MoTE-ECC: Energy-Scalable Elliptic Curve Cryptography for Wireless Sensor Networks

Zhe Liu[1], Erich Wenger[2], and Johann Großschädl[1]

[1] University of Luxembourg,
Laboratory of Algorithmics, Cryptology and Security (LACS),
6, rue Richard Coudenhove-Kalergi, L–1359 Luxembourg
{zhe.liu,johann.groszschaedl}@uni.lu
[2] Graz University of Technology,
Institute for Applied Information Processing and Communications,
Inffeldgasse 16a, A–8010 Graz, Austria
erich.wenger@iaik.tugraz.at

Abstract. Wireless Sensor Networks (WSNs) are susceptible to a wide range of malicious attacks, which has stimulated a body of research on "light-weight" security protocols and cryptographic primitives that are suitable for resource-restricted sensor nodes. In this paper we introduce MoTE-ECC, a highly optimized yet scalable ECC library for Memsic's MICAz motes and other sensor nodes equipped with an 8-bit AVR processor. MoTE-ECC supports scalar multiplication on Montgomery and twisted Edwards curves over Optimal Prime Fields (OPFs) of variable size, e.g. 160, 192, 224, and 256 bits, which allows for various trade-offs between security and execution time (resp. energy consumption). OPFs are a special family of "low-weight" prime fields that, in contrast to the NIST-specified fields, facilitate a parameterized implementation of the modular arithmetic so that one and the same software function can be used for operands of different length. To demonstrate the performance of MoTE-ECC, we take (ephemeral) ECDH key exchange between two nodes as example, which requires each node to execute two scalar multiplications. The first scalar multiplication is performed on a fixed base point (to generate a key pair), whereas the second scalar multiplication gets an arbitrary point as input. Our implementation uses a fixed-base comb method on a twisted Edwards curve for the former and a simple ladder approach on a birationally-equivalent Montgomery curve for the latter. Both scalar multiplications require about $9 \cdot 10^6$ clock cycles in total and occupy only 380 bytes in RAM when the underlying OPF has a length of 160 bits. We also describe our efforts to harden MoTE-ECC against side-channel attacks (e.g. simple power analysis) and introduce a highly regular implementation of the comb method.

1 Introduction

Some ten years ago, an article in MIT's Technology Review magazine identified Wireless Sensor Networks (WSNs) as one of the technologies that will change

the world in the 21st century [6]. This prediction could not have been more true given today's omnipresence of wireless sensors in various kinds of applications ranging from medical monitoring over home automation to environmental surveillance [1]. All these applications rely on distributed sensor nodes being able to collect, process and transmit data correctly and reliably, which has initiated a large body of research on the security of WSNs. Unfortunately, a WSN is, in general, harder to protect than a "traditional" (i.e. wired) network like e.g. an Ethernet-based LAN, which has two major reasons. First, the wireless nature of communication within a WSN makes eavesdropping fairly easy. Second, the nodes themselves can be subject to an attack since WSNs are often deployed in unattended areas. An attacker could, for example, capture one or more nodes and compromise them to obtain all stored data, or he may even reprogram the nodes and inject them into the WSN to conduct malicious activities [25].

Similar to conventional networks (e.g. the Internet), Public-Key Cryptography (PKC) can play a vital role in the security arena of WSNs [25]. The main problem with the practical use of PKC are the extremely constrained resources of battery-powered sensor nodes. For example, the prevalent MICAz mote from Memsic [8] is equipped with an 8-bit AVR processor (the ATmega128 [3]) and features only 4 kB of RAM and 128 kB flash memory. Gura et al [13] were the first to demonstrate that Elliptic Curve Cryptography (ECC) [15] is feasible on such restricted 8-bit platforms. Thanks to their so-called hybrid multiplication technique (a smart optimization of long-integer multiplication by exploiting the large register file of the ATmega128), they managed to reach an execution time of just $6.4 \cdot 10^6$ clock cycles for a full scalar multiplication on a SECG-specified elliptic curve over a 160-bit generalized-Mersenne prime field. TinyECC [23] is the currently most-widely used ECC library for WSNs; it is highly configurable and features a number of optimizations for "standardized" curves over 160 and 192-bit prime fields. Other examples of highly-optimized ECC implementations for 8-bit AVR processors are WM-ECC [33], Nano-ECC [31], MIRACL [7], and RELIC [2]. Very recently, it was shown that even high-security ECC using an elliptic curve over a 255-bit pseudo-Mersenne prime field is feasible on the ATmega128 [17]. However, as noted in [31], the feasibility of ECC on constrained devices does not automatically imply that it is attractive to use ECC since the state-of-the-art in terms of performance is still not satisfactory for many kinds of application. Therefore, the efficient implementation of ECC on sensor nodes remains an active research topic and approaches to further reduce the execution time (i.e. energy cost) and memory footprint are still eagerly sought.

