Batch Verification of EdDSA Signatures

Sabyasachi Karati, Abhijit Das
Department of Computer Science and Engineering
IIT Kharagpur, India
skarati,abhij@cse.iitkgp.ernet.in

Abstract. In AfricaCrypt 2012 and ACNS 2014, several algorithms are proposed for the batch verification of ECDSA signatures. In this paper, we make a comparative study of these methods for the Edwards curve digital signature algorithm (EdDSA). We describe the adaptation of Algorithms N, N$'$, S2$'$ and SP for EdDSA signatures. The randomization methods are also explained in detail. More precisely, we study seminumeric scalar multiplication and Montgomery ladders during randomization of EdDSA signatures. Each EdDSA signature verification involves a square-root computation. One may instead use an ECDSA-like verification procedure which avoids the expensive square-root computation. We study both these variants of EdDSA verification. Experimental results show that for small batch sizes the Algorithms S2$'$ and SP yield speedup comparable to what is achieved by Algorithm N$'$ which is originally proposed as the default EdDSA batch-verification algorithm.

Keywords: Elliptic Curve, Edwards Curve, Montgomery Ladder, Symbolic Computation, Batch Verification, ECDSA, EdDSA, Randomization.

1 Introduction

The concept of digital signatures is proposed in [1] by Diffie and Hellman. The first practically applicable signature scheme RSA is proposed by Rivest, Shamir and Adleman in 1978 [2]. The security of the RSA algorithm is based allegedly on the hardness of the factorization of products of two large primes. In 1985, ElGamal proposes a new type of digital signature scheme based on the discrete logarithm problem in prime fields [3]. The ElGamal signature scheme is the first digital-signature scheme which is probabilistic in nature. The Digital Signature Algorithm (DSA) [4] is a variant of the ElGamal digital signature scheme, proposed as a standard by the National Institute of Standards and Technology (NIST) in 1991. In 2001, the Elliptic Curve Digital Signature Algorithm (ECDSA) is proposed by Johnson et al. [5] and is again accepted as a digital-signature standard. Bernstein et al. in 2011 propose the Edwards curve digital signature algorithm (EdDSA) [6]. ECDSA and EdDSA derive their security from the apparent intractability of the discrete logarithm problem in elliptic and Edwards curves defined over finite fields.

To verify an ElGamal-like signature, one requires two finite-field exponentiations (for DSA) or two scalar multiplications in the underlying curve (for ECDSA and EdDSA). Each such modular exponentiation or scalar multiplication is considerably more time-consuming than the other finite-field operations. EdDSA verification additionally involves a square-root computation in the finite field. This overhead addresses the need of easy batch verification but incurs significant overhead even during individual verification. We can nevertheless adapt the ECDSA verification algorithm to EdDSA, thereby avoiding the costly square-root computation.

In all these ElGamal-like signature schemes, signature verification is somewhat slower than the signing procedure. Many applications (often real-time) need to verify multiple signatures in batches. In 1994, Naccache et al. introduce a method to handle signature batches [7]. They propose the concept of batch verification, where the verifier simultaneously verifies a batch in time less than the total time associated with the individual verification of the signatures. An interactive batch-verification procedure is proposed for DSA signatures in [7]. In 1997, the concept of batch RSA is introduced by Fiat [8]. 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. However, its drawback is that it does not adapt to the case of signatures from multiple signers.

The key sizes of ECDSA signatures are much smaller than the key sizes of RSA and DSA signatures at the same security level. In Table 1 derived from [11], L and N stand for the bit lengths of the public and the private keys for DSA, k is the bit length of the modulus in RSA, and f is the order of the base point of the elliptic-curve/Edwards-curve group. In order to achieve 256-bit security, ECDSA/EdDSA needs only 512-bit keys. At the same security level, the DSA and RSA key sizes should be at least $(15360, 512)$ and 15360 bits. Smaller key sizes make ECDSA/EdDSA attractive to many applications. Moreover, smaller key sizes lead to faster verification for ECDSA/EdDSA compared to RSA/DSA.

Table 1. Key sizes for digital-signature algorithms at different security levels

Bits of Security	DSA minimum (L,N)	RSA minimum k	ECDSA/EdDSA minimum f
80	(1024,160)	1024	160
112	(2048,224)	2048	224
128	(3072,256)	3072	256
192	(7680,384)	7680	384
256	(15360,512)	15360	512

The described batch-verification methods are not directly applicable to ECDSA signatures. ECDSA*, a modification of ECDSA introduced by Antipa et al. [12], permits an easy adaptation of the DSA batch-verification protocol of Naccache et al. Cheon and Yi [13] 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 accepted as a standard signature scheme like DSA, RSA or ECDSA [4]. Thus the use of ECDSA* is unacceptable, particularly in applications where interoperability is of important concern. Moreover, ECDSA* increases the signature size by approximately a factor of two compared to ECDSA without increasing the security.

Edwards curves, a normal form of elliptic curves, are introduced by Edwards in [14]. Bernstein et al. [6] apply these curves to cryptographic usage. Edwards curves offer faster addition and doubling formulas than elliptic curves. Moreover, the unified addition and doubling formulas make Edwards-curve cryptosystems resistant to simple side-channel attacks. A batch-verification procedure is also proposed by Bernstein et al. for Edwards curve digital signatures (EdDSA).

