Lightweight EdDSA Signature Verification for the Ultra-Low-Power Internet of Things

Johann Großschädl[1], Christian Franck[1], and Zhe Liu[2]

[1] University of Luxembourg, Department of Computer Science,
6, Avenue de la Fonte, L–4364 Esch-sur-Alzette, Luxembourg
{johann.groszschaedl,christian.franck}@uni.lu

[2] Nanjing University of Aeronautics and Astronautics (NUAA),
College of Computer Science and Technology,
29 Jiangjun Avenue, Nanjing 211106, Jiangsu, China
zhe.liu@nuaa.edu.cn

Abstract. EdDSA is a digital signature scheme based on elliptic curves in Edwards form that is supported in the latest incarnation of the TLS protocol (i.e. TLS version 1.3). The straightforward way of verifying an EdDSA signature involves a costly double-scalar multiplication of the form $kP - lQ$ where P is a "fixed" point (namely the generator of the underlying elliptic-curve group) and Q is only known at run time. This computation makes a verification not only much slower than a signature generation, but also more memory demanding. In the present paper we compare two implementations of EdDSA verification using Ed25519 as case study; the first is speed-optimized, while the other aims to achieve low RAM footprint. The speed-optimized variant performs the double-scalar multiplication in a simultaneous fashion and uses a Joint-Sparse Form (JSF) representation for the two scalars. On the other hand, the memory-optimized variant splits the computation of $kP - lQ$ into two separate parts, namely a fixed-base scalar multiplication that is carried out using a standard comb method with eight pre-computed points, and a variable-base scalar multiplication, which is executed by means of the conventional Montgomery ladder on the birationally-equivalent Montgomery curve. Our experiments with a 16-bit ultra-low-power MSP430 microcontroller show that the separated method is 24% slower than the simultaneous technique, but reduces the RAM footprint by 40%. This makes the separated method attractive for "lightweight" cryptographic libraries, in particular if both Ed25519 signature generation/verification and X25519 key exchange need to be supported.

1 Introduction

Digital signature schemes can be used to provide entity and data-origin authentication, integrity protection, and non-repudiation services, which makes them an essential tool for enabling secure communication over the Internet. Common security protocols like TLS rely on these services to authenticate the server to the client (and optionally the client to the server) and to securely exchange the public keys needed for the establishment of a shared pre-master secret [34]. To date, the most widely used signature schemes are based on the RSA algorithm [35] and a variant of the ElGamal cryptosystem, which is standardized by the NIST [30]. However, signature schemes operating on elliptic curves, such as the Elliptic Curve Digital Signature Algorithm (ECDSA) from [30], have gained in acceptance over the past few years. What makes ECDSA attractive is that its security is based on the intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP), which allows one to use much smaller groups compared to its classical counterpart RSA, whose security rests on the Integer Factorization Problem (IFP). For example, it is generally accepted that ECDSA instantiated with a 160-bit elliptic-curve group provides about the same level of security as the RSA signature scheme using a 1024-bit modulus [19]. Smaller group sizes directly translate into shorter signatures, which is a crucial feature in settings where communication bandwidth is limited or data transfer consumes a large amount of energy (e.g. battery-powered devices [15]). Another major difference between RSA and ECDSA is the (relative) complexity of signature generation versus signature verification. While the verification of an RSA signature is less costly than the generation, exactly the opposite holds for ECDSA: verifying an ECDSA signature is more demanding than signature generation.

From an arithmetic point of view, the main operation of elliptic curve cryptosystems such as ECDSA is scalar multiplication, a computation of the form $R = kP$ where k is a positive integer and R, P are points on an elliptic curve E over a finite field $\mathbb{F}_q$. This computation can be decomposed into a sequence of point additions and point doublings, both of which, in turn, consist of arithmetic operations in the field $\mathbb{F}_q$ [12, 19]. In the case of signature generation, the scalar multiplication is performed on a point P that is fixed and known a priori since it is part of the domain parameters (namely, it is generator of a subgroup of prime order). Therefore, it is possible to speed up the scalar multiplication through pre-computation of multiples of P following the comb approach or the windows method [19]. Both techniques are suitable for resource-limited devices with little RAM since, at any time, only one point from the table (but not the full table) is required as input for the computation, which means the table can actually be stored in non-volatile memory [26]. The verification of a signature is more costly and requires a double-scalar multiplication, which is a computation of the form $R = kP + lQ$ where P is fixed (it is actually the same point P as in the signature generation), while Q is the signer's public key and, thus, becomes only available at run time [19, 30]. There exist different implementation options for a double-scalar multiplication, whereby the most widely-used approach is to compute the sum $kP + lQ$ in a simultaneous fashion with "joint" doublings as described in [19, Algorithm 3.48]. Assuming that each of the two scalars k and l has a length of b bits, the simultaneous double-scalar multiplication technique requires b point doublings, while the number of point additions depends on the joint Hamming weight of the two scalars.

The Edwards-curve Digital Signature Algorithm (EdDSA) is a state-of-the-art signature scheme using elliptic curves in (twisted) Edwards form that was developed with the intention of achieving both high performance (especially in software) and high security [8, 9]. A variant of EdDSA as specified in RFC 8032 [21] is one of the digital signature systems supported in the most-recent version of the TLS protocol, i.e. TLS 1.3. EdDSA is a “Schnorr-like” signature scheme that combines the strong security and simplicity of classical Schnorr signatures [36] with the efficiency (and further positive implementation aspects) of twisted Edwards curves [6]. However, unlike the original Schnorr scheme, EdDSA uses a double-size hash function (to help alleviate concerns regarding hash-function security) and generates the per-message secret nonces in a deterministic fashion by hashing each message together with a long-term secret. Thus, EdDSA does not consume fresh randomness for each message to be signed, which makes the scheme attractive for constrained environments (e.g. embedded systems) where the generation of random numbers is very costly due to the absence of reliable sources of entropy. In ECDSA, on the other hand, a unique and unpredictable random number is required for each computation of a signature, whereby even a small weakness in the random-number generation can have fatal consequences and may, in the worst case, leak the signer’s secret key. Thus, the deterministic nonce generation method of EdDSA is not only a performance feature but also a security feature. To verify an EdDSA signature, one has to check whether an equation of the form $R = kP - lQ$ holds or not. This is normally accomplished by computing $kP - lQ$ and then comparing the obtained result with R [8].

A common problem of both ECDSA and EdDSA is that the verification is significantly slower and also consumes much more memory than the generation of a signature. The high computational complexity of the verification operation of curve-based signature schemes is widely recognized in the literature and has initiated a body of research on techniques to speed up double-scalar multiplication [19]. When using a simultaneous approach to compute $R = kP \pm lQ$, this can be achieved by representing the two scalars k and l in such a way that the number of required point arithmetic operations is reduced, or by reducing the individual cost of the point arithmetic operations, or through the combination of both (as in e.g. [7]). While the massive computational burden of verification affects basically any implementation, the problem of high memory consumption is mainly relevant for embedded software that runs on resource-limited devices with little memory, such as smart cards or wireless sensor nodes. Recently, Liu et al. [25] presented a lightweight elliptic curve software for embedded platforms and reported that, on a 16-bit MSP430 microcontroller, the verification operation of their signature scheme consumes about 5 kB of stack memory, while the signature generation needs a stack space of merely 1.6 kB. In other words, the verification is roughly three times more “memory hungry” than the generation of a signature. In the past, there was relatively little awareness of this problem because resource-constrained devices like smart cards were exclusively used to generate signatures, but not for verification. However, the recent growth of the Internet of things has created a demand to support advanced security protocols (involving verifications) on restricted devices, and in such settings the memory consumption is indeed a serious problem, as was recently pointed out in [3].

