Fast ECDH Key Exchange Using Twisted Edwards Curves with an Efficiently Computable Endomorphism

Johann Großschädl*, Zhe Liu*, Zhi Hu†, Chunhua Su‡, and Lu Zhou‡

*CSC and SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg

†School of Mathematics and Statistics, Central South University, Changsha, China

‡Division of Computer Science, University of Aizu, Aizuwakamatsu, Japan

***Abstract*—It is widely accepted that public-key cryptosystems play a major role in the security arena of the Internet of Things (IoT), but they need to be implemented efficiently to not deplete the scarce resources of battery-operated devices such as wireless sensor nodes. This paper describes a highly-optimized software implementation of scalar multiplication for Elliptic Curve Diffie-Hellman (ECDH) key exchange on resource-limited IoT devices that achieves fast execution times along with reasonably small code size and RAM consumption. Our software uses a special class of elliptic curves, namely twisted Edwards curves with an efficiently computable endomorphism similar to that of the so-called Gallant-Lambert-Vanstone (GLV) curves. This allows us to combine the main advantage of the GLV model, which is an efficiently-computable endomorphism to speed up variable-base scalar multiplication, with the fast and complete addition rules of the (twisted) Edwards model. We implemented variable-base scalar multiplication for static ECDH on two such curves, one over a 159-bit and the second over a 207-bit pseudo-Mersenne prime field, respectively, and evaluated their execution time on a 16-bit MSP430F1611 processor. The arithmetic operations in the prime field do not contain operand-dependent conditional statements (in particular no "if-then-else" clauses) and also the scalar multiplication follows a fixed execution path for a given (static) scalar. A variable-base scalar multiplication on curves over the 159 and 207-bit field takes about 2.63 and 4.84 million clock cycles, respectively, on an MSP430F1611 processor. These results compare favorably with the Montgomery ladder on the equivalent Montgomery curves, which is almost 50% slower.**

***Index Terms*—IoT security, Lightweight cryptography, ECDH key exchange, twisted Edwards curve, Endomorphism**

I. Introduction

The Internet of Things (IoT) is a frequently-used term to describe the currently ongoing evolution of the Internet into a network of smart objects ("things") that have the ability to communicate with each other and with centralized resources via the IPv6 (resp. 6LoWPAN) protocol [1]. Today, the two most important and widely noticed exponents of the IoT are RFID technology (which has become a key enabler of modern supply-chain management and industrial logistics) as well as Wireless Sensor Networks (WSNs), which have found widespread adoption in several application domains ranging from home automation over environmental surveillance and traffic control to medical monitoring. A recent white paper by Cisco estimates no less than 50 billion devices being connected to the Internet by the year 2020 [10], which implies that, in the near future, every person in the industrialized world will be surrounded by dozens of sensors, actuators, RFID tags, and many other kinds of smart objects yet to be developed. This evolution from the Internet of people to the Internet of things will have a profound impact on our daily life and change the way how we interact with the physical world surrounding us [1]. However, it is also evident that 50 billion smart devices connected to the Internet introduce unprecedented challenges to the security and privacy of their owners or users. On the one hand, each of these 50 billion devices is a potential target for a security attack over the Internet, similar to today's PCs and laptops. On the other hand, the plethora of IoT devices can also be (mis)used for a large-scale attack in the opposite direction, e.g. to trigger a denial of service.

Standardized security protocols, such as SSL and its more recent incarnation TLS, are an indispensable part of today's Internet infrastructure and have been integrated into modern web browsers, mail clients, and numerous other applications [8]. TLS allows two parties to establish an end-to-end secure (i.e. end-to-end authenticated and encrypted) communication channel over an insecure network like the Internet [28]. The emergence of the IoT quickly raised the question of how well TLS can be adapted for wireless networks that are characterized by relatively low bandwidth and high packet error rates (e.g. WSNs), for which connection-less datagram transport is simpler to implement and incurs less overhead than a reliable transport protocol such as TCP [19]. In order to facilitate the seamless expansion of common TLS-based security services into the IoT, the IETF developed the Datagram TLS (DTLS) protocol [29], a special variant of TLS to provide end-to-end security over unreliable datagram transport (e.g. UDP). This property, along with the fact that the full protocol stack can be implemented to fit into a few kB of RAM and run on an 8-bit microcontroller (as demonstrated in [28]), makes DTLS suitable for the IoT [19]. Both TLS and DTLS apply public-key cryptography in the handshake phase for authentication and key establishment. Since a large share of IoT devices are battery-powered and possess only very limited computational and energy resources, it makes sense to utilize Elliptic Curve Cryptography (ECC) [15] (i.e. ECDSA for digital signatures and ECDH for key exchange) instead of RSA.

The efficient implementation of ECC for resource-limited devices like smart cards and sensor nodes has been an active area of research in the past 20 years, yielding a considerable body of literature, e.g. [9], [20], [21], [25]. In 2008, Liu and Ning introduced TinyECC [21], a flexible and configurable

ECC library for wireless sensor nodes that has been used in countless WSN and IoT research projects and continues to be one of the most important lightweight ECC implementations to date. TinyECC was designed with the objective to achieve high performance, small code size, low memory (i.e. RAM) footprint, and the flexibility to support different curves given in Weierstraß form, including the NIST-recommended curve P192 [15]. However, the basic Weierstraß model is, in terms of efficiency and resistance to side-channel attacks, not state-of-the-art anymore, especially for ECDH key exchange, since a few "alternative" curve models with better implementation aspects exist. For example, Montgomery-form elliptic curves [27], such as the well-known Curve25519 [2], have a unique addition formula that enables one to efficiently compute the sum $P_1 + P_2$ of two points P_1, P_2 whose difference $P_1 - P_2$ is known. This so-called differential addition involves only the projective X and Z coordinate of the two points to be added (i.e. the Y coordinate is not used at all) and forms the basis of the Montgomery ladder for scalar multiplication, which is not only fast but also intrinsically secure from timing attacks and features a very low RAM footprint [9].