An application of our batch-verification algorithms is in secure vehicle-to-vehicle (V2V) communications in vehicular ad hoc networks (VANETs) (see [15] 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 provide a brief introduction to the ECDSA batch-verification algorithms of [16, 17] and the attacks against those [18]. Sections 3 elaborates the EdDSA algorithm given in [6]. An ECDSA-like variant of EdDSA verification is also discussed. Section 4 explains the adaptation of ECDSA batch-verification algorithms to EdDSA signatures. Section 5 deals with randomization issues in the context of EdDSA batch verification. Our experimental results are supplied and discussed in Section 6. Section 7 concludes the paper.

2 Background on ECDSA Batch Verification

We work over the elliptic curve

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

defined over a large prime field $\mathbb{F}_p$. We assume that the group $E(\mathbb{F}_p)$ is of prime order n. Let P be a fixed generator of $E(\mathbb{F}_p)$.

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

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

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

For simplicity, we assume that all of the t signatures come from the same signer, that is, $Q_i = Q$ for all i. In this case, Eqn(2) can be simplified as

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

Since the y-coordinates of R_i are not available in the signatures, we cannot straightaway compute the sum on the left side. In AfricaCrypt 2012, several batch-verification algorithms are proposed to solve this problem [16]. The naive algorithms are based upon the determination of the missing y-coordinate of each R_i using a square-root computation (we have $y_i^2 = r_i^3 + ar_i + b$). The symbolic-manipulation algorithms treat the unknown y-coordinates as symbols. Batch verification involves the eventual elimination of all these y-coordinates from Eqn(2) or (3) using the elliptic-curve equation. The symbolic algorithm S2$'$ turns out to be the fastest of the batch-verification algorithms proposed in [16].

In IndoCrypt 2012, Bernstein et al. [18] propose two attacks on these batch-verification algorithms. They also suggest that these attacks can be largely eliminated by randomizing the batch-verification process (a concept introduced by Naccache et al. [7]). For randomly chosen non-zero multipliers $\xi_1, \xi_2, \ldots, \xi_t$, the individual verification equations are now combined as

$$\sum_{i=1}^{t} \xi_i R_i = \left(\sum_{i=1}^{t} \xi_i u_i\right) P + \left(\sum_{i=1}^{t} \xi_i v_i Q_i\right) \tag{4}$$

or as

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

for the case of the same signer. Since the y-coordinates of R_i are not available in the ECDSA signatures, Eqn(4) or (5) is not directly applicable. Some efficient ways of randomizing the batch-verification algorithms of [16] are proposed in [19]. We mostly concentrate on standard ECDSA signatures (M, r, s) on M. If the ECDSA signature contains an extra bit to identify the correct square-root y of $r^3 + ar + b$ [20], we call this an ECDSA# signature. In another variant known as ECDSA* [20, 21], the entire point R replaces r in the signature. Neither ECDSA# nor ECDSA* is accepted as a standard. Since ECDSA* results in an unreasonable expansion in the signature size without any increase in the security, we do not consider this variant in this paper. ECDSA#, however, adds only one extra bit to a signature, and so we study the implications of having this extra bit.

2.1 ECDSA Batch Verification

The right side of Eqn(3) can be computed numerically using two scalar multiplications (or one double scalar multiplication). Let this point be (α, β). If R_i are reconstructed as $u_i P + v_i Q$, the effort is essentially the same as individual verification. The algorithms of [16] solve this problem in many ways.

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

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

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

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

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

In [17], Karati et al. propose a new ECDSA batch-verification algorithm based on elliptic-curve summation polynomial. This algorithm is known as Algorithm SP and, is theoretically and experimentally faster than S2$'$. In this algorithm, Eqns(2)–(5) are rewritten as

$$\sum_{i=1}^{t} (r_i, y_i) + (\alpha, -\beta) = \mathcal{O},$$

where (α, β) is the numeric sum on the right-hand side. This equation is satisfied if and only if $f_{t+1}(r_1, r_2, \ldots, r_t, \alpha) = 0$, where $f_k(x_1, x_2, \ldots, x_k)$ is the k-th summation polynomial that can be defined by induction on k [17].

2.2 Attacks on ECDSA Batch Verification

In the first attack of Bernstein et al. [18], 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 $\mathscr{O}$, 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 $R_1 = r_2s_2^{-1}Q_2$, $r_1 = x(R_1)$, and $s_1 = (e_1+r_1d_1)(k_2-e_2s_2^{-1})^{-1}$, where $e_1 = H(m_1)$, $e_2 = H(m_2)$, and H is a secure hash function. Now, R_1+R_2 and $(e_1s_1^{-1}+e_2s_2^{-1})P+r_1s_1^{-1}Q_1+r_2s_2^{-1}Q_2$ have the same value as $(k_2P+r_2s_2^{-1}Q_2)$. These forged signatures are verified if they are in the same batch.

Both these attacks become infeasible by the use of randomizers. If the verifier chooses l-bit randomizers, the security of the batch-verification procedure increases by 2^l. The randomizers need not be of full lengths (of lengths close to that of the prime order n of the relevant elliptic-curve group). As discussed in [22], much smaller randomizers typically suffice to make most attacks on batch-verification schemes infeasible. If the underlying field is of size d bits, then the best known algorithms (the square-root methods) to solve the ECDLP take $\tilde{O}(2^{d/2})$ times. As a result, $d/2$-bit randomizers do not degrade the security of the ECDSA scheme. Another possibility is to take $l = 128$ to get 128-bit security independent of the security guarantees of ECDSA.