In this paper we introduce MoTE-ECC, a light-weight ECC implementation for Memsic's MICAz motes and other 8-bit AVR-based sensor nodes (or, more generally, embedded devices equipped with an 8-bit AVR processor). The main goal we aimed to achieve with the design and implementation of MoTE-ECC was to find a suitable compromise between the following four requirements: (1) short execution times, (2) high flexibility and scalability (i.e. support of curves providing different levels of security), (3) low memory (i.e. RAM) footprint, and (4) some basic protection against passive implementation attacks. Energy is, in

general, the most precious resource of a battery-powered sensor node. Therefore, it is important to optimize the performance of ECC software because the energy consumption of scalar multiplication grows linearly with the execution time. Another essential requirement of an ECC implementation for WSNs is to support curves of different order (i.e. different cryptographic "strength") since the various tasks a sensor node performs during its lifetime have very different security needs [30, 22]. For example, a multi-tier security framework for WSNs can permit lower security levels (i.e. shorter keys) for some less-critical tasks in order to save energy. High flexibility at the field-arithmetic layer is difficult to achieve with the NIST-specified generalized Mersenne primes as each of them requires a different reduction routine (see [15, Section 2.2.6]). Consequently, an implementation supporting all five NIST fields needs five different modular reduction functions, which massively bloats code size. MoTE-ECC uses so-called *Optimal Prime Fields (OPFs)*, a special family of fields that allows for flexible yet efficient modular arithmetic [24, 34]. Formally, an OPF is defined through a prime of the form $p = u \cdot 2^k + 1$ where u is "small" in relation to 2^k (e.g. u is a 16-bit integer). All arithmetic functions of our OPF library get the factor u as well as the length of p as parameter and can process operands of arbitrary size (e.g. from 160 to 512 bits) without the need to re-compile the library. Thus, we can easily trade security versus performance and energy efficiency, which means MoTE-ECC is an energy-scalable ECC implementation.

The version of MoTE-ECC we describe in this paper supports two families of elliptic curves, namely Montgomery [27] and twisted Edwards [5] curves. An outstanding feature of elliptic curves in Montgomery form is the existence of a differential addition law that involves only the x-coordinate of the points. The so-called Montgomery ladder for scalar multiplication can use such differential additions in an efficient way and has the further benefit of a regular execution profile, which naturally protects against Simple Power Analysis (SPA) attacks [20]. Montgomery curves excel in settings where run-time memory (i.e. RAM) is scarce and scalar multiplication has to be performed with an arbitrary base point that is not known a priori. On the other hand, twisted Edwards curves provide the currently fastest formulas for general (i.e. non-differential) addition of points. Furthermore, the addition law presented in [5, Section 6] is complete if the curve parameters fulfill certain conditions. These properties make twisted Edwards curves attractive for applications that perform scalar multiplications by a fixed base point using e.g. the comb method [15]. In this paper, we take ephemeral ECDH key exchange as example[3] to evaluate the performance and memory consumption of MoTE-ECC. Ephemeral ECDH has one big advantage over static ECDH, namely *forward secrecy*, which is a highly desirable feature

[3] In accordance with previous work on ECC for WSNs (e.g. [23]), we use a straightforward (i.e. unauthenticated) variant of the ECDH protocol for our performance evaluation. However, a real-world application of (ephemeral) ECDH key exchange would require protection against Man-in-the-Middle (MitM) attacks, which can be achieved by signing the messages sent in each run of the protocol, or by using an advanced version of ECDH with "implicit" authentication, e.g. ECMQV.

for any kind of network, including WSNs. Our ECDH implementation combines the individual advantages of the Montgomery form and twisted Edwards form by exploiting the fact that every twisted Edwards curve is birationally equivalent to a Montgomery curve [5]. Ephemeral ECDH key exchange between two sensor nodes requires each node to execute two scalar multiplications; the first is performed with a fixed base point (namely the generator of an elliptic curve group) to generate a key pair, whereas the second scalar multiplication gets an arbitrary point as input and yields the shared secret key. Our implementation uses a fixed-base comb method on a twisted Edwards curve for the former and a Montgomery ladder on a Montgomery curve that is birationally equivalent to the Edwards curve for the latter, thereby combining the specific computational advantages of the two curve shapes in an optimal way[4].