In this paper, we present an approach to make the double-scalar multiplication required for the verification of an EdDSA signature more "lightweight" in terms of RAM footprint. Our basic idea is to exploit the birational equivalence between twisted Edwards curves and Montgomery curves in order to combine their individual arithmetic benefits. More concretely, we split the computation of $kP - lQ$ into two separate steps, namely the fixed-base scalar multiplication kP carried out with a fixed-base comb method using the twisted Edwards form of the curve, and the variable-base scalar multiplication lQ, which we perform with the straightforward (i.e. "X-coordinate-only") Montgomery ladder on the birationally-equivalent Montgomery curve [29]. At the end of the ladder computation, the (projective) Y coordinate of the result is recovered according to the formulae from [31], and the obtained projective point is then converted to the corresponding projective point on the birationally-equivalent twisted Edwards curve so that it can be subtracted from kP. Intuitively, one would expect this approach to be memory-efficient since the two scalar multiplications are carried out sequentially and both the fixed-base comb method on the twisted Edwards curve and the variable-base Montgomery ladder on the Montgomery curve can be optimized to have a RAM footprint of below 1 kB as shown in [26]. On the other hand, one would also expect the "separated" approach to be slower than a simultaneous double-scalar multiplication since it requires more point additions and doublings. The experimental results we report in this paper allow one to analyze the trade-offs between execution time and RAM footprint these two approaches provide. We also discuss some corner cases in the point conversion and the recovery of the Y coordinate that require special attention.

2 Preliminaries

In this section, we first describe the EdDSA signature scheme and then give an overview of the arithmetic properties of (twisted) Edwards curves.

EdDSA. The Edwards-curve Digital Signature Algorithm (EdDSA) is a state-of-the-art signature scheme that provides high speed in software (especially on 64-bit platforms) and high security [8, 9]. EdDSA was obviously inspired by the classical Schnorr signature algorithm [36], which, in its original form, uses $\mathbb{Z}_p$ as underlying algebraic structure, but can be straightforwardly adapted for elliptic curve groups; see e.g. [10, Sect. 4.2.3] for a formal description of a curve-based variant. However, EdDSA comes with a number of enhancements compared to [10] that were developed with the goal to improve the real-world security of the scheme. The major differences between EdDSA and the EC-Schnorr signature algorithm described in [10] are as follows.

– EdDSA is a deterministic signature scheme since it employs a deterministic process to generate the secret scalar r (called "session key" in [8]) needed to sign a message M. Concretely, EdDSA generates r by hashing a long-term secret together with M. In this way, the signing operation does not require any fresh randomness and it is also guaranteed that a value r is never used for different messages. On the other hand, the classical EC-Schnorr scheme from [10] has to produce a fresh random value r for each message M to be signed. This r must be unique for every M and chosen uniformly from the set $\{1, 2, \ldots, \ell - 1\}$, where ℓ is the order of the base point. Even marginal deviations from randomness or a slight non-uniformity of the distribution from which r is taken can enable an attack against the EC-Schnorr scheme that may allow an adversary to get the signer's private key. EdDSA avoids such problems and is, therefore, particularly suited for environments where accessing a source of high-quality randomness is not easily possible.

– A distinguishing characteristic of curve-based Schnorr signature schemes is that they hash the message M together with $R = rB$, i.e. the result of the scalar multiplication between the secret scalar r and the base point B. The EC-Schnorr variant specified in [10] actually computes $\textsc{Hash}(M, x_R)$ where x_R is the x-coordinate of R. EdDSA, on the other hand, also includes the signer's public key A in the hash computation; more precisely, it computes $\textsc{Hash}(R, A, M)$ as part of the signature generation. The purpose of this so-called key-prefixing is to provide an "inexpensive way to alleviate concerns that several public keys could be attacked simultaneously" [8]. Indeed, as recently proven by Bernstein [5], single-user security for Schnorr signatures tightly implies multi-user security for key-prefixed Schnorr signatures in the standard model. Shortly after the publication of [5], Kiltz et al. [22] found that key-prefixing is not needed to ensure multi-user security and provided a reduction showing that "strong" single-user unforgeability tightly implies "strong" multi-user unforgeability in the random oracle model. However, to date, proving multi-user security using standard unforgeability assumptions without key-prefixing remains being an open problem.
– EdDSA supports fast verification of (large) batches of signatures, which is not (efficiently) possible when using the EC-Schnorr scheme from [10]. The saving in execution time that can be achieved through a batch verification of 64 signatures (versus an individual verification of 64 signatures) is more than 52% according to the experimental results reported in [8]. To achieve this speed-up, the designers of EdDSA modified the signature generation to output the (compressed) point $R = rB$ as first component of the signature instead of $\textsc{Hash}(M, x_R)$ as in EC-Schnorr. This tweak does not impact the security compared to EC-Schnorr since, given an EC-Schorr signature and the signer's public key, one can always recover R as in [10, Sect. 4.2.3.2].
– When designing an elliptic curve signature scheme, it is common practice to choose a hash function with an output length matching the bit-length of the order ℓ of the base point B. Choosing the hash function in this way is also recommended for the EC-Schnorr algorithm in [10]. However, the designers of EdDSA were more conservative and recommend to employ a double-size hash function, claiming it "helps alleviate concerns regarding hash-function security" [8]. Specifically, they recommend to use SHA-512 when EdDSA is instantiated with a twisted Edwards curve that is birationally equivalent to Curve25519 and a base point B whose order ℓ has a bit-length of 253.

Algorithm 1. EdDSA signature generation (sketch)

```
Input: Domain parameters $(\mathbb{F}_q, E, B, \ell)$, signer's key pair $(a, A)$, signer's long-term
    secret $n$ for session-key generation, and message $M$.
Output: Signature $(R, s)$ of $M$.
1: $r \leftarrow \textsc{Hash}(n, M) \bmod \ell$
2: $R \leftarrow rB$
3: $h \leftarrow \textsc{Hash}(R, A, M) \bmod \ell$
4: $s \leftarrow r + ha \bmod \ell$
5: return $(R, s)$
```

Algorithm 1 specifies a (slightly) simplified version of the EdDSA signature generation as described in [8]. We left out some details that are not relevant in the context of the present paper. One such detail concerns the long-term secrets a and n, which are generated by hashing a secret "master key." In addition, the points R and A in line 3 and 5 are actually compressed, i.e. represented by the y-coordinate and one bit of the x-coordinate (see [8] for further details). When using the curve promoted by the EdDSA designers, which is a twisted Edwards curve birationally equivalent to Curve25519 [4], then a compressed point fits in 32 bytes and the complete signature has a size of 64 bytes. As shown in Algorithm 1, the message M is actually hashed twice, whereby one of the inputs to the second hash computation in line 3, namely the point $R = rB$, depends on the result of the first hash computation in line 1. This dependency may require the signer to buffer the complete message M, which could exceed the available memory capacity when M is large. Furthermore, this "double hashing" is also computationally expensive for large messages[3]. On the other hand, when M is relatively small, then the overall execution time of the signature generation is primarily determined by the scalar multiplication $R = rB$ in line 2, which is, in fact, a fixed-base scalar multiplication since B is a pre-defined point.