3 Edwards Curve Digital Signature Algorithm (EdDSA)

Bernstein et al. in [6] propose the Edwards Curve Digital Signature Algorithm (EdDSA). This signature scheme is based on the group structure of the twisted Edwards curve over a prime field $\mathbb{F}_p$ defined as

$$E : -x^2+y^2 = 1+dx^2y^2, \tag{6}$$

where d is not a square element in $\mathbb{F}_p$ and $d \notin \{0,-1\}$. To set up EdDSA signatures, one fixes the following domain parameters:

$$\begin{aligned}
b &= \text{an integer} \geqslant 10,\\
H &= \text{a cryptographic hash function whose output is } 2b \text{ bits long},\\
p &= \text{a prime congruent to 1 modulo 4},\\
d &= \text{a non-square element in } \mathbb{F}_p,\ d \neq 0,-1,\\
l &= \text{a prime in the range } \left[2^{b-4}, 2^{b-3}\right],\\
B &= \text{a point of the curve that acts as the base point, } B \neq (0,1).
\end{aligned}$$

These domain parameters are same for all the entities participating in a network. The Edwards-curve group is an additive group, where the sum of two points $P_1 = (x_1,y_1)$

and $P_2 = (x_2, y_2)$ on the curve is the point $P_3 = P_1 + P_2 = (x_3, y_3)$ that can be computed using the twisted Edwards-curve addition formula as given in [23]:

$$(x_3, y_3) = (x_1, y_1) + (x_2, y_2) = \left(\frac{x_1 y_2 + x_2 y_1}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 + x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right) \tag{7}$$

Now, we describe the three parts of the EdDSA signature scheme. The signer creates his/her key pair using Algorithm 1. Let M be a message, and the EdDSA signature of the message be (R, S). Algorithm 2 generates the signature of the message. The validity of the signature is checked by Algorithm 3.

Algorithm 1 EdDSA Key Generation

```
INPUT: Domain Parameters.
OUTPUT: Public key $A$, private key $k$.

  – Choose a random $b$-bit string as $k$.
  – Compute $H(k) = (h_0, h_1, \ldots, h_{2b-1})$.
  – Compute $a = 2^{b-2} + \sum_{3 \le i \le b-3} 2^i h_i$.
  – Compute $A = aB$.
```

Algorithm 2 EdDSA Signature Generation

```
INPUT: Domain Parameters, message $M$, private key $k$, and $H(k) = (h_0, h_1, \ldots, h_{2b-1})$.
OUTPUT: The EdDSA signature $(R', S)$ on $M$.

  – Compute $r = H(h_b, \ldots, h_{2b-1}, M)$.
  – Compute $R = rB \in E$.
  – $R'$ = (the sign bit of the $x$-coordinate of $R$) || (the $y$-coordinate of $R$).
  – Compute $S = r + H(R', A, M)a \pmod{l}$.
```

Algorithm 3 EdDSA Signature Verification

```
INPUT: Domain Parameters, message $M$, public key $A$, and signature $(R', S)$.
OUTPUT: Accept or reject.

  – Compute $H(R', A, M)$.
  – Compute $R$ from $R'$ (using a square-root computation as described in the text).
  – Accept the signature if and only if the equation $SB = R + H(R', A, M)A$ holds.
```

In the verification Algorithm 3, we have to compute R from R' which contains the sign bit of the x-coordinate and the y-coordinate of the point R. From the known y-

Algorithm 4 Alternative EdDSA Signature Verification

```
INPUT: Domain Parameters, message $M$, public key $A$, and signature $(R',S)$.
OUTPUT: Accept or reject.

  – Compute $H(R',A,M)$.
  – Extract the $y$-coordinate $R_y$ of $R$ from $R'$.
  – Accept the signature if and only if the equation $R_y = y(SB - H(R',A,M)A)$ holds.
```

coordinate, we first compute two x-coordinates by $x \equiv \pm\sqrt{\frac{y^2-1}{dy^2+1}} \pmod{p}$, and then solve the sign problem using the sign bit present in R'. We can avoid the square-root computation in the verification method. We propose an alternative signature-verification Algorithm 4 which is a straightforward adaptation of the ECDSA signature-verification algorithm. The correctness of Algorithm 4 can be easily proved as follows. We have $S = (r + H(R',A,M)a) \pmod{l}$, that is, $r = S - H(R',A,M)a$. Multiplying both sides by B, we get $rB = SB - H(R',A,M)aB$, that is $R = SB - H(R',A,M)A$. Therefore $y(R) = y(SB - H(R',A,M)A)$.

