomial in $\mathbb{F}_q[r_1, r_2, \ldots, r_t]$. Let η_t denote the maximum of the r_1-degrees in g_x, g_y and h. We first recursively derive an upper bound for η_t.

We compute $R = R' + R''$ with $R' = (R'_x, R'_y) = \sum_{i=1}^{\tau} R_i$ and $R'' = (R''_x, R''_y) = \sum_{i=\tau+1}^{t} R_i$, where $\tau = \lceil t/2 \rceil$. The r_1-degree of R' is η_τ, whereas the r_1-degree of R'' is 0. The initial r_1-degree of $\lambda = (R''_y - R'_y)/(R''_x - R'_x)$ is at most η_τ. Clearing y_1 from the denominator of λ changes the r_1-degree to $2\eta_\tau + 3$. Subsequent eliminations of $y_2, \ldots, y_t$ finally reduces λ with a y-free denominator. The maximum r_1-degree of this expression for λ is $2^{t-1}(2\eta_\tau + 3)$. Therefore, λ^2 has r_1-degree no more than $2^t(2\eta_\tau + 3)$. Subsequent computations of $R_x = \lambda^2 - R'_x - R''_x$ and $R_y = \lambda(R'_x - R_x) - R'_y$ indicate that

$$\eta_t \leqslant (2^t + 2^{t-1})(2\eta_\tau + 3) + 2\eta_\tau \leqslant (2^t + 2^{t-1})(2\eta_\tau + 3) + 2\eta_\tau$$

with $\tau = \lceil t/2 \rceil$. Solving this recurrence gives the upper bound $\eta_t \leqslant 2^{2t+3\lceil \log_2 t \rceil + 2}$.

Now, we follow a sequence of squaring and monomial multiplication to convert $R_x = \alpha$ to a set of linear equations. If Δ_i is the r_1-degree of the i-th equation, we have

$$\begin{aligned} \Delta_1 &= \eta_t, \\ \Delta_i &\leqslant 2\Delta_{i-1} + 3 \ \text{ for } \ i \geqslant 2. \end{aligned}$$

The recurrence relation pertains to the case of squaring. One easily checks that $\Delta_i \leqslant (\eta_t + 3)2^{i-1}$ for all $i \geqslant 1$. Finally, we consider $\det M = 0$. The r_1-degree of this equation is

$$\delta \leqslant \Delta_1 + \Delta_2 + \cdots + \Delta_\mu \leqslant (\eta_t + 3)(2^\mu - 1) \leqslant \left(2^{2t+3\lceil \log_2 t \rceil + 2} + 3\right)\left(2^{2^{t-1}-1} - 1\right).$$

Notice that this is potentially a very loose upper bound for δ. In general, we avoid squaring. Multiplication by a monomial can increase the r_1-degree by 3 if the monomial contains y_1. If the monomial does not contain y_1, the r_1-degree does not increase at all. Nevertheless, this loose upper bound is good enough in the present context.

C Number of Roots of $\det M = 0$

Let us write the equation $\det M = 0$ as $D(r_1, r_2, \ldots, r_t) = 0$, where the r_i-degree of the multivariate polynomial D is $\leqslant \delta$ for each i. We assume that D is not identically zero. We plan to show that the maximum number $B^{(t)}$ of roots of D is $\leqslant t\delta q^{t-1}$. To that effect, we first write D as a polynomial in r_t:

$$\begin{aligned} D(r_1, r_2, \ldots, r_t) = {} & D_\delta(r_1, r_2, \ldots, r_{t-1}) r_t^\delta + D_{\delta-1}(r_1, r_2, \ldots, r_{t-1}) r_t^{\delta-1} + \cdots + \\ & D_1(r_1, r_2, \ldots, r_{t-1}) r_t + D_0(r_1, r_2, \ldots, r_{t-1}). \end{aligned}$$

Since D is not identically zero, at least one D_i is not identically zero. If $(r_1, r_2, \ldots, r_{t-1})$ is a common root of each D_i, appending any value of r_t gives a root of D. The maximum number of common roots of $D_0, D_1, \ldots, D_\delta$ is $B^{(t-1)}$. On the other hand, if $(r_1, r_2, \ldots, r_{t-1})$ is not a common root of all D_i, there are at most δ values of r_t satisfying $D(r_1, r_2, \ldots, r_t) = 0$. We, therefore, have

$$B^{(t)} \leqslant B^{(t-1)} q + (q^{t-1} - B^{(t-1)})\delta = (q - \delta) B^{(t-1)} + \delta q^{t-1}. \tag{21}$$