Algorithm 2. EdDSA signature verification (sketch)

```
Input: Domain parameters $(\mathbb{F}_q, E, B, \ell)$, signer's public key $A$, message $M$, and alleged
    signature $(R, s)$.
Output: Acceptance or rejection of signature.
1: $h \leftarrow \textsc{Hash}(R, A, M) \bmod \ell$
2: return *Accept* if $R = sB - hA$ and *Reject* otherwise
```

Algorithm 2 describes the operations that need to be performed in order to verify an EdDSA signature. In particular, for short messages, one can assume that the hash computation in line 1 is relatively inexpensive, which means the

[3] RFC 8032 [21] specifies besides the original EdDSA scheme also a pre-hash version that replaces the message M in Algorithm 1 by its hash value $m = \textsc{Hash}(M)$. This pre-hashing potentially reduces the execution time and RAM requirements for large messages, but loses the collision-resilience feature of the original EdDSA.

overall execution time will be mainly determined by checking whether R equals $sB - hA$ or not. This check can be carried out in a few different ways, but the most common approach is to compute $sB - hA$ using an algorithm optimized for double-scalar multiplication (i.e. an algorithm that computes sB and hA in an interleaved or simultaneous fashion with "joint" doublings) and compare the result with R. The performance can be further improved by pre-computation of multiples of the points B and A (and possibly also combinations thereof) as well as by using a special representation of the two scalars s and h to minimize their joint weight; see e.g. [7, 12, 19] for a more detailed treatment. However, on memory-restricted devices, it generally makes sense to represent the scalars in Joint-Sparse Form (JSF) [37] since in this case the verifier has to pre-compute and store just two points, namely $B - A$ and $B + A$. An alternative technique to verify an EdDSA signature consists of computing $R + hA$ and sB, and then checking whether they are equal or not, which can be efficiently done using the projective representations of the points (i.e. no projective-to-affine conversions are required). However, a drawback of this approach is that the verifier has to carry out a costly decompression of R.

Twisted Edwards Curves. EdDSA uses a special class of elliptic curves, the so-called *twisted Edwards (TE)* curves, which were first described by Bernstein et al. in 2008 [6]. A TE curve over a non-binary finite field $\mathbb{F}_q$ is defined by an equation of the form

$$E_T: \ ax^2 + y^2 = 1 + dx^2y^2 \tag{1}$$

where a and d are distinct and non-zero. The order of a TE curve is a multiple of four, and every TE curve contains a point of order two, which is $(0, -1)$. An interesting feature of TE curves is the existence of a neutral element $\mathcal{O} = (0, 1)$ that is an affine point on the curve. The formula for point addition

$$\underbrace{(x_3, y_3)}_{P_3} = \underbrace{(x_1, y_1)}_{P_1} + \underbrace{(x_2, y_2)}_{P_2} = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}\right)$$

is *unified* and can, therefore, also be used for point doubling, i.e. it yields the correct result when $P_1 = P_2$. Further, it is *complete* when a is a square and d is a non-square in $\mathbb{F}_q$, so that the correct sum is computed for any pair of points (including special cases like $P_1 = \mathcal{O}$, $P_2 = \mathcal{O}$, $P_2 = -P_1$). The additive inverse of a point (x, y) is the point $(-x, y)$. Any TE curve is birationally-equivalent to a Montgomery curve [29] (i.e. a curve defined by $By^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$) and vice versa. The specific TE curve recommended by the EdDSA designers is birationally-equivalent to Curve25519 [4] and has the parameters $a = -1$ and $d = -121665/121166 \in \mathbb{F}_p$ with $p = 2^{255} - 19$. The group $E_T(\mathbb{F}_p)$ is isomorphic to $\mathbb{Z}_\ell \times \mathbb{Z}_8$ where ℓ is a 253-bit prime (see [8, 9] for more details).

When $a = -1$, the *extended TE coordinates* introduced in [20] allow one to perform a "mixed" point addition with only seven multiplications (7M) in the underlying field [26]. Doubling a point in extended projective coordinates costs three multiplications (3M) and four squarings (4S).

3 Implementation Options for EdDSA Verification

In this section we will have a closer look at different ways the verification of an EdDSA signature can be implemented, whereby we pay special attention to the double-scalar multiplication $sB - hA$. The straightforward approach, which is used by most (lightweight) cryptographic libraries, is to compute sB and hA in a combined fashion (i.e. with "joint" doublings) following e.g. the simultaneous or interleaving strategy [19]. An alternative approach is to completely separate these two scalar multiplications and exploit the birational equivalence between the TE form and the Montgomery form.

3.1 Simultaneous Double-Scalar Multiplication

There are two main approaches for performing the double-scalar multiplication $sB - hA$ in a combined fashion, namely the simultaneous method [19] and the interleaving technique [28], which have their origin in corresponding algorithms for multi-exponentiation. Both methods reduce the number of point doublings by half (compared to the separate computation of sB and hA) at the expense of increased RAM consumption for storing a pre-computed table that contains multiples (and possibly also linear combinations[4]) of the two base points A and B. Furthermore, both methods can utilize a "low-weight" representation of the scalars, e.g. *Non-Adjacent Form (NAF)* or *Joint-Sparse Form (JSF)* [19], which determines the actual execution time (i.e. the number of point additions) and the size of the pre-computed table. However, when RAM is limited, it makes generally sense to restrict the size of the table to a few points, e.g. four points including A and B. In this case, the simultaneous double-scalar multiplication with a JSF representation of the scalars s and h executes, on average, the same number of point additions in the evaluation phase as the interleaving technique with width-3 NAFs (see [19, Table 3.6] for a more detailed analysis). Since the width-3 NAFs of s and h require more RAM than their JSF representation, we decided to implement the simultaneous method.

The JSF utilizes a binary (i.e. radix-2) signed-digit number system with the digit set $D = \{-1, 0, 1\}$ to represent a pair of integers a, b such that they have minimal joint Hamming weight, which means the number of non-$(0, 0)$ columns is as small as possible. Solinas gave in [37] a formal definition of the JSF based on three properties and also proved both its uniqueness and optimality. More concretely, he showed that any pair of integers has a unique JSF and that this JSF has the least density of non-$(0, 0)$ columns among all joint expansions. The number of digits of the JSF representation of two positive integers exceeds the bitlength of the larger of these two integers by at most one digit. However, since each digit is from D and requires two bits for its representation, the JSF of the scalars s and h needed for EdDSA verification occupies 128 bytes in RAM.

[4] The main difference between the simultaneous method and the interleaving method is that, in the latter case, the table entries are disjoint with respect to the two base points A and B (i.e. each pre-computed value involves only a single base point).