Another family of elliptic curves with excellent arithmetic features are twisted Edwards curves, introduced by Bernstein et al in 2008 [3]. Their most outstanding feature is a fast and complete addition law that requires just seven multiplications in the underlying prime field $\mathbb{F}_p$ to perform a mixed addition (i.e. a point addition where one point is given in projective coordinates and the other in affine coordinates) [6]. In comparison, a mixed point addition on a Weierstraß curve costs normally eight multiplications and three squarings in $\mathbb{F}_p$. Bos et al showed in [5] that twisted Edwards curves can achieve similar performance as Montgomery curves in variable-base scalar multiplication (or may even slightly outperform them at the 256-bit security level), but do so only at the expense of high RAM consumption. A third family of elliptic curves over prime fields for fast ECDH key exchange was presented by Gallant, Lambert, and Vanstone [13] at the IACR Crypto conference 2001. These so-called GLV elliptic curves are, in essence, Weierstraß curves over $\mathbb{F}_p$ with an efficiently-computable endomorphism, whereby the two most relevant versions are (i) curves with j-invariant $j = 1728$ and $p \equiv 1 \bmod 4$, and (ii) curves with $j = 0$ and $p \equiv 1 \bmod 3$. This endomorphism makes it possible to implement an n-bit scalar multiplication $k \cdot P$ through an operation of the form $k_1 \cdot P + k_2 \cdot Q$, where k_1, k_2 have a length of just about $n/2$ bits. Performing these two half-length scalar multiplication simultaneously based on "Shamir's trick" [15] eliminates half of the point doublings (and also a few point additions) compared to the double-and-add method using the n-bit scalar k. GLV curves outperform Montgomery curves in the variable-base setting, but are more difficult to protect against side-channel attacks [11].

The literature also contains a few papers with approaches to combine the arithmetic advantages of (twisted) Edwards curves and GLV curves, which resulted in (twisted) Edwards curves that are equipped with endomorphisms. Galbraith, Lin and Scott extended in [12] the GLV method to a larger class of elliptic curves, namely all curves defined over a quadratic extension field $\mathbb{F}_{p^2}$ that have their j-invariants belong to the base field $\mathbb{F}_p$. Any so-called GLS curve $E(\mathbb{F}_{p^2})$ features an endomorphism arising from the p-th power Frobenius map on a quadratic twist E' of E, which is very easy to compute in $\mathbb{F}_{p^2}$. Section 4 of [12] shows how this approach can be used with the twisted Edwards model for GLS curves and presents an explicit formula for the endomorphism. By exploiting this endomorphism along with a two-dimensional decomposition of k, similar to the GLV method, it is possible to reduce the number of point doublings in an n-bit scalar multiplication to roughly $n/2$. However, the GLS technique can be extended to dimension 4 (and even higher dimensions) by using curves over $\mathbb{F}_{p^2}$ with a large automorphism group and splitting the scalar into four $(n/4)$-bit parts. Hu et al [18] studied various implementation aspects of the 4-dimensional endomorphism-accelerated scalar multiplication on j-invariant-0 GLS curves (i.e. curves of the form $y^2 = x^3 + b$ over $\mathbb{F}_{p^2}$) and provided a decomposition method with a concrete bound on the lengths of the sub-scalars. According to the results they obtained on a Core2 processor, the 4-dimensional scalar multiplication on a j-invariant-0 GLS curve given in Weierstraß form is up to 27% faster than the basic 2-dimensional scalar multiplication on a comparable GLS curve in twisted Edwards form.

Longa and Sica [26] generalized Hu et al's 4-dimensional method from the family of j-invariant-0 curves to the twists of any GLV curve over $\mathbb{F}_{p^2}$ (called GLV-GLS curves). More concretely, they described how to "merge" the GLV and GLS approaches by combining the GLV-endomorphism ϕ and the GLS-endomorphism ψ to compute a scalar multiplication via the equation $kP = k_1P + k_2\phi(P) + k_3\psi(P) + k_4\psi\phi(P)$, where the length of k_1, k_2, k_3 and k_4 is only about a quarter of the length of k. Further contributions of [26] are an algorithm to decompose any $k \in [1, \ell]$ into four integers k_1, k_2, k_3, k_4 so that $\max_i(|k_i|) \leq c\ell^{1/4}$ for an explicit (small) constant c and techniques to speed up the GLV-GLS method by switching to the twisted Edwards model with $a = -1$. Recently, Costello and Longa [7] introduced Four$\mathbb{Q}$, a special twisted Edwards curve over the extension field $\mathbb{F}_{p^2}$ with $p = 2^{127} - 1$ that is equipped with two efficient endomorphisms (very similar to GLV-GLS curves) and, consequently, supports 4-dimensional scalar multiplication. Thanks to the very unique combination of a fast algorithm for scalar multiplication (by exploiting the two endomorphisms), fast point additions/doublings (enabled by the twisted Edwards form), and fast field operations (due to the Mersenne prime p), Four$\mathbb{Q}$ was able to set impressive new speed records for variable-base scalar multiplication on numerous platforms [24]. However, the superior performance of Four$\mathbb{Q}$ (and also GLV-GLS curves) comes at the expense of large code size and high RAM footprint. For example, the results in [20], [22] show that 4-dimensional scalar multiplication on Four$\mathbb{Q}$ or a similarly strong GLV-GLS curve takes over 2.5 kB RAM on an MSP430 processor, which is more than five times the RAM footprint of Curve25519 [16].

In this paper we analyze the suitability of twisted Edwards curves with an efficiently-computable endomorphism for the

IoT, in particular when implemented for constrained devices equipped with an MSP430 processor [30], whereby we focus on variable-base scalar multiplication. While previous papers on such "endomorphism curves" aimed primarily to increase performance, i.e. tried to answer the question of how *fast* an endomorphism-accelerated scalar multiplication can become (see e.g. [12] and [7]), we direct our attention to the question of how *small* (in terms of RAM consumption and code size) it can be implemented. Answering this question is crucial to get a better understanding of the trade-offs between execution time, memory footprint, and code size these curves offer. In addition, we strive to answer the question of whether curves with endomorphisms are still faster than Montgomery curves when one optimizes the scalar multiplication for small code size and low RAM usage instead of high speed. We resort to conventional GLV curves in twisted Edwards form since we consider them more "resource-friendly" than GLV-GLS and similar speed-oriented curves with two endomorphisms. Another argument against GLV-GLS curves is their dependence on quadratic extension fields, which are not supported by the vast majority of software libraries and hardware accelerators for ECC. The specific curve we use in our work is a twisted Edwards curve of the form $-x^2 + y^2 = 1 + x^2y^2$ over a prime field $\mathbb{F}_p$ of order $p = 2^k - c$ (i.e. a pseudo-Mersenne prime field). This curve is birationally-equivalent to the GLV curve $y^2 = x^3 + x/4$ and was used before by Liu et al [22] in the context of a hardware implementation of ECDSA signature verification. The present paper complements the work of Liu et al by showing the curve's suitability for efficient software implementation of (static) ECDH key exchange.