Moreover, we have

$$B^{(1)} \leqslant \delta. \tag{22}$$

By induction on t, one can show that $B^{(t)} \leqslant t\delta q^{t-1}$. This bound is rather tight, particularly for $\delta \ll q$ (as it happens in our cases of interest). A polynomial D satisfying equalities in (21) and (22) can be constructed as $D(r_1, r_2, \ldots, r_t) = \Delta(r_1)\Delta(r_2)\cdots\Delta(r_t)$, where Δ is a square-free univariate polynomial of degree δ, that splits over $\mathbb{F}_q$. By the principle of inclusion and exclusion (or by explicitly solving the recurrence (21)), we obtain the total number of roots of this D as

$$\begin{aligned} & \delta t q^{t-1} - \binom{t}{2}\delta^2 q^{t-1} + \binom{t}{3}\delta^3 q^{t-3} - \cdots + (-1)^{t-1}\delta^t \\ = {} & q^t - (q - \delta)^t \\ = {} & \delta(q^{t-1} + (q-\delta)q^{t-2} + (q-\delta)^2 q^{t-3} + \cdots + (q-\delta)^{t-1}). \end{aligned}$$

If $\delta \ll q$, this count is very close to $t\delta q^{t-1}$. It remains questionable whether our equation $\det M = 0$ actually encounters this worst-case situation, but this does not matter, at least in a probabilistic sense.

D Derivation of the Probabilities p_i

Like Algorithm S1, we first symbolically compute $R = \sum_{i=1}^{t} R_i$, and arrive at Eqns (10) and (11). Then, we set $\phi = R_x - \alpha$ in step 9 of Algorithm S2. If ϕ is identically zero, then for any values of $y_1, y_2, \ldots, y_t$, batch verification succeeds without using the Eqns (4) in the elimination phase at all. This situation occurs if all the coefficients of all the monomials (and also the constant term) in ϕ are zero. We name the monomials (of even total degrees, including that of degree zero) as $z_1, z_2, \ldots, z_{\mu+1}$, where $\mu = 2^{t-1} - 1$. We write this situation as

$$\rho_1 z_1 + \rho_2 z_2 + \cdots + \rho_{\mu+1} z_{\mu+1} = 0. \tag{23}$$

with each $\rho_i = 0$. For the moment, we treat the x-coordinates $r_1, r_2, \ldots, r_t$ as symbols. Each ρ_i in Eqn (23) is a polynomial in $\mathbb{F}_q[r_1, r_2, \ldots, r_t]$. Let δ' be the maximum degree of each individual r_j in each ρ_i. As already derived in Appendix B, δ' is bounded from above by $\Delta_1 = \eta_t \leqslant 2^{2t+3\lceil \log_2 t \rceil + 2}$. If we restrict our attention to the values $t \leqslant 8$, we see that $\delta' \leqslant 2^{27}$. Let the tuple $(r_1, r_2, \ldots, r_t)$ be a root of ρ_i. As in Appendix C, we estimate that there are $\leqslant t\delta' q^{t-1}$ such tuples. The total number of t-tuples over $\mathbb{F}_q$ is q^t. Therefore, a randomly chosen tuple $(r_1, r_2, \ldots, r_t)$ is a root of ρ_i with probability $\leqslant t\delta' q^{t-1}/q^t = t\delta'/q$. Now, the total number of ρ_i's is $\mu + 1$. Therefore, the probability that a randomly chosen tuple $(r_1, r_2, \ldots, r_t)$ over $\mathbb{F}_q$ is a root of all the ρ_i's is $p_1 \leqslant (t\delta'/q)^{\mu+1}$. For $t \leqslant 8$, $\delta' \leqslant 2^{27}$ and $q \geqslant 2^{160}$, we have $p_1 \leqslant 2^{-16510}$.

Even if ϕ is not identically zero at the beginning of the elimination phase, it should never become so before all of $y_1, y_2, \ldots, y_t$ are eliminated. Let p_i denote the probability that ϕ becomes identically zero before the elimination of y_i. We have calculated p_1 above. Here, we calculate p_i for $i = 2, 3, \ldots, t-1$. Let δ'_i be the total degree in all r_j's in ϕ just before the elimination of y_i. We have $\delta'_i = 2\delta'_{i-1} + 3 \approx 2\delta'_i = 2^{t-i}\delta'$. Moreover, at this point, the number of even-degree monomials in $y_i, y_{i+1}, \ldots, y_t$ in ϕ is $2^{t-i} = (\mu+1)/2^{i-1}$. Therefore, like the expression for p_1, we derive that $p_i \leqslant (t\delta'_i/q)^{2^{t-i}} = (t\delta'_i/q)^{\frac{\mu+1}{2^{i-1}}}$.

The probability that ϕ becomes identically zero just before the elimination of y_i, but never earlier, is $(1-p_1)(1-p_2)\cdots(1-p_{i-1})p_i$. Therefore, the probability that ϕ becomes identically zero in any one of the $t-1$ elimination rounds is

$$\pi \leqslant \sum_{i=1}^{t-1} \left[p_i \prod_{j=1}^{i-1} (1-p_j) \right].$$

For practical ranges of parameter values, all p_i are very close to zero, so we can approximate $1-p_i$ by 1, and conclude that

$$\pi \approx \sum_{i=1}^{t-1} p_i.$$

Moreover, p_{t-1} is the most dominating term in the above summation, so we have

$$\pi \approx p_{t-1} \leqslant (t\delta'_{t-1}/q)^2 \approx (t2^{t-2}\delta'/q)^2 \leqslant 2^{6t+8\lceil \log_2 t \rceil + 2}/q^2.$$