Algorithm 3. Simultaneous method for double-scalar multiplication.

Input: Twisted Edwards curve E_T over $\mathbb{F}_q$ of cardinality $h\ell$ where ℓ is prime, rational points $A \in E_T(\mathbb{F}_q)$ and $B \in E_T(\mathbb{F}_q)$, scalars $h \in [0, \ell-1]$ and $s \in [0, \ell-1]$.
Output: Point $R = sB - hA$ in affine coordinates.

```
1:  $(s', h') \leftarrow$ JointSparseForm$(s, h)$
2:  $T \leftarrow [-A,\, B+A,\, B,\, B-A]$        {table with 2 affine and 2 proj. points}
3:  $T \leftarrow$ ProToExtAff$(T)$               {table with 4 extended affine points}
4:  $Q \leftarrow \mathcal{O}$
5:  for $i$ from Length$(s', h') - 1$ down to 0 do
6:      $Q \leftarrow 2Q$
7:      $d_i \leftarrow 3s'_i + h'_i$
8:      if $(d_i > 0)$ then $Q \leftarrow Q + T[d_i - 1]$ end if
9:      if $(d_i < 0)$ then $Q \leftarrow Q - T[\mathrm{Abs}(d_i) - 1]$ end if
10: end for
11: $R \leftarrow$ ProToAff$(Q)$
12: return $R$
```

Algorithm 3 shows a simplified implementation of the simultaneous method for the double-scalar multiplication $sB - hA$ using the JSF for the scalars. The computation of the JSF of the scalars s and h in line 1 is relatively inexpensive and can be done as specified in e.g. [37] or [19, Algorithm 3.50]. Thereafter, the entries of the table T are generated, starting with the sum $S = B + A$ and the difference $D = B - A$, which we obtain using the projective addition formulae from [6, Sect. 6]. We convert these two (projective) points to affine coordinates by taking advantage of the simultaneous inversion technique, i.e. we invert the product $Z_S Z_D$ and then obtain $1/Z_S$ and $1/Z_D$ by multiplying the result of the inversion by Z_D and Z_S, respectively [19, Algorithm 2.26]. Next, the four affine points $-A$, B, S, and D have to be represented in extended affine coordinates of the form (u, v, w) where $u = (x + y)/2$, $v = (y - x)/2$, $w = dxy$ [9, 24] and stored in table T. The bulk of the computation, in particular the doubling and addition/subtraction of points, is carried out in a relatively simple loop whose number of iterations corresponds to the length of the JSF expansion of the two scalars (approximately the bitlength of ℓ). In each iteration, a point doubling is performed (line 6) and an index d_i to access the table T is calculated based on the digits s'_i and h'_i (line 7). This index d_i is in the range $[-4, 4]$; depending on its value and sign, an entry of table T may be added or subtracted as specified in line 8 and 9. Since the negation of a point is cheap, it suffices to have a table with only four pre-computed points. The point Q (represented with extended projective coordinates) is initialized to the neutral element $\mathcal{O}$ and updated in each iteration of the loop until it eventually holds the result $sB - hA$, which is finally converted to standard affine coordinates (line 11).

Since, on average, roughly half of the columns of the JSF expansion of s and h are not $(0, 0)$ [37], the probability that $d_i \neq 0$ is roughly 50%. Thus, it can be expected that only in roughly half of the iterations of the loop a point addition (or subtraction) is actually performed. On the other hand, a point doubling is carried out in each iteration. Using the basic cost models for a mixed addition

(7M) and projective doubling (3M + 4S) mentioned in Sect. 2, we can estimate that, on average, $0.5 \cdot 7 + 3 = 6.5$ multiplications and four squarings in $\mathbb{F}_p$ are executed per iteration. Consequently, the complete cost of the loop amounts to about $6.5n$ multiplications and $4n$ squarings in $\mathbb{F}_p$ (where n is the length of the JSF expansion), i.e. roughly 6.5M + 4S per scalar bit. The pre-computed table T contains four points in extended affine coordinates, which means in our case the table occupies 384 bytes in RAM (96 bytes per point).

3.2 Two Separate Scalar Multiplications

An obvious alternative to the simultaneous method for obtaining $sB - hA$ is to split the computation into two completely separate parts, namely a fixed-base scalar multiplication sB, and a variable-base scalar multiplication hA. Intuitively, one expects this separated approach to be slower than the simultaneous method since significantly more point doublings have to be performed, which is likely the reason why this approach has, to our knowledge, never been analyzed in the literature. However, this disadvantage can be mitigated by exploiting the birational equivalence between TE curves and Montgomery curves, enabling us to take advantage of the highly-efficient Montgomery ladder to implement the variable-base scalar multiplication hA. The primary advantage of the separated approach is low memory consumption (in relation to the simultaneous method) since it requires neither a table with pre-computed points nor additional space for a JSF representation of the two scalars.

Algorithm 4. Scalar multiplication on TE curve using Montgomery ladder

```
Input: Twisted Edwards curve $E_T$ over $\mathbb{F}_q$ of cardinality $h\ell$ where $\ell$ is prime, rational
    point $P = (x, y) \in E_T(\mathbb{F}_q)$ with $\mathrm{ord}(P) \geq \ell$, scalar $k \in [0, \ell - 1]$.
Output: Point $Q = kP$ in projective coordinates.
1: if $k = 0$ then return $(0 : 1 : 1)$
2: if $k = \ell - 1$ then return $(-x : y : 1)$
3: $P_m \leftarrow$ TedToMon($P$)
4: $(Q_1, Q_2) \leftarrow$ MonLadder($P_m$)
5: $Q_r \leftarrow$ RecoverY($Q_1, Q_2, P_m$)
6: $Q \leftarrow$ MonToTed($Q_r$)
7: return $Q$
```

Algorithm 4 explains how one can perform a variable-base scalar multiplication kP (where P is a public key, i.e. a rational point on a TE curve) using the Montgomery ladder on the birationally-equivalent Montgomery curve, which is in the case of Ed25519 a curve[5] that is isomorphic to Curve25519. At first, the point P on the TE curve is mapped to the Montgomery curve with help of the

[5] The specific Montgomery curve that is birationally-equivalent to the TE curve used by Ed25519 has the same parameter A as Curve25519 (i.e. $A = 48662$ [4]), but the parameter B differs since $B = -(A + 2) = -48664$ instead of $B = 1$.

formulae given in [6]. This mapping involves a costly inversion, since to achieve maximum performance, the input point for the Montgomery ladder needs to be represented in affine coordinates. Thereafter, the Montgomery ladder is carried out in a similar fashion as in X25519 key exchange [4] (i.e. the point arithmetic involves only the projective X and Z coordinate) and, thus, achieves the same efficiency. However, there are two deviations from the X25519 ladder, namely (i) the Y coordinate of the resulting point has to be recovered, and (ii) the main loop of the ladder (as specified in e.g. [17]) needs to be modified because, unlike X25519, it can not be taken for granted that the most significant "1" bit of the scalar is always at the same position. Finally, the resulting point in projective $(X : Y : Z)$ coordinates has to be converted to the corresponding point on the TE curve. This TE point can be in projective coordinates since it is added to the result of the fixed-base scalar multiplication sB, which is usually also given in projective coordinates. Only at the very end, a single inversion is necessary to get the final result (i.e. the sum of the results of the two scalar multiplications) in affine coordinates. Although the basic principle of performing the variable-base scalar multiplication hA on the birationally-equivalent Montgomery curve is fairly straightforward, there are a couple of corner cases that require special attention. Such corner cases can occur in (i) the point conversions between the TE form and the Montgomery form, and (ii) the recovery of the Y coordinate at the end of the Montgomery ladder.