We also made an effort to protect MoTE-ECC against passive, non-invasive implementation attacks. In the case of our ephemeral ECDH, this boils down to protecting the two scalar multiplications against Simple Power Analysis (SPA) attacks since, in each run of the protocol, a freshly generated random scalar is used, i.e. Differential Power Analysis (DPA) is not possible. All field-arithmetic operations are implemented in a highly regular fashion (i.e. without conditional statements such as `if-then-else` constructs) so that always exactly the same sequence of instructions is executed, independent of the value of the operands [24]. Furthermore, we developed a new approach for performing the fixed-base comb method with the goal of reducing SPA-leakage in relation to a standard implementation. It should be noted, however, that WSN applications have less stringent demands regarding side-channel resistance than e.g. smart cards. The integration of countermeasures against all known forms of side-channel attacks would introduce unfeasible overheads for most WSN applications. Instead, we aimed to protect our ECDH implementation against a so-called stealthy node compromise (i.e. a side-channel attack mounted "in the field" without physically capturing a node) as described in [9]. Since such in-field attacks are carried out under sub-optimal conditions (e.g. large noise levels), it normally suffices to have some basic countermeasures in place.

2 Arithmetic in Optimal Prime Fields

The prime fields we use in MoTE-ECC belong to a special class of finite fields known as Optimal Prime Fields (OPFs) [11]. These fields are defined by primes that can be written as $p = u \cdot 2^k + v$ whereby u and v are "small" compared to 2^k so that they fit into one or two registers of the target platform. MoTE-ECC supports OPFs with $2^{15} \leq u < 2^{16}$ (i.e. u is 16 bits long) and $v = 1$. A concrete example is $p = 65356 \cdot 2^{144} + 1$ (i.e. $u = 65356$ and $k = 144$), which happens to be a 160-bit prime that looks as follows in hex notation.

$$p = \text{0xFF4C0000000000000000000000000000000000001}$$

[4] MoTE-ECC is an abbreviation for Montgomery and Twisted Edwards based ECC.

Primes of such form are characterized by a low Hamming weight since only the two most significant bytes and the least significant byte are non-zero; all bytes in between are zero. The low weight of p allows for specific optimization of the modular arithmetic because only the non-zero bytes of p need to be processed in the reduction operation. For example, Montgomery's algorithm [26] can be simplified for these primes so that the modular reduction has only linear complexity, similar to generalized-Mersenne or pseudo-Mersenne primes [15].

2.1 Parameterized OPF Library

Our implementation of arithmetic operations in OPFs is largely based upon the OPF library for 8-bit AVR processors described in [24]. This library provides the full spectrum of arithmetic functionality required for scalar multiplication on Montgomery and twisted Edwards curves (i.e. addition, subtraction, multiplication, squaring and inversion), whereby each operation includes a reduction modulo a low-weight prime of the form $p = u \cdot 2^k + 1$. Both multiplication and squaring employ a special variant of the Montgomery reduction method [26] so that only the non-zero bytes of p are processed. All functions of the library are written in assembly language and optimized to yield a good trade-off between performance and (binary) code size. Furthermore, the arithmetic functions are parameterized, which means the operand length is not fixed (i.e. hard-coded) but passed as parameter to the function. In this way, the OPF library provides a high degree of flexibility as one and the same function can process operands of any length. Another important feature of the library is its resilience against SPA attacks as all arithmetic operations are implemented in a regular fashion and execute always the same sequence of instructions, regardless of the actual value of the operands. Only the inversion from [24] has a non-regular execution profile; therefore, we implemented a Fermat-based inversion from scratch (see below). A detailed description of the OPF library can be found in [24].