II. Preliminaries

In this section, we recap the basic properties and features of the two special families of elliptic curves our work is based on, namely GLV curves and twisted Edwards curves.

A. Gallant-Lambert-Vanstone (GLV) Curves

At Crypto 2001, Gallant et al [13] presented an ingenious method to accelerate scalar multiplication on elliptic curves equipped with an endomorphism ϕ whose characteristic polynomial has small coefficients. Such an endomorphism allows one to perform a scalar multiplication $k \cdot P$ by a computation of the form

$$k \cdot P = k_1 \cdot P + k_2 \cdot \phi(P) \tag{1}$$

where k_1 and k_2 are "small" scalars satisfying the equation $k = k_1 + k_2\lambda \bmod \ell$, λ is an integer root of the characteristic polynomial of ϕ modulo ℓ, and ℓ is the (prime) order of the point P. Even though Gallant et al did not give a concrete upper bound on the size of $|k_1|$ and $|k_2|$, it could be proven that $\max(|k_1|, |k_2|) \le c\sqrt{\ell}$ for some explicit (relatively small) constant c, see e.g. Theorem 1 in [26]. In other words, it is possible to "decompose" any n-bit scalar k into two scalars k_1 and k_2 of a length of just around $n/2$ bits. The two half-length scalar multiplications $k_1 \cdot P$ and $k_2 \cdot \phi(P)$ can be done in an interleaved fashion (as specified by Algorithm 3.51 in [15]) or simultaneously (Algorithm 3.48 in [15], sometimes referred to as "Shamir's trick"), which approximately halves the number of point doublings and also reduces the overall number of point additions in relation to the double-and-add method using the full-length (i.e. n-bit) scalar k.

Gallant et al discussed in [13] several examples of curves that come with an efficiently-computable endomorphism; the most important ones are Weierstraß curves with j-invariant 0 and 1728, respectively. We summarize both examples in the following. Let p be a prime with $p \equiv 1 \bmod 3$ and let E_1 be an elliptic curve over $\mathbb{F}_p$ given by

$$E_1 : \; y^2 = x^3 + b \tag{2}$$

Equation (2) is nothing else than a short Weierstraß equation with $a = 0$, which means the j-invariant of E_1 is 0. Let β be an element of order 3 in $\mathbb{F}_p$. Then, the map

$$\phi_1 : \; (x, y) \mapsto (\beta x, y), \; O \mapsto O \tag{3}$$

is an endomorphism of E_1 defined over $\mathbb{F}_p$ with characteristic polynomial ${\lambda_1}^2 + \lambda_1 + 1$. If $P = (x, y) \in E_1(\mathbb{F}_p)$ has prime order ℓ, then $\phi_1(P)$ corresponds to the scalar multiplication of P by λ_1, i.e. we have $\phi_1(P) = (\beta x, y) = \lambda_1 P$, where λ_1 is an integer satisfying ${\lambda_1}^2 + \lambda_1 + 1 = 0 \bmod \ell$. Note that $\phi_1(P)$ can be computed using a single multiplication in $\mathbb{F}_p$. Since the curve parameter $a = 0$, the formula for point doubling in Jacobian coordinates (see [15, page 90]) can be optimized to take only three multiplications and four squarings in $\mathbb{F}_p$.

Let p be a prime with $p \equiv 1 \bmod 4$ and let E_2 be an elliptic curve over $\mathbb{F}_p$ given by

$$E_2 : \; y^2 = x^3 + ax \tag{4}$$

Equation (4) is a (short) Weierstraß equation with $b = 0$, and consequently, the curve E_2 has j-invariant 1728. Let α be an element of order 4 in $\mathbb{F}_p$. Then, the map

$$\phi_2 : \; (x, y) \mapsto (-x, \alpha y), \; O \mapsto O \tag{5}$$

is an endomorphism of E_2 defined over $\mathbb{F}_p$ with characteristic polynomial ${\lambda_2}^2 + 1$. When $P = (x, y) \in E_2(\mathbb{F}_p)$ has prime order ℓ, then $\phi_2(P)$ corresponds to the scalar multiplication of P by λ_2, i.e. we have $\phi_2(P) = (-x, \alpha y) = \lambda_2 P$, where λ_2 is an integer satisfying ${\lambda_2}^2 + 1 = 0 \bmod \ell$. Note that $\phi_2(P)$ can be computed via a multiplication and a negation in $\mathbb{F}_p$. The Jacobian doubling on E_2 requires either four multiplications and four squarings in $\mathbb{F}_p$ (if E_2 is isomorphic or isogenious to the curve $x^3 - 3x$) or four multiplications (of which one is performed by $a \neq -3$) and six squarings [15, page 88].

B. Twisted Edwards (TE) Curves

Twisted Edwards curves (henceforth denoted as TE curves) were introduced by Bernstein et al in [3] as a generalization of ordinary (untwisted) Edwards curves. A TE curve defined over a prime field $\mathbb{F}_p$ is governed by the equation

$$E_T : \; ax^2 + y^2 = 1 + dx^2y^2 \tag{6}$$

where a and d are two distinct, non-0 elements of $\mathbb{F}_p$. Like Montgomery curves (see Section I and [27]), TE curves have a co-factor of (at least) 4; therefore, not every elliptic curve can be represented in TE form. Bernstein et al [3] were the first to describe an interesting connection between TE curves and Montgomery curves, namely their birational equivalence over $\mathbb{F}_p$: any TE curve given by Equation (6) is birationally-equivalent to a Montgomery curve specified by the equation $E_M: By^2 = x^3 + Ax^2 + x$ where

$$A = 2(a + d)/(a - d) \quad \text{and} \quad B = 4/(a - d). \tag{7}$$

The converse is also true: any Montgomery curve over $\mathbb{F}_p$ is birationally-equivalent over $\mathbb{F}_p$ to a TE curve. Bernstein et al provided the following map to send a point $P_t = (x_t, y_t)$ on E_T to the corresponding point $P_m = (x_m, y_m)$ on E_M.