Corner Cases of Point Conversion. An affine point (x_t, y_t) on a TE-form elliptic curve E_T can be converted to the corresponding point (x_m, y_m) on the birationally-equivalent Montgomery curve E_M using the following map [6].

$$\phi :\ (x_t, y_t) \mapsto (x_m, y_m) = \left(\frac{1+y_t}{1-y_t}, \frac{1+y_t}{(1-y_t)x_t} \right) \tag{2}$$

Obviously, the map ϕ is not defined for $x_t = 0$ or $y_t = 1$. Since the parameters a and d of a TE curve E_T must be distinct and nonzero, there exists only one point with $y_t = 1$, namely the neutral element $(0, 1)$, which corresponds to the point at infinity $\mathcal{O}$ on the birationally-equivalent Montgomery curve. There are two points on E_T with $x_t = 0$; one is the neutral element $(0, 1)$ and the other is the point $(0, -1)$. This point has order 2 and corresponds to the point $(0, 0)$ on the Montgomery curve, which also has order 2 [6].

Given an affine point (x_m, y_m) on a Montgomery curve E_M governed by the equation $By^2 = x^3 + Ax^2 + x$, one can compute the corresponding point on the birationally-equivalent TE curve E_T using the map

$$\psi :\ (x_m, y_m) \mapsto (x_t, y_t) = \left(\frac{x_m}{y_m}, \frac{x_m - 1}{x_m + 1} \right). \tag{3}$$

The map ψ is not regular at points with $x_m = -1$ or $y_m = 0$; in particular ψ is undefined at the affine point $(0, 0)$ on E_M. Another special case for which ψ is irregular are the points with $x_m = -1$. By setting x_m to -1, we can write the

Montgomery-curve equation as $By_m^2 = A - 2$ to make it clear that such points only exist when $(A-2)/B$ is a square in $\mathbb{F}_p$, which obviously does not apply to our curve. Hence, in summary, corner cases in the conversion of points between the TE model and the birationally-equivalent Montgomery model can only be caused by points of low order. However, since the input point for the variable-base scalar multiplication hA is the signer's public key A, it should never have low order, provided the signer generated his/her key pair in a proper fashion as specified in [8]. We will discuss low-order points further in the next subsection and describe how our implementation deals with them.

Corner Cases of $\boldsymbol{Y}$-Coordinate Recovery. Situations that require special attention can also emerge during the recovery of the Y coordinate as described in [31]. According to Algorithm 4, the Montgomery ladder actually returns two points, namely $Q_1 = kP_m$ and $Q_2 = Q_1 + P_m = kP_m + P_m = (k+1)P_m$ (see [13] for details). The X and Z coordinates of these two points, along with the affine x and y coordinates of the input point P_m, allow one to re-compute the projective Y-coordinate of Q_1, which is relatively inexpensive since it requires only ten multiplications and six additions/subtractions in $\mathbb{F}_p$. Given the points $Q_1 = (X_1 : Z_1)$, $Q_2 = (X_2 : Z_2)$, $P_m = (x_m, y_m)$, a full projective representation of Q_1 (including Y coordinate) can be obtained as follows [31]:

$$\begin{aligned}
X_r &= 2By_mZ_1Z_2X_1 \\
Y_r &= Z_2[(X_1 + x_mZ_1 + 2AZ_1)(X_1x_m + Z_1) - 2AZ_1^2] - (X_1 - x_mZ_1)^2X_2 \\
Z_r &= 2By_mZ_1Z_2Z_1
\end{aligned}$$

The coordinate Z_r of this new representation of Q_1 is a product of y_m, Z_1^2, and Z_2, but this does normally not change the value of the affine x coordinate since $x_r = X_r/Z_r = X_1/Z_1$. However, the equation for Z_r shows that recovering the affine y coordinate $y_r = Y_r/Z_r$ does not work when (i) the y coordinate of the ladder-input P_m is 0, or (ii) the projective Z coordinate of one of the output-points of the ladder (i.e. Z_1 or Z_2) is 0. The former case is only possible when P_m has order 2 [13], which means $P_m = (0,0)$ since there are no other points in the 2-torsion group of our Montgomery curve. A pragmatic approach to deal with this corner case is to simply reject a public key if it has low order (as we will discuss in detail in the next subsection). Preventing low-order points from entering the ladder also simplifies the analysis of the second corner case, i.e. the case $Z_1 = 0$ or $Z_2 = 0$. Namely, when we exclude low-order points as input to the ladder and insist that k is in the range $[0, \ell - 1]$, then $Z_1 = 0$ (i.e. $Q_1 = \mathcal{O}$) is only possible when $k = 0$. On the other hand, $Z_2 = 0$ (i.e. $Q_2 = \mathcal{O}$) implies $Q_1 = -P_m$ since $Q_2 = Q_1 + P_m$, which can only occur when $k = \ell - 1$.

So, in summary, when the order of the ladder input P_m is at least ℓ, there remain only two corner cases that require special attention when recovering the Y coordinate at the end of the ladder, namely $k = 0$ and $k = \ell - 1$. As shown in Algorithm 4, our implementation handles these special cases through if-then clauses (line 1 and 2) without actually executing the ladder.

Single Ladder for X25519 and Ed25519. The Montgomery ladder can be used to implement not only EdDSA verification, but also ECDH key exchange as described in [4]. This naturally raises the question whether one and the same ladder implementation can serve both cryptosystems and, in this way, reduce the code size of an ECC library. As already mentioned before, there are some subtle differences between a conventional X25519 ladder (see e.g. [17]) and the ladder we use to compute hA as part of EdDSA verification. In particular, due to the so-called "clamping" of scalars according to [4], the highest "1" bit of an X22519 scalar is always at the same position, which is not guaranteed for the scalar h computed during EdDSA verification (line 1 of Algorithm 2) since it is a hash value reduced modulo ℓ. Furthermore, a ladder for X25519 key exchange has to be resistant to timings attacks, whereas a ladder for EdDSA verification does not. Nonetheless, it is possible to implement a "unified" ladder that suits both X25519 and Ed25519 by adopting one of the following two strategies. The first is to initialize the ladder as usual (i.e. $Q_1 = (x_m : 1)$ and $Q_2 = 2Q_1$), then scan for the most-significant "1" bit in the scalar, and start the iteration of the ladder loop from the next-lower bit. This scanning for the highest "1" bit does not introduce a vulnerability to timing attacks, even when it is implemented in a naive way, since X25519 scalars are always "clamped" as specified in [4]. An alternative way is to initialize the ladder with $Q_1 = (0 : 1)$ and $Q_2 = (x_m : 1)$ to make it work correctly with leading "0" bits in a scalar. More precisely, this initialization allows one to fix the number of ladder iterations for X25519 and Ed25519 to e.g. 256 because the processing of leading "0" bits does not change Q_1 and also not the quotient X_2/Z_2 [13]. We implemented the first method as it enables slightly better performance for EdDSA verification.