4 Batch Verification of EdDSA

Like ECDSA, only the y-coordinate of an Edwards-curve point is sent in an EdDSA signature. An extra bit to identify the correct x-coordinate is included in the signature. All the batch-verification algorithms studied in connection with ECDSA apply equally well to EdDSA signatures. Suppose that we want to verify a batch $(M_1,R'_1,S_1),(M_2,R'_2,S_2),\ldots,(M_t,R'_t,S_t)$ of t EdDSA signatures. Let R_i be the corresponding point of R'_i. We combine the individual verification equations for the t signatures as:

$$\left(\sum_{i=1}^{t} S_i\right) B - \sum_{i=1}^{t} H(R'_i,A_i,M_i)A_i = \sum_{i=1}^{t} R_i. \tag{8}$$

If all the signatures are from the same signer, that is, $A_1 = A_2 = \cdots = A_t = A$, then Eqn(8) simplifies to:

$$\left(\sum_{i=1}^{t} S_i\right) B - \left(\sum_{i=1}^{t} H(R'_i,A_i,M_i)\right) A = \sum_{i=1}^{t} R_i. \tag{9}$$

Eqn(9) requires only two scalar multiplications. Unlike ECDSA, an EdDSA signature contains an extra bit of information to identify the x-coordinate of R uniquely (after solving a quadratic equation). We can compute the full Edwards-curve point R_i from R'_i for all i. This calls for t square-root computations modulo p. This algorithm is similar to Algorithm N$'$ of [16] and is called Algorithm EdN$'$ here. If the extra bit is not available in the EdDSA signature (or is ignored) to uniquely distinguish the x-coordinate, we have to try all the 2^t combinations of points to verify the batch. We call this naive method Algorithm EdN. The original EdDSA paper [6] recommends Algorithm EdN$'$ as the default batch-verification algorithm.

4.1 Adaptation of Algorithm S2′

We can remove the overhead of square-root computations altogether. The adaptation of Algorithm S2′ can solve this problem. Let us call this adapted version Algorithm EdS2′. We first divide the t Edwards-curve points $R_1, R_2, \ldots, R_t$ in two groups. Then, we rewrite Eqn (8) as:

$$\left(\sum_{i=1}^{\lfloor \frac{t}{2} \rfloor} R_i\right) = \left(\sum_{i=1}^{t} S_i\right) B - \left(\sum_{i=1}^{t} H(R'_i, A_i, M_i)\right) A - \left(\sum_{i=\lfloor \frac{t}{2} \rfloor + 1}^{t} R_i\right). \qquad (10)$$

We treat the x-coordinates of the points R_i as symbols and compute the symbolic sum of the two sides of Eqn(10). Let the symbolic sum on the left-hand side of Eqn(10) be Q_1, and that on the right-hand side be Q_2. For a valid batch, Q_1 and Q_2 are two symbolic representations of the same point. We have $y(Q_1) \in \mathbb{F}_p\left[x_1, x_2, \ldots, x_{\lfloor \frac{t}{2} \rfloor}\right]$ and $y(Q_2) \in \mathbb{F}_p\left[x_{\lfloor \frac{t}{2} \rfloor + 1}, x_{\lfloor \frac{t}{2} \rfloor + 2}, \ldots, x_t\right]$. Let $\phi = y(Q_1) - y(Q_2)$, so ϕ is a polynomial in $\mathbb{F}_p[x_1, x_2, \ldots, x_t]$. In ϕ, the maximum degree of any x_i is 1. We write ϕ as $ux_1 + v$, where $u, v \in \mathbb{F}_p[x_2, \ldots, x_t]$. Multiplying ϕ with $ux_1 - v$, we get

$$(ux_1 - v)\phi = (ux_1 - v)(ux_1 + v) = u^2 x_1^2 - v^2.$$

Substituting x_1^2 by $\frac{y_1^2 - 1}{dy_1^2 + 1}$, we get $\phi' = (ux_1 - v)\phi = u^2\left(\frac{y_1^2 - 1}{dy_1^2 + 1}\right) - v^2$. To keep the degrees of all remaining x_i to $\leqslant 1$, a substitution phase follows this elimination, in which we replace x_i^2 by $\frac{y_i^2 - 1}{dy_i^2 + 1}$ for all $i = 2, 3, \ldots, t$. Using the same procedure, we eliminate all the symbolic x-coordinates $x_2, x_3, \ldots, x_t$ one by one. At the end, if we obtain the zero polynomial, we accept the batch of signatures, else we reject it.

4.2 Edwards-Curve Summation Polynomials and Adaptation of Algorithm SP

Here, we mention the adaptation necessary to make Algorithm SP of [17] work for EdDSA batch verification. The two base cases f_2 and f_3 of Edwards-curve summation polynomials, and the recurrence relation to compute the summation polynomial f_t for $t \geq 4$ are:

$$\begin{aligned}
f_2(y_1, y_2) &= y_1 - y_2, \\
f_3(y_1, y_2, y_3) &= c^2(V - d^2 U y_1^2 y_2^2) y_3^2 - 2y_1 y_2 (V - dU) y_3 + (V y_1^2 y_2^2 - U), \\
&\qquad \text{where } U = (c^2 - y_1^2)(c^2 - y_2^2) \text{ and } V = (1 - c^2 d y_1^2)(1 - c^2 d y_2^2), \\
f_t(y_1, y_2, \ldots, y_t) &= \mathrm{Res}_Y(f_{t-k}(y_1, \ldots, y_{t-k-1}, Y), f_{k+2}(y_{t-k}, \ldots, y_t, Y)) \\
&\qquad\qquad \text{for } t \geqslant 4 \text{ and for any } k \text{ in the range } 1 \leqslant k \leqslant t - 3.
\end{aligned}$$