$$(x_t, y_t) \mapsto (x_m, y_m) = \left(\frac{1 + y_t}{1 - y_t}, \frac{1 + y_t}{(1 - y_t)x_t}\right) \tag{8}$$

This map is not defined when $x_t = 0$ or $y_t = 1$. The inverse map to obtain $P_t \in E_T(\mathbb{F}_p)$ corresponding to $P_m \in E_M(\mathbb{F}_p)$ is

$$(x_m, y_m) \mapsto (x_t, y_t) = \left(\frac{x_m}{y_m}, \frac{x_m - 1}{x_m + 1}\right). \tag{9}$$

The above map is irregular at some points, namely the points P_m with $x_m = -1$ or $y_m = 0$ (see [3, Section 3.3] for a more detailed discussion of these exceptional points). A TE curve contains a *neutral element* in affine coordinates, which is the rational point $O = (0, 1)$. The point $(0, -1)$ has order 2. When the product ad is a square in $\mathbb{F}_p$, then there are two points of order 2 at infinity on the desingularization of E_T. If d is a square, then there are two points of order 4 at infinity on the desingularization of E_T [3].

The TE model features a unique addition law with various interesting implementation aspects. Given two (affine) points $P_1 = (x_1, y_1) \in E_T(\mathbb{F}_p)$ and $P_2 = (x_2, y_2) \in E_T(\mathbb{F}_p)$, their sum $P_1 + P_2$ can be computed as

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}\right). \tag{10}$$

This addition formula is *unified*, which means it can also be used to double a point. Furthermore, the addition formula is *complete* when a is a square in $\mathbb{F}_p$ and d is a non-square in $\mathbb{F}_p$. Completeness refers to the fact that the obtained sum is correct for any pair $P_1, P_2 \in E_T(\mathbb{F}_p)$, including corner cases like $P_1 = O$, $P_2 = O$, or $P_1 = -P_2$. On the other hand, if the curve parameters a, d do not meet these conditions, then the addition formula fails in certain cases, namely exactly when $P_1 - P_2$ is a point of order 2 or 4 at infinity. However, this implies that the addition formula given above always works correctly for point doubling (without exceptional cases) since $P_1 - P_1 = O$ has order 1 [4]. Hişil et al [17] introduced the so-called extended coordinates for TE curves, which enable extremely efficient point addition when $a = -1$. The addition formula from [17] is complete if a is a square and d a non-square in $\mathbb{F}_p$. However, a TE curve not satisfying these two conditions can still have a complete addition formula when using the extended coordinate system, provided the points to be added have both odd order [17, Theorem 1].

III. Twisted Edwards Curves with Endomorphism

Both the efficiency and security of ECDH key exchange is to a large extent determined by certain properties of the used elliptic curve. In the following, we describe a simple method to generate a secure TE curve with an efficiently-computable endomorphism. Furthermore, we discuss some peculiarities to take into account when this kind of TE curve is employed in an ECDH key exchange protocol.

A. Curve Generation

The generation of an elliptic curve for cryptographic usage requires one to take into account a multitude of security and efficiency criteria [5]. A major example for the former is the requirement that the group of rational points has to contain a large sub-group of prime order ℓ (or, equivalently, the co-factor $h = \#E(\mathbb{F}_p)/\ell$ should be small). In addition, the curve must not belong to one of the known classes of weak curves for which the ECDLP can be solved significantly faster than one would normally expect, e.g. anomalous curves or curves with a small embedding degree. A further requirement often mentioned in the literature, especially in the context of "x-coordinate-only" key exchange with a Montgomery curve, is twist security, which means that the curve and its quadratic twist are cryptographically strong [2], [25]. However, when using a TE curve, twist security is less crucial (at least from a theoretical perspective) since the point arithmetic involves both coordinates and public keys in ECDH key exchange are usually validated to ensure the incoming x and y coordinate form a point on the curve [7]. This validation is cheap; it is just a check whether x and y satisfy Equation (6).

With respect to efficiency, there are two basic requirements to consider in the curve generation, namely (i) the TE curve has to provide an efficiently-computable endomorphism, and (ii) the parameter a of the TE curve must be -1 to achieve maximum performance with Hişil et al's addition formula in extended coordinates [17]. The fact that any elliptic curve is representable through a Weierstraß equation enables a simple approach to find a suitable TE curve, which we sketch in the following. Given the parameters a and d of a TE curve, one can use Equation (7) to get the parameters A and B of the birationally-equivalent Montgomery curve. These parameters can, in turn, be used to compute a Weierstraß representation of the Montgomery curve. Combining the two computations leads to the following equations for obtaining the parameters a_w, b_w of a Weierstraß curve that is birationally-equivalent to the TE curve with the parameters a and d.

$$a_w = -(a^2 + 14ad + d^2)/48 \tag{11}$$

$$b_w = -(a + d)(a^2 - 34ad + d^2)/864 \tag{12}$$

Since endomorphisms are preserved across curve models, the two efficiency requirements can be easily met by fixing the TE parameter a to -1 and choosing d so that either $a_w = 0$ or $b_w = 0$. The right side of Equation (11) becomes 0 when $a^2 + 14ad + d^2 = d^2 - 14d + 1 = 0$, which is exactly the case for $d = 7 \pm 4\sqrt{3}$. However, a TE curve with these parameters can only exist if 3 is a square in the underlying prime field $\mathbb{F}_p$, i.e. if $p \equiv \pm 1 \bmod 12$. On the other hand, an inspection of Equation (12) reveals two basic options for choosing the TE parameter d so that b_w of the corresponding Weierstraß equation becomes 0. The first choice requires d to satisfy the equation $a^2 - 34ad + d^2 = d^2 + 34d + 1 = 0$, which means $d = 17 \pm 12\sqrt{2}$. These d parameters are elements of $\mathbb{F}_p$ when $p \equiv \pm 1 \bmod 8$. The second choice is $d = -a = 1$.