Computation of $\boldsymbol{R = sB - hA}$. Besides the variable-base scalar multiplication hA, we also have compute sB, where $s \in [0, \ell - 1]$ is extracted from the signature to be verified and $B \in E_T(\mathbb{F}_p)$ is the generator of the cyclic sub-group specified by the parameter set for EdDSA [8]. This computation is a fixed-base scalar multiplication and can be carried out much faster than the variable-base scalar multiplication hA. Our software implementation executes this fixed-base scalar multiplication via a *fixed-base comb method* [19] with a radix-2^4 signed-digit representation for the scalar s, which means four bits of s are processed at once. The implementation uses a look-up table of eight pre-computed points that are stored in flash memory (and not in RAM) since B is fixed and known a priori. Our implementation of the fixed-base comb method is, in essence, the same as in [26], where a detailed description can be found. After computation of the two points sB and hA, which are obtained in projective coordinates, the latter has to be subtracted from the former. We use the (projective) addition formulae provided in [6] for this subtraction. Finally, the point $R = sB - hA$ is converted to standard affine coordinates and then compressed so that it can be compared with the compressed point contained in the signature.

Thanks to the extended projective coordinates introduced in [20], a mixed point addition (i.e. an addition where one point is given in extended projective coordinates and the other point in extended affine coordinates) costs just seven multiplications (7M) in $\mathbb{F}_p$ [26, 24]. Furthermore, doubling a point in extended projective coordinates requires four multiplications (4M) and three squarings (3M). Our fixed-base comb method (with eight pre-computed points) executes $n/4$ point doublings the same number of point additions, where n refers to the bitlength of the scalar. The overall cost of the fixed-base scalar multiplication sB amounts to $(7n + 3n)/4 = 2.5n$ multiplications and $4n/4 = 1n$ squarings in $\mathbb{F}_p$, i.e. $2.5\text{M} + 1\text{S}$ per bit of the scalar. Thanks to the Montgomery ladder, the variable-base scalar multiplication hA takes only $5\text{M} + 4\text{S}$ per scalar bit [4]. In summary, the overall cost of the separated approach to compute sB and hA is $7.5\text{M} + 5\text{S}$ per bit, which is only slightly (i.e. $1\text{M} + 1\text{S}$) worse than the average number of multiplications/squarings for the simultaneous technique. Both the simultaneous technique and the separated method also involve two inversions in $\mathbb{F}_p$, one at the beginning and one at the end of the scalar multiplication.

3.3 Compatibility with other ECC Libraries

The initial Ed25519 specification from [8, 9] does not mandate much validation of input data and is also relatively vague when it comes to dealing with certain "corner cases." In particular, Ed25519 as specified in [8, 9] does not validate the signer's public key A; it does not even carry out a *partial* public-key validation (by checking $cA \neq \mathcal{O}$ [1], where c is the cofactor, i.e. $c = \#E(\mathbb{F}_p)/\ell$) to ensure that A does not have low order. However, due to the lack of key validation, the Ed25519 signature scheme can not guarantee non-repudiation or resilience to key-substitution attacks (see [11, Sect. A] for an example). Another problem is the omission of clear guidance on how to handle corner cases, which has led to a number of Ed25519 variants, as well as inconsistencies and incompatibilities between implementations. As analyzed in e.g. [11, 16], existing implementations of variants or tweaks of Ed25519 differ with respect to the following aspects.

- whether a non-canonically encoded scalar s is accepted as valid input,
- whether non-canonically encoded points A, R are accepted as valid input,
- whether the points A, R are allowed to have low order,
- whether the verification procedure uses the cofactored ("batched") equation $8R = 8(sB - hA)$ or the more strict cofactorless equation $R = sB - hA$.

The specific way how an Ed25519 implementation deals with corner cases does not affect the verification of honestly-generated signatures, but can cause divergence when the signer (or an attacker) crafts a signature so that it is accepted by some implementations and rejected by others. This is especially problematic when an Ed25519 signature is verified by many entities seeking for a consensus (e.g. contract signing, electronic voting, blockchain transactions [11]).

Our software is compatible with the widely-used LibSodium library (version 1.0.16 or newer), which means it rejects an alleged signature when s, A or R is non-canonically encoded, or when A or R has low order. Any alleged signature passing these input checks is then verified using the cofactorless equation.

4 Experimental Results

The target platform of our performance assessment of the two implementation options for EdDSA verification described in the last section is the well-known and widely-used 16-bit MSP430 architecture from Texas Instruments. MSP430 microcontrollers were designed for extremely low power dissipation; this covers not only the active processing power, but also standby and memory read/write power, respectively [14]. Regarding the latter it should be noted that MSP430 devices were among the first to be equipped with Ferro-electric Random Access Memory (FRAM), which has similar attributes like SRAM (e.g. fast read and write operations, low power dissipation, high reliability and endurance), but is non-volatile, like EEPROM or flash memory, and can hold data even after it is powered off. This feature makes it relatively easy to switch from active mode to standby or sleep mode, thereby enabling energy savings even for short periods of inactivity, since data can simply remain in FRAM. For these reasons, Texas Instruments markets the MSP430 family as "ultra-low-power" microcontrollers to emphasize their suitability for the Internet of Things (IoT) [14].

The MSP430 uses the von-Neumann memory model, which means code and data share a unified address space, and there is a single address bus and single data bus that connects the CPU core with RAM, flash/ROM, and peripheral modules. Twelve out of a total of 16 registers (each 16 bits wide) are available for general use; the remaining four serve a special purpose. The MSP430 architecture has a reduced instruction set consisting of 27 core instructions that can be split into three categories: double-operand instructions (which overwrite one of the operands with the result), single-operand instructions, and jumps. This minimalist instruction set is orthogonal and supports seven addressing modes altogether, including modes for direct memory-to-memory transfers without an intermediate register holding [38]. The used addressing mode(s) determine the latency of double-operand instructions, which can vary between one clock cycle (when both source and destination operand are held in registers) and six clock cycles (operands are in RAM or in flash). Some MSP430 models, including the MSP430F1611 we use for our benchmarking, have a memory-mapped hardware multiplier capable to carry out (16×16)-bit multiply and multiply-accumulate operations [38]. Since this multiplier is a memory-mapped peripheral, it has to be accessed by writing the two operands to specific locations in memory. The MSP430F1611 is equipped with 10 kB RAM and 48 kB flash.

Our implementation of the field-arithmetic operations is a slightly modified and improved version of the ECC software for MSP430(X) devices introduced in [24]. This library is written in Assembly language and provides all low-level operations needed to perform point addition and doubling on Montgomery and TE curves, respectively. Since our target device is a 16-bit microcontroller, the elements of $\mathbb{F}_p$ are represented as arrays of (unsigned) 16-bit words, i.e. arrays of type `uint16_t`. Except for inversion, the arithmetic functions do not execute operand-dependent conditional jumps or branches (i.e. their execution time is constant), which contributes to preventing timing attacks against the signature generation. Although the verification of an EdDSA signature does not involve

Table 1. Execution time and binary code size of 255-bit field-arithmetic operations on an MSP430F1611 microcontroller.