2.2 Fermat-Based Inversion in OPFs

Unlike to field addition and multiplication, inversion in $\mathbb{F}_p$ is not executed during the scalar multiplication when using projective coordinates [15]. In fact, the inversion operation is only needed to convert the result from projective back to affine coordinates. Two well-known techniques for computing the inverse of an element of $\mathbb{F}_p$ are the Extended Euclidean Algorithm (EEA) [15] and Fermat's little theorem. However, the EEA is highly irregular and may leak information about the input value that could be exploited to mount an attack as described in [28] to recover a few bits of the secret scalar. Therefore, it is mandatory to use an inversion algorithm that executes in a regular way and is not vulnerable to SPA and SPA-like attacks. Our implementation of the inversion in OPFs is based on Fermat's little theorem $a^{p-2} \equiv a^{-1} \bmod p$, i.e. we perform inversion via exponentiation. Unfortunately, the conventional square-and-multiply exponentiation method with an exponent of the form $u \cdot 2^k - 1$ requires n squarings and almost n multiplications, whereby n denotes the bit-length of p. To reduce

Algorithm 1. Optimized exponentiation-based inversion for OPFs

Input: Element a of $\mathbb{F}_p$ with $p = u \cdot 2^k + 1$.
Output: $r \equiv a^{u \cdot 2^k - 1} \equiv a^{-1} \bmod p$.

1: $u' \leftarrow u - 1$
2: $r \leftarrow a$, $b \leftarrow \lceil ld(k) - 2 \rceil$, $i \leftarrow 1$
3: **while** $b > 0$ **do**
4: $\quad t \leftarrow r^2$
5: $\quad$ **for** $j = 1$ **to** $i - 1$ **do**
6: $\quad\quad t \leftarrow t^2$
7: $\quad$ **end for**
8: $\quad r \leftarrow r \cdot t$
9: $\quad i \leftarrow i \ll 1$
10: $\quad$ **if** k & $b > 0$ **then**
11: $\quad\quad r \leftarrow r^2 \cdot a$
12: $\quad\quad i \leftarrow i + 1$
13: $\quad$ **end if**
14: $\quad b \leftarrow b \gg 1$
15: **end while**
16: $t \leftarrow r \cdot a$
17: $b \leftarrow 1$
18: **while** $b < 0x8000$ **do**
19: $\quad$ **if** u' & $b > 0$ **then**
20: $\quad\quad r \leftarrow r \cdot t$
21: $\quad$ **end if**
22: $\quad t \leftarrow t^2$
23: $\quad b \leftarrow b \ll 1$
24: **end while**
25: **if** u' & $b > 0$ **then**
26: $\quad r \leftarrow r \cdot t$
27: **end if**

execution time, we developed an inversion technique that is specifically crafted for OPFs (specified in Algorithm 1). Thanks to this algorithm, it is possible to nearly halve the overall number of operations to n squarings plus only $HW(k) + HW(u-1) + 1$ multiplications, where $HW(x)$ denotes the Hamming weight of x. In practice, this optimization almost doubles the performance compared to a straightforward square-and-multiply exponentiation. Algorithm 1 operates in two phases; in the first phase, a^{2^k-1} is calculated using the exponentiation method of Itoh and Tsujii [18], which was originally proposed for binary extension fields. In the second phase, a right-to-left square-and-multiply algorithm is carried out. Note that, for our primes, this second phase is much shorter since u is (at most) 16 bits long. In step 16, the multiplication $r \cdot a = a^{2^k-1} \cdot a$ yields $t = a^{2^k}$, which is needed for the right-to-left square-and-multiply algorithm.

3 Scalar Multiplication for Ephemeral ECDH

Performing ephemeral ECDH key exchange between two sensor nodes requires each node to execute two scalar multiplications; one to generate an ephemeral key pair and the second to obtain the shared secret. MoTE-ECC uses a twisted Edwards curve [5] for the former and a Montgomery curve [27] for the latter. In fact, it is more correct to say that both scalar multiplications are computed on the same elliptic curve; in one case we adopt the twisted Edwards form of this curve and in the second case its Montgomery form. We describe both forms in the next subsection and also discuss the execution time of scalar multiplication algorithms by counting the number of underlying field operations, whereby we adhere to the following notation: M (multiplication), S (squaring), A (addition or subtraction), I (inversion), and D (multiplication by a curve constant).