Of all these options for d, the last one (i.e. $d = 1$) is the best choice because it is least restrictive about the properties of the underlying field and provides an arithmetic advantage since most formulae for point addition (in particular those in [3] and [17]) involve a multiplication by d. In this way, we obtain a TE curve with $a = -1$ and $d = 1$, represented by the following equation

$$E_T : \; -x^2 + y^2 = 1 + x^2y^2. \tag{13}$$

The described approach for generating a TE curve is somewhat unusual since these specific values for a and d are the outcome of a process that took solely efficiency criteria into account, but no security requirements. However, a TE curve is not fully specified without an underlying field. Hence, the final task of our curve generation procedure is to obtain an appropriate prime field for which $a = -1$ and $d = 1$ defines a cryptographically strong TE curve. This contrasts with conventional techniques for curve generation where the field is normally chosen first and then one has to find suitable curve parameters for that specific field.

Most "lightweight" elliptic curve cryptosystems for use in the IoT provide security levels of 80, 96, 112, and 128 bits [25], though a few alternative levels were also proposed. An example is the ECDSA variant of Liu et al [22], which uses the TE curve defined by Equation (13) over a 207-bit prime field and targets a medium security level lying in the middle between 80 and 128 bits of security, respectively. More concretely, Liu et al based their ECDSA implementation on the pseudo-Mersenne prime $p = 2^{207} - 5131$ in order to facilitate the modular reduction operation. We use the same TE curve in this paper, plus an additional one over the smaller prime $p = 2^{159} - 7339$, providing roughly 80 bits of security. This means we evaluate the performance of ECDH key exchange with the following two TE curves.

$$E_{159} : \; -x^2 + y^2 = 1 + x^2y^2 \bmod 2^{159} - 7339 \tag{14}$$
$$E_{207} : \; -x^2 + y^2 = 1 + x^2y^2 \bmod 2^{207} - 5131 \tag{15}$$

Each of the two primes is congruent to 5 modulo 8 (i.e. the field $\mathbb{F}_p$ contains an element α of order 4) and the TE curve with $a = -1$ and $d = 1$ is birationally-equivalent over $\mathbb{F}_p$ to the GLV curve $y^2 = x^3 + x/4$, thereby implying the existence of an efficiently-computable endomorphism ϕ. Both curves have a co-factor of $h = 8$ and satisfy all further requirements a GLV curve needs to satisfy to be suitable for cryptographic purposes. However, when using the TE model, the concrete computation of ϕ differs slightly from Equation (5), which is only valid for the Weierstraß model. Liu et al presented in [22] the following equation for computing ϕ in TE form.

$$\phi : \; (x, y) \mapsto (\alpha x, 1/y) \tag{16}$$

The characteristic polynomial of ϕ is $\lambda^2 + 1$, which means this endomorphism corresponds to ϕ_2 on the j-invariant-1728 curve $y^2 = x^3 + ax$ described in Section II-A. The map given by Equation (16) fixes the identity element $O = (0, 1)$, but is obviously not defined when $y = 0$, i.e. for points of the form $(\pm 1/\sqrt{a}, 0)$. Such points only exist if a is a square in $\mathbb{F}_p$ and have order 4. For any other other point (x, y), it is trivial to show that $(\alpha x, 1/y)$ satisfies Equation (13), and is therefore indeed a point on the curve, since $\alpha^2 = -1$. Equation (16) is fairly easy to derive from ϕ_2 by simply mapping the points $P = (x, y)$ and $Q = \phi_2(P) = \lambda P = (-x, \alpha y)$ on the GLV curve $y^2 = x^3 + x/4$ to the corresponding ones on the birationally-equivalent TE curve. A similar equation was presented before in [12, Section 4.1] for computing an automorphism on the classical (un-twisted) Edwards curve $x^2 + y^2 = 1 - x^2y^2$. The endomorphism ϕ in the TE model is more costly than ϕ_2 in the GLV model (since it involves the inversion of y), but can still be computed efficiently enough to be beneficial [22]. In addition, it is possible to avoid the inversion by switching to projective coordinates, as will be shown in Section IV.

B. Secure ECDH Key Exchange

ECDH key exchange with TE curves, including TE curves with an efficiently-computable endomorphism, differs from the x-coordinate-only key exchange with Montgomery curves (e.g. Curve25519) and also from key exchange using prime-order Weierstraß curves (e.g. NIST curves) in that it requires protection against attacks based on invalid points. Examples of such invalid points are points not lying on the given curve and points outside the prime-order subgroup, which includes points of low order. Fortunately, ECDH key exchange can be easily and effectively protected against these *small subgroup attacks* and *invalid curve attacks*.

When using a Montgomery curve, it is possible to perform an entire key exchange with only the x-coordinates of points [2], which does not hold true for any other curve model. In particular, when implementing ECDH with a TE curve, it is necessary to transmit both the x and the y-coordinate of the point representing the public key (or, at least, one bit of the y-coordinate if point compression is applied) since both are needed as input for the scalar multiplication. The first step in an ECDH key exchange is to validate the received point; this consists of (i) checking that the x and y-coordinate are in the range of $[0, p - 1]$, and (ii) checking that x and y satisfy the curve equation, i.e. Equation (13), which costs two squarings and a multiplication in $\mathbb{F}_p$. Accepting points not lying on the correct elliptic curve can have serious security consequences and may, in the worst case, leak the full secret key.

Even if an incoming point satisfies the curve equation, it is not guaranteed that the point belongs to the right subgroup and has prime order $\ell = \#E(\mathbb{F}_p)/h$. In general, when using a TE curve with co-factor $h = 8$, a point $P \in E(\mathbb{F}_p)$ can have order 1, 2, 4, 8, ℓ, 2ℓ, 4ℓ, or 8ℓ. Points of low order require special attention to thwart small subgroup attacks, which can reveal (parts of) the secret scalar k. Depending on the curve parameters a and d, as well as the concrete implementation of the point addition, it may also be necessary to pay special attention to points of order 2ℓ, 4ℓ, and 8ℓ to ensure that the result of the scalar multiplication is correct. As mentioned in Section II-B, the TE addition law from [3] is complete when a is a square and d is a non-square in $\mathbb{F}_p$, which is not the case for our TE curve since $d = 1$ is a square. However, the fast addition formulae presented by Hişil et al in [17] can be complete even if a is not a square or d is a square, provided the points to be added are of odd order. This requirement is always met in the course of a scalar multiplication kP when the base point P has prime order ℓ. However, verifying this property requires one to compute ℓP and check whether the result is O. A less costly approach to ensure completeness is *co-factor multiplication*, which not only prevents exceptions in the addition formulae caused by even-order points, but also thwarts small subgroup attacks. So, instead of computing the scalar multiplication kP as usual, we first multiply P by the co-factor, i.e. we compute $P' = hP = 8P$, and then use P' as base point for the scalar multiplication by k, which gives as result $kP' = khP = 8kP$. Of course, each of the two parties involved in the ECDH key exchange needs to carry out this cofactor multiplication to get the correct shared secret[1].