Operation	Exec. time (cycles)	Code size (bytes)
Addition	322	100
Subtraction	332	140
Multiplication (incl. red.)	5388	352
Squaring (incl. red.)	3826	388
Mul. by 32-bit constant	1040	240
Inversion (incl. masking)	197102	942

any secret information, it still makes sense to use a constant-time $\mathbb{F}_p$-arithmetic library since it can be shared between the signature generation and verification functions. The $\mathbb{F}_p$-inversion of our library is based on the Extended Euclidean Algorithm (EEA), but uses a "multiplicative masking" technique to randomize the execution time and thwart timing attacks (see [24] for details).

Table 1 specifies the execution time (including function-call overhead) and code size of the most important operations of our $\mathbb{F}_p$-arithmetic library on an MSP430F1611 microcontroller. These timings are slightly better than the ones reported in [24], which is due to a couple of further Assembly optimizations we added to the source code. The code size of the full library for $\mathbb{F}_p$-arithmetic is just slightly more than 2.2 kB, which is very small compared to other MSP430 implementations, e.g. [2, 18, 23, 27, 32, 33]. This small code size became possible because our arithmetic library is not purely optimized for high performance (as most other libraries) but aims for a trade-off between size and speed.

Table 2. Execution time, RAM footprint, and binary code size (excluding the field arithmetic) of point-arithmetic operations on an MSP430F1611 microcontroller.

Operation	Exec. time (cycles)	RAM footpr. (bytes)	Code size (bytes)
Point addition (TE curve)	39718	72	272
Point doubling (TE curve)	33451	68	268
Point addition (Mon curve)	25811	132	220
Point doubling (Mon curve)	20776	128	184
Recovery Y coord. (Mon curve)	56117	96	302
Conversion Mon to TE	22519	124	116
Conversion TE to Mon	22521	124	112

Table 2 summarizes the execution time, RAM footprint, and (binary) code size of some point-arithmetic operations. Point addition and point doubling on a Montgomery curve is significantly faster than on a TE curve, which is little surprising since the projective point arithmetic on the former involves only the

Table 3. Execution time, RAM footprint, and binary code size of scalar multiplication and full EdDSA verification on an MSP430F1611 microcontroller.

Operation	Exec. time (cycles)	RAM footpr. (bytes)	Code size (bytes)
Table pre-computation (TE curve)	261926	612	288
Double-scalar mul. (TE curve)	14126254	878	674 + 2230
EdDSA verification (simultaneous)	14206712	980	6143
Fixed-base scalar mul. (TE curve)	4682599	596	602 + 2230
Variable-base scalar mul. (Mon curve)	12138929	478	1356 + 2230
EdDSA verification (separated)	17516534	596	7850

X and Z coordinate. The recovery of the projective Y coordinate is a bit more costly, but this operation is performed only once. The results for the code size in the right column cover only the size of the function itself and do not include sub-functions like the field-arithmetic operations (this makes sense because the field arithmetic is shared across all higher-level operations).

Finally, Table 3 compares the execution time, RAM footprint, and code size of the simultaneous method and the separated technique for double-scalar multiplication and full EdDSA signature verification, respectively. As expected, the separated technique is slower than the simultaneous method, but the difference (with respect to overall verification time) is relatively small, namely about 3.3 million clock cycles, which is approximately 24% of the verification time of the simultaneous method. On the other hand, the simultaneous method consumes almost 1 kB RAM, which is 394 bytes more than the amount of RAM needed for the separated technique. This significant difference can be explained by the fact that the separated method (i) does not need to store table with four pre-computed points in RAM, and (ii) also does not occupy RAM for storing the JSF representation of two scalars. The execution time of EdDSA verification is mainly dominated by the double-base scalar multiplication, which contributes more than 98% to the overall execution time when the message to be verified is small. The execution times for the entire EdDSA verification listed in Table 3 were determined with a message of a length of only a few bytes, which means the compression function of the SHA-512 hash function was executed only once to obtain the 512-bit digest. Our assembler implementation of the compression function has an execution time of about 38500 clock cycles, which is negligible compared to the double-scalar multiplication.

5 Conclusions

All major elliptic-curve signature schemes have in common that the verification of a signature requires much more computation time than its generation. Even worse, most existing implementation results reported in the literature indicate that verifying an EdDSA signature consumes significantly more RAM than the signing operation, which poses a serious problem for resource-restricted devices like sensor nodes that often have only a few kilobytes of RAM. The enormous computational cost and large RAM footprint of the verification is mainly due to the double-scalar multiplication $R = sB - hA$, which is normally implemented using the simultaneous method with joint doublings. In this paper we proposed an alternative approach that splits the computation of $sB - hA$ up into two separate operations: a fixed-base scalar multiplication sB and a variable-base scalar multiplication hA. By exploiting the birational equivalence between the twisted Edwards model and the Montgomery model, we compute the variable-base scalar multiplication with the fast Montgomery ladder. Our experiments show that, on a 16-bit MSP430F1611 microcontroller, the separated method is only 24% slower than the simultaneous method, but consumes about 40% less RAM, mainly because it does not need to store a table of pre-computed points and and also does not require a JSF-representation of the scalars. This makes the separated approach an attractive alternative to the simultaneous technique whenever RAM is a scarce resource.

Acknowledgements. Zhe Liu is supported by the National Key R&D Program of China (Grant No. 2020AAA0107703), the National Natural Science Foundation of China (Grants No. 62132008, 61802180), the Natural Science Foundation of Jiangsu Province (Grant No. BK20180421), and the National Cryptography Development Fund (Grant No. MMJJ20180105).

References

1. Antipa, A., Brown, D.R., Menezes, A.J., Struik, R., Vanstone, S.A.: Validation of elliptic curve public keys. In: Desmedt, Y.G. (ed.) Public Key Cryptography — PKC 2003. Lecture Notes in Computer Science, vol. 2567, pp. 211–223. Springer Verlag (2003)
2. Ateniese, G., Bianchi, G., Capossele, A.T., Petrioli, C.: Low-cost standard signatures in wireless sensor networks: A case for reviving pre-computation techniques? In: Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS 2013). pp. ??–?? The Internet Society (2013)
3. Bauer, J., Staudemeyer, R.C., Pöhls, H.C., Fragkiadakis, A.G.: ECDSA on things: IoT integrity protection in practise. In: Lam, K.Y., Chi, C.H., Qing, S. (eds.) Information and Communications Security — ICICS 2016. Lecture Notes in Computer Science, vol. 9977, pp. 3–17. Springer Verlag (2016)
4. Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) Public Key Cryptography — PKC 2006. Lecture Notes in Computer Science, vol. 3958, pp. 207–228. Springer Verlag (2006)
5. Bernstein, D.J.: Multi-user Schnorr security, revisited. Cryptology ePrint Archive, Report 2015/996 (2015). `http://eprint.iacr.org/2015/996`
6. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards curves. In: Vaudenay, S. (ed.) Progress in Cryptology — AFRICACRYPT 2008. Lecture Notes in Computer Science, vol. 5023, pp. 389–405. Springer Verlag (2008)
7. Bernstein, D.J., Chuengsatiansup, C., Lange, T.: Double-base scalar multiplication revisited. Cryptology ePrint Archive, Report 2017/037 (2017). `http://eprint.iacr.org/2017/037`

8. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems — CHES 2011. Lecture Notes in Computer Science, vol. 6917, pp. 124–142. Springer Verlag (2011)
9. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering **2**(2), 77–89 (Sep 2012)
10. Bundesamt für Sicherheit in der Informationstechnik (BSI): Elliptic Curve Cryptography. Technical Guideline TR-03111, available for download at `http://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03111/BSI-TR-03111_pdf.html` (2012)
11. Chalkias, K., Garillot, F., Nikolaenko, V.: Taming the many EdDSAs. In: van der Merwe, T., Mitchell, C.J., Mehrnezhad, M. (eds.) Security Standardisation Research — SSR 2020. Lecture Notes in Computer Science, vol. 12529, pp. 67–90. Springer Verlag (2020)
12. Cohen, H., Frey, G.: Handbook of Elliptic and Hyperelliptic Curve Cryptography, Discrete Mathematics and Its Applications, vol. 34. Chapmann & Hall\CRC (2006)
13. Costello, C., Smith, B.: Montgomery curves and their arithmetic. Journal of Cryptographic Engineering **8**(3), 227–240 (Sep 2018)
14. Dang, D., Plant, M., Poole, M.: Wireless connectivity for the Internet of Things (IoT) with MSP430 microcontrollers (MCUs). White paper, available for download at `http://www.ti.com/lit/wp/slay028/slay028.pdf` (Mar 2014)
15. de Meulenaer, G., Gosset, F., Standaert, F.X., Pereira, O.: On the energy cost of communication and cryptography in wireless sensor networks. In: Proceedings of the 4th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WIMOB 2008). pp. 580–585. IEEE Computer Society Press (2008)
16. de Valence, H.: It's 255:19AM. Do you know what your validation criteria are? Blog post, available online at `http://hdevalence.ca/blog/2020-10-04-its-25519am` (2020)
17. Düll, M., Haase, B., Hinterwälder, G., Hutter, M., Paar, C., Sánchez, A.H., Schwabe, P.: High-speed Curve25519 on 8-bit, 16-bit and 32-bit microcontrollers. Designs, Codes and Cryptography **77**(2–3), 493–514 (Dec 2015)
18. Gouvêa, C.P., Oliveira, L.B., López, J.: Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller. Journal of Cryptographic Engineering **2**(1), 19–29 (May 2012)
19. Hankerson, D.R., Menezes, A.J., Vanstone, S.A.: Guide to Elliptic Curve Cryptography. Springer Verlag (2004)
20. Hişil, H., Wong, K.K.H., Carter, G., Dawson, E.: Twisted Edwards curves revisited. In: Pieprzyk, J. (ed.) Advances in Cryptology — ASIACRYPT 2008. Lecture Notes in Computer Science, vol. 5350, pp. 326–343. Springer Verlag (2008)
21. Josefsson, S., Liusvaara, I.: Edwards-Curve Digital Signature Algorithm (EdDSA). Internet Research Task Force, Crypto Forum Research Group, RFC 8032 (Jan 2017)
22. Kiltz, E., Masny, D., Pan, J.: Optimal security proofs for signatures from identification schemes. In: Robshaw, M.J., Katz, J. (eds.) Advances in Cryptology — CRYPTO 2016. Lecture Notes in Computer Science, vol. 9815, pp. 33–61. Springer Verlag (2016)
23. Liu, A., Ning, P.: TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In: Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008). pp. 245–256. IEEE Computer Society Press (2008)

24. Liu, Z., Großschädl, J., Li, L., Xu, Q.: Energy-efficient elliptic curve cryptography for MSP430-based wireless sensor nodes. In: Liu, J.K., Steinfeld, R. (eds.) Information Security and Privacy — ACISP 2016. Lecture Notes in Computer Science, vol. 9722, pp. 94–112. Springer Verlag (2016)
25. Liu, Z., Longa, P., Pereira, G.C., Reparaz, O., Seo, H.: Four$\mathbb{Q}$ on embedded devices with strong countermeasures against side-channel attacks. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems — CHES 2017. Lecture Notes in Computer Science, vol. 10529, pp. 665–686. Springer Verlag (2017)
26. Liu, Z., Wenger, E., Großschädl, J.: MoTE-ECC: Energy-scalable elliptic curve cryptography for wireless sensor networks. In: Boureanu, I., Owezarski, P., Vaudenay, S. (eds.) Applied Cryptography and Network Security — ACNS 2014. Lecture Notes in Computer Science, vol. 8479, pp. 361–379. Springer Verlag (2014)
27. Marín, L., Pawlowski, M.P., Jara, A.J.: Optimized ECC implementation for secure communication between heterogeneous IoT devices. Sensors **15**(9), 21478–21499 (Sep 2015)
28. Möller, B.: Algorithms for multi-exponentiation. In: Vaudenay, S., Youssef, A.M. (eds.) Selected Areas in Cryptography — SAC 2001. Lecture Notes in Computer Science, vol. 2259, pp. 165–180. Springer Verlag (2001)
29. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**(177), 243–264 (Jan 1987)
30. National Institute of Standards and Technology (NIST): Digital Signature Standard (DSS). FIPS Publication 186-4, available for download at `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf` (Jul 2013)
31. Okeya, K., Sakurai, K.: Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems — CHES 2001. Lecture Notes in Computer Science, vol. 2162, pp. 126–141. Springer Verlag (2001)
32. Pabbuleti, K.C., Mane, D.H., Schaumont, P.: Energy budget analysis for signature protocols on a self-powered wireless sensor node. In: Saxena, N., Sadeghi, A.R. (eds.) Radio Frequency Identification: Security and Privacy Issues — RFIDSec 2014. Lecture Notes in Computer Science, vol. 8651, pp. 123–136. Springer Verlag (2014)
33. Pendl, C., Pelnar, M., Hutter, M.: Elliptic curve cryptography on the WISP UHF RFID tag. In: Juels, A., Paar, C. (eds.) RFID Security and Privacy — RFIDSec 2011. Lecture Notes in Computer Science, vol. 7055, pp. 32–47. Springer Verlag (2012)
34. Rescorla, E.K.: The Transport Layer Security (TLS) Protocol Version 1.3. Internet Engineering Task Force, Network Working Group, RFC 8446 (Aug 2018)
35. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM **21**(2), 120–126 (Feb 1978)
36. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO '89. Lecture Notes in Computer Science, vol. 435, pp. 239–252. Springer Verlag (1990)
37. Solinas, J.A.: Low-weight binary representations for pairs of integers. Tech. Rep. CORR 2001-41, Centre for Applied Cryptographic Research (CACR), University of Waterloo, Waterloo, Canada (2001)
38. Texas Instruments, Inc.: MSP430x1xx Family User's Guide (Rev. F). Manual, available for download at `http://www.ti.com/lit/ug/slau049f/slau049f.pdf` (Feb 2006)