The summation polynomial f_t evaluated at the t arguments $y_1, y_2, \ldots, y_t$ is zero if and only if there exists an x_i in $\overline{\mathbb{F}}_p$ for each y_i, where $1 \leqslant i \leqslant t$, such that $-x_i^2 + y_i^2 = 1 + dx_i^2 y_i^2$. If the batch-verification condition of Eqn(8) or (9) is expressed as $\sum_{i=1}^{t}(x_i, y_i) + (-\alpha, \beta) = \mathscr{O}$, it therefore suffices to check whether $f_{t+1}(y_1, y_2, \ldots, y_t, \beta) = 0$. To restrict our attention to curve points defined over $\mathbb{F}_p$ only, we need to carry out the sanity check introduced in [17]. The sanity check for Edwards curves follows the same procedure as for elliptic curves (check whether the Legendre symbol $\left(\frac{(y_i^2 - 1)/(dy_i^2 + 1)}{p}\right) = 1$).

5 Randomization of EdDSA Batch-Verification Algorithms

EdDSA signatures can be randomized easily by methods similar to the randomization methods for ECDSA. For randomly chosen multipliers $\xi_1, \xi_2, \ldots, \xi_t$, we now verify whether the following equality holds:

$$\left(\sum_{i=1}^{t} \xi_i S_i\right) B - \sum_{i=1}^{t} \xi_i H(R'_i, A_i, M_i) A_i = \sum_{i=1}^{t} \xi_i R_i. \tag{11}$$

For the case of the same signer, that is, $A_1 = A_2 = \cdots = A_t = A$, Eqn(11) simplifies to:

$$\left(\sum_{i=1}^{t} \xi_i S_i\right) B - \left(\sum_{i=1}^{t} \xi_i H(R'_i, A_i, M_i)\right) A = \sum_{i=1}^{t} \xi_i R_i. \tag{12}$$

The default batch-verification algorithm for EdDSA is EdN$'$, in which we explicitly and uniquely compute the points R_i by square-root computations modulo p. Subsequently, their multiples $\xi_i R_i$ can be computed *numerically*. We finally check whether the condition of Eqn(11) or (12) holds. The process does not involve any symbolic or summation-polynomial computation. In a variant denoted by EdN, we assume that R_i cannot be uniquely determined, so we need to try all possible combinations of the signs of x_i. For each combination, randomization proceeds numerically as in the case of EdN$'$.

We may, however, ignore the presence of the extra bit in R'_i identifying the correct value of x_i. By doing so, we can adapt the randomized Algorithms EdS2$'$ and EdSP to work for EdDSA. This is motivated by a need to avoid costly square-root computations of Algorithm EdN$'$.

In order to apply Algorithm EdS2$'$ to the batch-verification Eqn(11) or (12), it suffices to compute the y-coordinates of all $\xi_i R_i$. As in the case of ECDSA, we can uniquely compute $y(\xi_i R_i)$ from the knowledge of ξ_i and $y(R_i)$ alone. More precisely, let $R = (x, y)$ be a point on the Edwards curve. Any multiple uR of R can be expressed as (hx, k), where $h, k \in \mathbb{F}_p$ are fully determined by (u and) the y-coordinate of R. R itself is so expressed with $h = 1$ and $k = y$. The sum of two multiples $P_1 = (h_1 x, k_1)$ and $P_2 = (h_2 x, y_2)$ of R is $P_1 + P_2 = (h_3 x, k_3)$, where

$$\begin{aligned} h_3 &= (h_1 k_2 + h_2 k_1)/(1 + d h_1 h_2 k_1 k_2 f), \\ k_3 &= (k_1 k_2 + h_1 h_2 f)/(1 - d h_1 h_2 k_1 k_2 f), \end{aligned}$$

with f precomputed as $f = x^2 = (y^2 - 1)/(dy^2 + 1) \in \mathbb{F}_p$. For Edwards curves, the doubling formula is the same as the addition formula. That is, the double of $P_1 = (h_1 x, k_1)$ is $2P_1 = (h_4 x, k_4)$, where

$$\begin{aligned} h_4 &= 2h_1 k_1/(1 + d h_1^2 k_1^2 f), \\ k_4 &= (k_1^2 + h_1^2 f)/(1 - d h_1^2 k_1^2 f). \end{aligned}$$

We henceforth refer to this computation of $y(\xi_i R_i)$ as the *seminumeric* randomization method.

We can also use *Montgomery ladders* [24] to compute $y(\xi_i R_i)$. For deriving the Montgomery-ladder formulas, let $P_1 = (h_1, k_1)$ and $P_2 = (h_2, k_2)$ be two points on the curve. For point addition, we need the y-coordinate of the point $P_1 - P_2$ as follows.

$$y(P_1 + P_2) = \frac{2k_1k_2(1 + dh_1^2h_2^2)}{1 - dh_1^2h_2^2(k_1k_2)^2} - y(P_1 - P_2).$$

Here, $h_i^2 = (k_i^2 - 1)/(dk_i^2 + 1)$ for $i = 1, 2$. Finally, point doubling uses the formula

$$y(2P_1) = \frac{k_1^2 + h_1^2}{1 - dh_1^2k_1^2},$$

where $h_1^2 = (k_1^2 - 1)/(dk_1^2 + 1)$. These formulas can be easily converted to projective coordinates.