Co-factor multiplication is a relatively cheap operation; in our case of $h = 8$, it requires just three point doublings. The software implementation for 16-bit MSP430 microcontrollers we present in the next section employs the projective addition formulae given in [3, Section 6], which are unified and can therefore be used to double a point. As explained in Section II-B, these formulae always produce the correct result for the doubling operation, even if a is non-square or d is a square in $\mathbb{F}_p$. For $a = -1$ and $d = 1$, the computational cost of these formulae is 10 multiplications (10M), one squaring (1S), and six additions or subtractions (6A) in the underlying field. The point $P' = hP$ is either the neutral element O (this happens when $\mathrm{ord}(P) \leq h$) or has prime order ℓ, which guarantees the completeness of Hişil et al's addition formulae in the course of a scalar multiplication [17]. Even though P' is obtained in projective (X, Y, Z) coordinates, determining whether it is the neutral element is easy; we just have to check whether $X = 0$ and $Y = Z$. If this is the case, then the key exchange should be aborted since the public key P has low order.

IV. Implementation Details and Results

The total execution time of ECDH key exchange depends not only on the algorithm for scalar multiplication, but also on the efficiency of the field and point arithmetic.

A. Field Arithmetic

As mentioned in Section III-A, the two TE curves we use for prototype implementation and performance evaluation are defined over pseudo-Mersenne primes, i.e. primes of the form $p = 2^k - c$, where c is small in relation to 2^k (typically, c is chosen to fit in a single register of the target platform). The two specific primes of these TE curves are the 159-bit prime $p = 2^{159} - 7339$ and the 207-bit prime $p = 2^{207} - 5131$. Since our target device is a 16-bit microcontroller, namely a Texas Instruments MSP430F1611 [30], we represent the elements of $\mathbb{F}_p$ as arrays of unsigned 16-bit words, i.e. arrays of type `uint16_t`. Our implementation of the arithmetic operations in $\mathbb{F}_p$ is a slightly modified and improved version of the ECC library for MSP430 microcontrollers introduced in [23]. This library is written in Assembly language and provides various low-level field operations needed to perform point additions and point doublings on TE curves. All arithmetic functions are parameterized with respect to the length of the operands (i.e. number of 16-bit words they consist of), which makes it possible to support operands of different length (in steps of 16 bits) without recompilation. Except for inversion, the arithmetic functions do not execute operand-dependent conditional jumps or branches (i.e. they have constant execution time), which helps to prevent timing attacks. The inversion is based on the Extended Euclidean Algorithm (EEA), but uses an ordinary "multiplicative masking" technique as protection against timing attacks. This means that we first multiply the field element Z to be inverted by a random number R, then invert the product, and finally multiply $(ZR)^{-1}$ again by R to get Z^{-1}. Even though such a masked inversion does not have constant execution time, it can still resist timing attacks.

TABLE I
Execution time and code size of arithmetic operations in a 159-bit and a 207-bit prime field on an MSP430F1611.

Operation	Execution time (cycles)		Code size (bytes)
	159-bit field	207-bit field	
Addition	220	268	100
Subtraction	228	276	140
Multiplication (incl. red.)	2430	3762	352
Squaring (incl. red.)	1942	2812	388
Inversion	129528	202632	942

Table I summarizes the execution time (including function call overhead) and code size of various arithmetic operations in $\mathbb{F}_p$ on a TI MSP430F1611 microcontroller. The timings for the 159-bit field are slightly better than that reported in [23], which is due to some low-level assembly optimizations we added to the source code. An in-depth description of the implementation of the $\mathbb{F}_p$-arithmetic can be found in [23].

B. Point Arithmetic

Our implementation of the point operations performed in the course of a scalar multiplication uses a slightly modified variant of Hişil et al's extended projective coordinates from [17]. More specifically, we represent a point P in extended twisted Edwards coordinates through a quintuple of the form $(X : Y : E : H : Z)$, where $EH = T = XY/Z$, the fourth coordinate (similar to [6] and [14]). On the other hand, a point in basic affine coordinates $P = (x, y)$ takes the form of a triple

[1]The leakage of key-bits due to points of low order can also be prevented by multiplying $k \in [1, \ell - 1]$ by $h = 8$ or by generating k as in [2] so that its three least significant bits are 0. However, while both methods thwart small subgroup attacks, they do not guarantee exception-free point arithmetic.

(u, v, w) with $u = (y + x)/2$, $v = (y - x)/2$, and $w = dxy$ as described in e.g. [14]. However, since the parameter d of the TE curves we use is always 1, the coordinate w is simply the product xy. Using these representations, a "mixed" addition (i.e. an addition of a point in extended projective coordinates with a point given in extended affine coordinates) based on the unified formulae from [17, Section 3.1] costs only seven multiplication (7M) and six additions or subtractions (6A) in $\mathbb{F}_p$. Doubling a point given in our special extended projective coordinates costs 3M, 4S, and 6A. As mentioned in Section III-B, we also need a "conventional" projective point-addition function for co-factor multiplication (and another task to be discussed below), which has a cost of 10M, 1S, and 6A.

On our MSP430F1611 microcontroller, the execution time of a mixed addition on the 159-bit curve is 18485 cycles; the doubling is about 10.8% faster and takes 16488 cycles. The corresponding cycle counts for addition and doubling on the 207-bit curve are 28097 and 24252, respectively.

C. Scalar Multiplication

Algorithm 1 shows the main steps to compute a variable-base scalar multiplication (including co-factor multiplication) $R = 8kP$ on a TE curve $E : -x^2 + y^2 = 1 + x^2y^2$ that features an efficiently-computable endomorphism ϕ.