3.1 Montgomery and Twisted Edwards Curves

Montgomery Curve. In 1987, Peter Montgomery introduced a special model of elliptic curves, today known as Montgomery model [27]. An elliptic curve in Montgomery form over a prime field $\mathbb{F}_p$ is defined through the equation

$$E_M : \; Bv^2 = u^3 + Au^2 + u \tag{1}$$

where $A \in \mathbb{F}_p \setminus \{-2, 2\}$ and $B \in \mathbb{F}_p \setminus \{0\}$. The major attraction of these curves is the possibility to perform point arithmetic with the x-coordinate only. More precisely, when using projective coordinates to represent curve points, just the X and Z coordinate are needed to perform point addition and doubling. A so-called "differential" point addition requires exactly $3M + 2S + 6A$, whereas the doubling of a point costs $2M + 2S + 1D + 4A$. The Montgomery ladder is well known for being a *per se* highly regular algorithm for scalar multiplication on Montgomery curves [29]. Its regularity is simply due to the fact that it always executes both a point addition and a point doubling per scalar bit, irrespective of whether said bit is 0 or 1. Therefore, given a scalar k and base point P, the cost of computing $k \cdot P$ amounts to $5M + 4S + 1D + 10A$ per bit of k.

Normally, the parameter A is chosen so that multiplication by $(A + 2)/4$ is fast. However, in our case this means the Montgomery image of $(A + 2)/4$ has to be small since, as stated in Section 2.1, the OPF library uses Montgomery's reduction technique [26] for the modular multiplication.

Twisted Edwards Curve. Currently, elliptic curves in twisted Edwards form offer the most efficient formulae for general (i.e. non-differential) point addition [16], which makes them attractive for practical implementations. According to Bernstein et al [5], a twisted Edwards curve over a non-binary field $\mathbb{F}_q$ is given by an equation of the form

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

whereby a and d are distinct, non-zero elements of $\mathbb{F}_q$. The authors of [5] also introduced formulae for addition and doubling on such a curve using standard projective coordinates. Thereafter, Hişil et al proposed an extended coordinate system that includes an auxiliary coordinate $t = xy$ [16]. Instead of representing a point on a twisted Edwards curve E_E by its x and y coordinate only, one can use the extended affine coordinates (x, y, t). The corresponding projective coordinates of that point are $(X : Y : T : Z)$, whereby the auxiliary coordinate T has the property $T = XY/Z$ with $Z \neq 0$. Thanks to these coordinates, Hişil et al were able to devise very efficient point addition formulae, especially if the parameter $a = -1$. After applying some straightforward optimizations [14], the computational cost of a mixed point addition on a curve with $a = -1$ amounts to $7M + 6A$, while a doubling requires $3M + 4S + 6A$.

Birational Equivalence. Bernstein et al proved in [5] that the set of twisted Edwards curves over a non-binary field $\mathbb{F}_q$ is equivalent to the set of Montgom-

ery curves over $\mathbb{F}_q$. In particular, they showed that the twisted Edwards curve E_E over $\mathbb{F}_q$ with non-zero parameters a, d is birationally equivalent over $\mathbb{F}_q$ to the Montgomery curve E_M given by

$$A = 2(a+d)/(a-d), \quad B = 4/(a-d) \tag{3}$$

For an arbitrary point (x, y) on the twisted Edwards curve E_E, we can get the related point (u, v) on the equivalent Montgomery curve E_M as follows

$$u = (1+y)/(1-y), \quad v = (1+y)/((1-y)x) \tag{4}$$

Conversely, given the curve parameters $A \in \mathbb{F}_q \setminus \{-2, 2\}$ and $B \in \mathbb{F}_q \setminus \{0\}$, the corresponding Montgomery curve E_M is birationally equivalent over $\mathbb{F}_q$ to the twisted Edwards curve E_E with the parameters

$$a = (A+2)/B, \quad d = (A-2)/B \tag{5}$$

and the point (x, y) corresponding to (u, v) can be obtained as follows

$$x = u/v, \quad y = (u-1)/(u+1) \tag{6}$$

3.2 Generation of Curves

The security of elliptic curve cryptosystems is based on the intractability of the underlying Elliptic Curve Discrete Logarithm Problem (ECDLP). To date, the most efficient algorithm for solving a generic instance of the ECDLP in a given elliptic curve group $E(\mathbb{F}_p)$ has complexity $\mathcal{O}(\sqrt{n})$ where n is the largest prime divisor of $\#E(\mathbb{F}_p)$ [15]. Therefore, one must be careful to choose a field $\mathbb{F}_p$ and a curve E over $\mathbb{F}_p$ so that $E(\mathbb{F}_p)$ has prime order or contains a large subgroup of prime order. More concretely, when writing $\#E(\mathbb{F}_p)$ as a product of a prime n and a co-factor h, then n should have a length of approximately 160 bits and h should be small, e.g. $h \leq 4$ [15]. Furthermore, one has to ensure that E does not belong to a class of curves for which the ECDLP can be solved in less than $\sqrt{n}$ steps. Examples for such "weak" curves over $\mathbb{F}_p$ are anomalous curves and curves with small embedding degree (e.g. supersingular curves). Depending on the application, further security criteria not directly related to the ECDLP in $E(\mathbb{F}_p)$ may need to be considered. One example is *twist security*, which means that not only the curve E itself, but also the quadratic twist E' of E meets all criteria for hardness of the ECDLP. Using a curve with a secure twist thwarts certain implementation attacks (e.g. [10]) and allows for a simplification of the ECDH protocol when only the x-coordinates of points are exchanged [4].