Let us now theoretically compare the performance of the seminumeric method with that of the Montgomery-ladder method. Let $P_1 = (\alpha_1 x, \beta_1, \gamma_1)$ and $P_2 = (\alpha_2 x, \beta_2, \gamma_2)$ be two points on the curve in standard projective coordinates. The seminumeric method computes the sum $P_3 = P_1 + P_2 = (\alpha_3 x, \beta_3, \gamma_3)$ and the double $P_4 = 2P_1 = (\alpha_4 x, \beta_4, \gamma_4)$ as given below:

Point Addition

$A = \gamma_1 \cdot \gamma_2,\ B = A^2,\ C = \alpha_1 \cdot \alpha_2,\ C_1 = C \cdot f_x,\ D = \beta_1 \cdot \beta_2,\ E = d \cdot C_1 \cdot D,\ F = B - E,$
$G = B + E,\ \alpha_3 = A \cdot F \cdot ((\alpha_1 + \beta_1) \cdot (\alpha_2 + \beta_2) - C - D),\ \beta_3 = A \cdot G \cdot (D + C_1),$
$\gamma_3 = F \cdot G.$

Point Doubling

$B = (\alpha_1 + \beta_1)^2,\ C = \alpha_1^2,\ C_1 = C \cdot f_x,\ D = \beta_1^2,\ E_1 = C_1 + D,\ E_2 = C + D,\ H = \gamma_1^2,$
$J = E_1 - 2 \cdot H,\ \alpha_4 = (B - E_2) \cdot J,\ \beta_4 = E_1 \cdot (C_1 + D),\ \gamma_4 = E_1 \cdot J.$

Each of seminumeric point addition and point doubling requires one extra field multiplication than the optimized implementation given in [25]. More precisely, seminumeric point addition and doubling take $(11M + 1S)$ and $(4M + 4S)$ field operations respectively (ignoring the negligible time consumed by multiplication by d and field addition).

The Montgomery-ladder method requires $(14M + 6S)$ field operations for each addition and doubling combined in each iteration.

We can use any windowed variant of point multiplication in the seminumeric point multiplication method. On the contrary, no effective windowed variant is known for Montgomery ladders. Moreover, the practical ladder described in [26] is efficient only for constant multipliers, which is not the case with randomized batch verification. We therefore use only the binary ladder.

Let us use l-bit randomizers. If we use the w-NAF method in the seminumeric computation, the precomputation stage needs $(4M + 4S) + (2^{w-1} - 1)(11M + 1S)$ field operations, and $\left(\frac{l}{w+1}\right)(11M + 1S)$ field operations are required to perform the scalar multiplication. The seminumeric scalar multiplication is faster than the Montgomery-ladder method if

$$(4M + 4S) + (2^{w-1} - 1)(11M + 1S) + (4M + 4S)l + \left(\frac{l(11M + 1S)}{w + 1}\right) \leqslant l(14M + 6S).$$

Putting $w = 4$ and assuming $1M \approx 1S$, we deduce that for $l \geqslant 10$ the seminumeric method is faster than the Montgomery-ladder method.

6 Experimental Results

The algorithms are implemented in a 2.33 GHz Xeon server running Ubuntu Linux Version 2012 LTS. The algorithms are implemented using the GP/PARI calculator [27] (version 2.5.0 compiled by the GNU C compiler 4.6.2). We have used the symbolic-computation facilities of the calculator in our programs. All other functions (like scalar multiplication and square-root computation) are written as subroutines with minimal function-call overheads. Since the algorithms are evaluated in terms of the numbers of field operations, this gives a fair comparison of experimental data with the theoretical estimates. We have implemented windowed, w-NAF and frac-w-NAF methods for square-root computations and for numeric and seminumeric randomization methods. We have used affine and standard projective coordinates. We have performed all the experiments on the Edwards curve Ed25519 [6].

Table 2 lists the overheads associated with all the batch-verification algorithms. We present the times required for the numeric and seminumeric scalar multiplications in Table 4. The best results obtained are highlighted and used in speedup computations. In the randomization of the batch-verification algorithms, the scalars are not constant, so we have to compute the addition chain for each scalar multiplication. The timing figures presented in Table 4 include the addition-chain computation times. Table 3 shows the square-root computation times obtained by various windowed algorithms. The times needed to carry out the Montgomery-ladder scalar multiplication are supplied in Table 5. Finally, the overall speedup figures obtained by the four batch-verification algorithms EdN, EdN′, EdS2′ and EdSP are listed in Table 6. In the speedup table, we include the results using both the default signature-verification Algorithm 3 and the ECDSA-like signature-verification Algorithm 4.

For batch sizes in the range $2 \leqslant t \leqslant 7$, the speedup obtained by Algorithms EdS2′ and EdSP is competitive with that obtained by the default batch-verification Algorithm EdN′. Algorithms EdS2′ and EdSP outperform Algorithm EdN′ if we use the default Algorithm 3 for individual verification. On the other hand, if we use the ECDSA-like verification Algorithm 4 for individual verification, Algorithm EdS2′ outperforms Algorithm EdN′ for batch sizes $t \leqslant 7$, and Algorithm EdSP is faster than Algorithm EdN′ for batch sizes $\leqslant 5$.