Algorithm 1. Scalar multiplication on TE curve with endomorphism

```
Input: Scalar $k$, point $P = (x, y)$ on TE curve with $a = -1$, $d = 1$,
    and co-factor $h = 8$.
Output: Scalar product $R = 8kP$.
 1: $(k_1, k_2) \leftarrow$ DecomposeScalar($k$)
 2: $(l, m) \leftarrow$ JointSparseForm($k_1, k_2$)
 3: if $(-x^2 + y^2 \neq 1 + x^2y^2)$ then return $O$
 4: $Q \leftarrow 8P$                          {co-factor multiplication}
 5: if $Q = O$ then return $O$
 6: $S \leftarrow \phi(Q)$                     {$S$ is in projective coordinates}
 7: $T \leftarrow [S, Q - S, Q, Q + S]$        {table with 4 projective points}
 8: $T \leftarrow$ ProToExtAff($T$)            {table with 4 ext. affine points}
 9: $R \leftarrow O$
10: for $i$ from Max($|l|, |m|$) by 1 down to 0 do
11:     $R \leftarrow 2R$
12:     $k_i \leftarrow 3l_i + m_i$
13:     if $(k_i > 0)$ then $R \leftarrow R + T[i - 1]$ end if
14:     if $(k_i < 0)$ then $R \leftarrow R - T[$Abs$(i) - 1]$ end if
15: end for
16: $R \leftarrow$ ProToAff($R$)
17: return $R$
```

At the beginning, the n-bit scalar k is "decomposed" into two sub-scalars k_1 and k_2 (each of which is roughly $n/2$ bits long) such that $k = k_1 + k_2 \cdot \lambda \bmod \ell$, similar as discussed in Section II-A for ordinary GLV curves. This decomposition is a quite simple operation and can be carried out as described in e.g. [13], [15], [11], whereby the latter reference provides a secure (i.e. timing-attack-resistant) version. Thereafter, the two half-length scalars (k_1, k_2) are converted to the so-called *Joint Sparse Form (JSF)*, a special and unique representation of pairs of integers using the digit set $\{-1, 0, 1\}$ to minimize their joint Hamming weight, i.e. the overall number of non-0 columns. In this way, the JSF representation (which can be computed according to Algorithm 3.50 in [15]) reduces the overall number of point additions to be executed in the main loop starting at line 10. Then, we check whether P satisfies the curve equation so as to thwart invalid point attacks. The next step is the co-factor multiplication $Q = 8P$, which we carry out through three consecutive point doublings using the projective addition formulae from [3, Section 6]. As pointed out in Section II-B, these formulae are "unified" and always produce the correct result when used for point doubling. The rest of Algorithm 1 (i.e. from line 6 onwards) is similar to scalar multiplication on a conventional GLV curve, with the exception that Q is given in projective coordinates.

In line 6 we obtain the projective point S by applying the endomorphism ϕ of our TE curve to $Q = (X_q : Y_q : Z_q)$ using a projective version of Equation (16) as follows.

$$(\alpha x, 1/y) = (\alpha X_q/Z_q, Z_q/Y_q) = (\alpha X_q Y_q : Z_q^2 : Y_q Z_q) \quad (17)$$

In this way, the computation of ϕ costs three multiplications (3M) and one squaring (1S) in the field $\mathbb{F}_p$, but (unlike the affine formula for ϕ) does not involve an expensive inversion anymore. The next step is to compute a table T that contains the points Q and S, as well as their sum and difference, so that $T[0] = S$, $T[1] = Q - S$, $T[2] = Q$, and $T[3] = Q + S$. We use again the projective addition formulae from [3] to obtain $Q + S$ and $Q - S$. The points in T have to be converted from projective to extended affine coordinates in order to be able to exploit Hisil et al's fast (7M) mixed addition in the main loop. To simplify this operation, we first transform the table so that all points have the same projective Z coordinate; this costs 15M altogether. Then, we just have to invert one single Z coordinate, obtain affine (x, y) coordinates for each of the four points in T, and compute the extended affine coordinates (u, v, w), which requires further 12 multiplications (12M) plus an inversion (1I). Hence, it is possible to compute both the endomorphism $S = \phi(Q)$ and the table T with one inversion altogether, which is also necessary for ordinary GLV curves to obtain $Q + S$ and $Q - S$ in affine coordinates.

The rest of Algorithm 1 (i.e. the main loop from line 10 to 15 and the conversion of the result from projective to affine coordinates) is basically the same as for ordinary GLV curves (see [15, Section 3.5] for more details). In essence, the main loop carries out a simultaneous double-scalar multiplication $R = lQ + mS$ with joint doublings ("Shamir's trick") based on a JSF representation of the scalars l and m.

D. Performance Evaluation

As mentioned in Section II-A, the decomposition of an n-bit scalar k (in line 1 of Algorithm 1) yields two sub-scalars k_1 and k_2, each of which is roughly $n/2$ bits long. Also the JSF representation of k_1, k_2 (denoted l, m in Algorithm 1) has a length of approximately $n/2$ digits, whereby a digit can be either -1, 0, or 1 [15]. Due to the properties of the JSF, the joint Hamming weight of l and m (i.e. the number of non-0 columns of l, m) is, on average, $n/4$ in our case. This means that the main loop performs roughly $n/4$ point additions (on average) and about $n/2$ point doublings (e.g. 40 additions and

80 doublings when using the curve over our 159-bit field). In summary, the computational cost of the main loop amounts to 7M/4 + (3M + 4S)/2 = 3.25M + 2S per scalar bit, which is significantly below the 5M + 4S of the Montgomery ladder on a Montgomery curve.

TABLE II
EXECUTION TIME, RAM FOOTPRINT, AND CODE SIZE OF VARIABLE-BASE SCALAR MULTIPLICATION ON AN MSP430F1611.

Curve	Exec. time	RAM footpr.	Code size
	cycles	bytes	bytes
Our 159-bit TE curve	2631265	672	5680
159-bit Montg. curve	3800509	352	4132
Our 207-bit TE curve	4840407	834	5680
207-bit Montg. curve	7223444	426	4132

Table II summarizes the execution time, RAM footprint, and code size of scalar multiplication on our TE curves and the birationally-equivalent Montgomery curves over the 159 and 207-bit field, respectively. As expected, the two TE curves outperform their Montgomery counterparts significantly; the speed-up factor is about 1.44 for the 159-bit curve and even slightly bigger (namely 1.49) for the stronger curve over the 207-bit prime field. However, the TE timings do not include the decomposition of k and the computation of the JSF since both is not needed for static ECDH (instead of storing k we simply store l and m). The downside of TE curves with an efficiently-computable endomorphism is their relatively large demand for RAM, which exceeds that of Montgomery curves by between 91% (159-bit) and 96% (207-bit). This increased memory footprint is mainly due to the extended coordinates and the table T containing four points. The binary code size is roughly 37% larger (5680 vs. 4132 bytes).

Our implementation of scalar multiplication on TE curves with efficient endomorphism can withstand timing attacks in a static ECDH setting since, when k is fixed, the main loop of Algorithm 1 always executes exactly the same sequence of additions and doublings. In summary, our results indicate that TE curves with an efficiently-computable endomorphism can be an excellent alternative to Montgomery curves.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[2] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *Public Key Cryptography — PKC 2006*. Springer, 2006, pp. 207–228.

[3] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards curves," in *Progress in Cryptology — AFRICACRYPT 2008*. Springer, 2008, pp. 389–405.

[4] D. J. Bernstein and T. Lange, "A complete set of addition laws for incomplete Edwards curves," *Journal of Number Theory*, vol. 131, no. 5, pp. 858–872, May 2011.

[5] J. W. Bos, C. Costello, P. Longa, and M. Naehrig, "Selecting elliptic curves for cryptography: An efficiency and security analysis," *Journal of Cryptographic Engineering*, vol. 6, no. 4, pp. 259–286, Nov. 2016.

[6] D. Chu, J. Großschädl, Z. Liu, V. Müller, and Y. Zhang, "Twisted Edwards-form elliptic curve cryptography for 8-bit AVR-based sensor nodes," in *Proceedings of the 1st ACM Workshop on Asia Public-Key Cryptography (AsiaPKC 2013)*. ACM Press, 2013, pp. 39–44.

[7] C. Costello and P. Longa, "Fourℚ: Four-dimensional decompositions on a ℚ-curve over the Mersenne prime," in *Advances in Cryptology — ASIACRYPT 2015*. Springer, 2015, pp. 214–235.

[8] T. Dierks and E. K. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," IETF Network Working Group, RFC 5246, 2008.

[9] M. Düll, B. Haase, G. Hinterwälder, M. Hutter, C. Paar, A. H. Sánchez, and P. Schwabe, "High-speed Curve25519 on 8-bit, 16-bit and 32-bit microcontrollers," *Designs, Codes and Cryptography*, vol. 77, no. 2–3, pp. 493–514, Dec. 2015.

[10] D. Evans, "The Internet of things: How the next evolution of the Internet is changing everything," Cisco IBSG white paper, available for download at http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf, Apr. 2011.

[11] A. Faz-Hernández, P. Longa, and A. H. Sánchez, "Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV-GLS curves," in *Topics in Cryptology — CT-RSA 2014*. Springer, 2014, pp. 1–27.

[12] S. D. Galbraith, X. Lin, and M. Scott, "Endomorphisms for faster elliptic curve cryptography on a large class of curves," *Journal of Cryptology*, vol. 24, no. 3, pp. 446–469, Jul. 2011.

[13] R. P. Gallant, R. J. Lambert, and S. A. Vanstone, "Faster point multiplication on elliptic curves with efficient endomorphism," in *Advances in Cryptology — CRYPTO 2001*. Springer, 2001, pp. 190–200.

[14] M. Hamburg, "Fast and compact elliptic-curve cryptography," Cryptology ePrint Archive, Report 2012/309, 2012, available for download at http://eprint.iacr.org.

[15] D. R. Hankerson, A. J. Menezes, and S. A. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[16] G. Hinterwälder, A. Moradi, M. Hutter, P. Schwabe, and C. Paar, "Full-size high-security ECC implementation on MSP430 microcontrollers," in *Progress in Cryptology — LATINCRYPT 2014*. Springer, 2015, pp. 22–38.

[17] H. Hişil, K. K.-H. Wong, G. Carter, and E. Dawson, "Twisted Edwards curves revisited," in *Advances in Cryptology — ASIACRYPT 2008*. Springer, 2008, pp. 326–343.

[18] Z. Hu, P. Longa, and M. Xu, "Implementing the 4-dimensional GLV method on GLS elliptic curves with j-invariant 0," *Designs, Codes and Cryptography*, vol. 63, no. 3, pp. 331–343, Jun. 2012.

[19] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "DTLS based security and two-way authentication for the Internet of things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2710–2723, Nov. 2013.

[20] J. Kwon, S. C. Seo, and S. Hong, "Efficient implementations of four-dimensional GLV-GLS scalar multiplication on 8-bit, 16-bit, and 32-bit microcontrollers," *Applied Sciences*, vol. 8, no. 6, p. 900, Jun. 2018.

[21] A. Liu and P. Ning, "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)*. IEEE Computer Society Press, 2008, pp. 245–256.

[22] Z. Liu, J. Großschädl, Z. Hu, K. Järvinen, H. Wang, and I. Verbauwhede, "Elliptic curve cryptography with efficiently computable endomorphisms and its hardware implementations for the Internet of things," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 773–785, May 2017.

[23] Z. Liu, J. Großschädl, L. Li, and Q. Xu, "Energy-efficient elliptic curve cryptography for MSP430-based wireless sensor nodes," in *Information Security and Privacy — ACISP 2016*. Springer, 2016, pp. 94–112.

[24] Z. Liu, P. Longa, G. C. Pereira, O. Reparaz, and H. Seo, "Fourℚ on embedded devices with strong countermeasures against side-channel attacks," in *Cryptographic Hardware and Embedded Systems — CHES 2017*. Springer, 2017, pp. 665–686.

[25] Z. Liu, E. Wenger, and J. Großschädl, "MoTE-ECC: Energy-scalable elliptic curve cryptography for wireless sensor networks," in *Applied Cryptography and Network Security — ACNS 2014*. Springer, 2014, pp. 361–379.

[26] P. Longa and F. Sica, "Four-dimensional Gallant-Lambert-Vanstone scalar multiplication," in *Advances in Cryptology — ASIACRYPT 2012*. Springer, 2012, pp. 719–739.

[27] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of Computation*, vol. 48, no. 177, pp. 243–264, Jan. 1987.

[28] V. Perelman, "Security in IPv6-enabled wireless sensor networks: An implementation of TLS/DTLS for the Contiki operating system," M.Sc. Thesis, Jacobs University, Bremen, Germany, Jun. 2012.

[29] E. K. Rescorla and N. G. Modadugu, "Datagram Transport Layer Security Version 1.2," IETF Network Working Group, RFC 6347, 2012.

[30] 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.