In short, replacing square-root computations by symbolic or resultant computations does not degrade the batch-verification process, so long as we restrict only to small batches of signatures. However, the overhead of the default batch-verification algorithm EdN′ increases linearly with the batch size, whereas that of EdS2′ or EdSP increases exponentially. Consequently, EdN′ must eventually take over the exponential algorithms (not demonstrated in the experimental results though).

7 Conclusion

In this paper, we port several batch-verification algorithms proposed for ECDSA to EdDSA signatures. We also address the issues of randomizing the batch-verification

Table 2. Overhead (in ms) of different batch-verification algorithms for EdDSA

Batch Size	Algorithm			
t	EdN	EdN$'$	EdS2$'$	EdSP
2	0.08	0.03	0.06	0.06
3	0.24	0.04	0.12	0.10
4	0.63	0.06	0.24	0.12
5	1.54	0.07	0.52	0.28
6	3.71	0.08	0.96	1.36
7	8.74	0.10	2.02	2.72

Table 3. Times (in ms) of square-root computations in the underlying field

↓ Algorithm			Times (in ms)
w-numeric (affine)	$w=3$		0.36
	$w=4$		**0.28**
	$w=5$		**0.28**
w-NAF-numeric (affine)	$w=3$		**0.28**
	$w=4$		**0.28**
	$w=5$		0.32
Frac-w-NAF-numeric (affine)	$w=3$	$m=1$	0.33
	$w=4$	$m=1$	**0.28**
		$m=3$	0.32
		$m=5$	0.32
	$w=5$	$m=1$	0.32
		$m=3$	0.32
		$m=5$	0.32
		$m=7$	0.36
		$m=9$	0.32
		$m=11$	0.32
		$m=13$	0.36

Table 4. Times (in ms) of the numeric and seminumeric randomization methods

↓ Algorithm			Numeric Methods		SemiNumeric Methods	
			$l = 128$	$l = 255$	$l = 128$	$l = 255$
w-numeric (affine)	$w = 3$		2.28	4.40	2.40	4.68
	$w = 4$		2.28	4.41	2.44	4.61
	$w = 5$		2.45	4.40	2.60	4.69
w-NAF-numeric (affine)	$w = 3$		2.28	4.53	2.44	4.81
	$w = 4$		2.20	4.40	2.33	4.64
	$w = 5$		2.28	4.25	2.36	4.48
Frac-w-NAF-numeric (affine)	$w = 3$	$m = 1$	2.44	4.77	2.61	5.01
	$w = 4$	$m = 1$	2.45	4.76	2.52	4.96
		$m = 3$	2.44	4.73	2.53	4.97
		$m = 5$	2.44	4.73	2.56	4.89
	$w = 5$	$m = 1$	2.40	4.61	2.53	4.80
		$m = 3$	2.48	4.60	2.52	4.85
		$m = 5$	2.44	4.64	2.57	4.85
		$m = 7$	2.49	4.69	2.56	4.88
		$m = 9$	2.48	4.73	2.60	4.88
		$m = 11$	2.48	4.68	2.60	4.89
		$m = 13$	2.52	4.72	2.64	4.89
w-numeric (Jacobian projective)	$w = 3$		1.28	2.48	**1.40**	2.76
	$w = 4$		1.28	2.36	1.44	**2.68**
	$w = 5$		1.36	**2.44**	1.52	**2.68**
w-NAF-numeric (Jacobian projective)	$w = 3$		1.32	2.60	1.48	2.88
	$w = 4$		**1.24**	2.49	1.44	2.81
	$w = 5$		1.28	**2.44**	1.44	2.73
Frac-w-NAF-numeric (Jacobian projective)	$w = 3$	$m = 1$	1.53	2.92	1.60	3.12
	$w = 4$	$m = 1$	1.48	2.88	1.64	3.12
		$m = 3$	1.48	2.84	1.60	3.12
		$m = 5$	1.48	2.84	1.64	3.12
	$w = 5$	$m = 1$	1.49	2.80	1.60	3.04
		$m = 3$	1.48	2.85	1.64	3.04
		$m = 5$	1.48	2.80	1.60	3.04
		$m = 7$	1.48	2.84	1.60	3.09
		$m = 9$	1.52	2.84	1.65	3.08
		$m = 11$	1.53	2.81	1.64	3.04
		$m = 13$	1.52	2.84	1.64	3.08

Table 5. Times (in ms) of the Montgomery-ladder randomization method

Coordinate system	$l = 128$	$l = 255$
Affine	2.96	5.85
Standard projective	1.96	3.88

Table 6. Speedup (over individual verification) obtained by different randomized and non-randomized batch-verification methods in the case of the same signer for two verification algorithms

Batch Verification	Randomization	Batch	Algorithm 3		Algorithm 4	
Algorithm	Algorithm	Size	None*	$l = 128$	None*	$l = 128$
EdN	Numeric	2	1.87	1.29	1.77	1.22
		3	2.60	1.60	2.46	1.51
		4	3.11	1.78	2.94	1.68
		5	3.30	1.84	3.12	1.74
		6	3.01	1.75	2.85	1.65
		7	2.32	1.49	2.19	1.41
EdN′	Numeric	2	1.89	1.30	1.78	1.23
		3	2.69	1.63	2.54	1.54
		4	3.41	1.87	3.22	1.77
		5	4.06	2.06	3.84	1.94
		6	4.66	2.20	4.41	2.08
		7	5.20	2.31	4.92	2.19
EdS2′	Seminumeric	2	2.09	1.33	1.98	1.26
		3	3.10	1.68	2.93	1.59
		4	4.03	1.93	3.81	1.82
		5	4.78	2.08	4.52	1.97
		6	5.30	2.17	5.01	2.06
		7	5.23	2.16	4.95	2.05
EdSP	Seminumeric	2	2.09	1.33	1.98	1.26
		3	3.11	1.69	2.94	1.59
		4	4.13	1.95	3.90	1.84
		5	5.00	2.12	4.73	2.01
		6	4.96	2.11	4.69	2.00
		7	4.75	2.08	4.49	1.96

* without randomization

process. Our experimental results demonstrate that the default batch-verification algorithm proposed for EdDSA can be slightly improved by using the new developments based on symbolic and resultant computations, at least for small batch sizes. Further advances in this new area of research can substantially enhance the applicability of the new proposals for EdDSA signatures. It is a challenging open problem whether the time complexities of the new algorithms can be brought down from exponential to polynomial. Prospects of achieving breakthroughs are expected to keep research in this area alive in near future.

References

1. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22** (1976) 644–654
2. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21** (1978) 120–126
3. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31** (1985) 469–472
4. NIST: The digital signature standard. Communications of the ACM **35**(7) (1992) 36–40
5. Johnson, D., Menezes, A., Vanstone, S.A.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Sec. **1**(1) (2001) 36–63
6. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering **2**(2) (2012) 77–89
7. Naccache, D., M'Raihi, D., Rapheali, D., Vaudenay, S.: Can DSA be improved: Complexity trade-offs with the Digital Signature Standard. In: Eurocrypt '94, Spinger (1994) 85–94
8. Fiat, A.: Batch RSA. Journal of Cryptology **10** (1997) 75–88
9. Harn, L.: Batch verifying multiple RSA digital signatures. Electronics Letters **34**(12) (1998) 1219–1220
10. Hwang, M.S., Lin, I.C., Hwang, K.F.: Cryptanalysis of the batch verifying multiple RSA digital signatures. Informatica **11**(1) (2000) 15–19
11. NIST: SP 800-52 Rev. 1. NIST Special publication (2013)
12. Antipa, A., Brown, D., Gallant, R., Lambert, R., Struik, R., Vanstone, S.: Accelerated verification of ECDSA signatures. In: SAC. Volume 3897 of Lecture Notes in Computer Science., Springer (2006) 307–318
13. Cheon, J.H., Yi, J.H.: Fast batch verification of multiple signatures. In: PKC. Volume 4450 of Lecture Notes in Computer Science., Springer (2007) 442–457
14. Edwards, H.M.: A normal form for elliptic curves. Bulletin of American Mathematical Society **44**(3) (2007) 393–422
15. 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) (2013) 1–27
16. Karati, S., Das, A., Chowdhury, D.R., Bellur, B., Bhattacharya, D., Iyer, A.: Batch verification of ECDSA signatures. In Mitrokotsa, A., Vaudenay, S., eds.: AFRICACRYPT. Volume 7374 of Lecture Notes in Computer Science., Springer (2012) 1–18
17. Karati, S., Das, A.: Faster batch verification of standard ECDSA signatures using summation polynomials. In: ACNS. Volume 8479 of Lecture Notes in Computer Science., Springer (2014) 438–455

18. Bernstein, D.J., Doumen, J., Lange, T., Oosterwijk, J.J.: Faster batch forgery identification. In: INDOCRYPT. Volume 7668 of Lecture Notes in Computer Science., Springer (2012) 454–473
19. Karati, S., Das, A., Chowdhury, D.R.: Using randomizers for batch verification of ecdsa signatures. IACR Cryptology ePrint Archive **2012** (2012) 582
20. Antipa, A., Brown, D.R.L., Gallant, R.P., Lambert, R.J., Struik, R., Vanstone, S.A.: Accelerated verification of ECDSA signatures. In: Selected Areas in Cryptography. (2005) 307–318
21. Cheon, J.H., Yi, J.H.: Fast batch verification of multiple signatures. In Okamoto, T., Wang, X., eds.: Public Key Cryptography. Volume 4450 of Lecture Notes in Computer Science., Springer (2007) 442–457
22. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: EUROCRYPT. Volume 1403 of Lecture Notes in Computer Science., Springer (1998) 236–250
23. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards curves. In: Africacrypt. Volume 5023 of Lecture Notes in Computer Science., Springer (2008) 389–405
24. Montgomery, P.L.: Speeding up pollard and elliptic curve methods of factorization. In: Mathematics of Computation. Volume 48(177). (1987) 243–264
25. Bernstein, D.J., Lange, T.: Explicit-formulas database (2007) Available at `http://www.hyperelliptic.org/EFD/index.html`.
26. Montgomery, P.L.: Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains. Microsoft research article (1992) 582
27. PARI Group: PARI/GP home (2008) Available at `http://pari.math.u-bordeaux.fr/`.