Besides the security requirements from above, our curve generation process also takes certain efficiency criteria into account to ensure the point arithmetic on both the twisted Edwards curve and its Montgomery equivalent can achieve the best possible execution times. When generating a Montgomery curve, it is common practice to choose the curve parameter A such that $(A+2)/4$ is small (as suggested in [27]). In our case, this actually means the Montgomery image

of $(A+2)/4$ has to be small since our OPF library uses Montgomery reduction for the multiplication in $\mathbb{F}_p$. The second curve parameter B does not appear in the addition/doubling formulae and, therefore, has no impact on the execution time. On the other hand, the point arithmetic on a twisted Edwards curve is most efficient when the parameter a is fixed to -1 as in this case Hisil et al's fast and complete $7M$ formula for mixed addition can be used [16]. The second parameter d appears as operand in the complete addition formula described in Section 3.1 of [16], but not in the dedicated addition from [16, Section 3.2]. In our case, we can still use the complete addition formula without loss of performance since the comb method always adds a fixed base point P (or a multiple of P), which allows us to pre-compute $2dT_2$ as suggested in [16]. Another issue to consider is that the addition formula from [16, Section 3.1] is only complete when a is a square and d a non-square in $\mathbb{F}_p$. Fortunately, $a = -1$ is always a square in an OPF defined by a prime of the form $p = u \cdot 2^k + 1$; this becomes immediately evident by an evaluation of the Legendre symbol $\left(\frac{-1}{p}\right)$ taking into account that $(p-1)/2$ is highly even for all our primes.

As pointed out in Subsection 3.1, every twisted Edwards curve over a non-binary finite field $\mathbb{F}_q$ is birationally equivalent over $\mathbb{F}_q$ to a Montgomery curve and, conversely, every Montgomery curve is birationally equivalent to a twisted Edwards curve [5]. However, this does not imply that every Montgomery curve is birationally equivalent to a twisted Edwards curve with a fast and complete addition law. The goal of our curve generation procedure is to find a Montgomery curve along with its twisted Edwards counterpart so that both satisfy the security and efficiency criteria outlined above. To achieve this, we used the computer algebra system Magma. Magma provides an extensive pool of functions for computations on elliptic curves given in both short and long (non-simplified) Weierstraß form, but does not directly support the twisted Edwards from. However, a twisted Edwards curve with the parameters $a, d \in \mathbb{F}_q$ can be expressed via a non-simplified Weierstraß equation as follows.

$$a_2 = \frac{a+d}{2}\,,\; a_4 = \left(\frac{a-d}{4}\right)^2\,,\text{ and }\; a_1 = a_3 = a_6 = 0 \tag{7}$$

The above formulas were derived by simply exploiting the fact that any twisted Edwards curve over a non-binary field $\mathbb{F}_q$ is birationally equivalent to a Montgomery curve, which was formally proven in [5]. We fixed the parameter a to -1 to take advantage of the fast formulas for point addition and doubling presented in [16]. Furthermore, we only consider values of d that are non-square so as to ensure completeness of the addition formula.

3.3 Regular Digit-Set Conversion for Comb Method

When performing a scalar multiplication $k \cdot P$ on a fixed base point P, one can take advantage of the so-called comb method to reduce execution time [15]. In general, the comb method processes $w \geq 2$ bits of the scalar k at once and requires pre-computation and storage of (up to) 2^w curve points, all of which are

linear combinations of w multiples of P. An n-bit scalar multiplication consists of exactly $d = \lceil n/w \rceil$ point doublings and at most d point additions. Thus, the w-bit comb method cuts the number of point doublings by a factor of w versus the binary (i.e. "double-and-add") technique. The number of point additions is not constant but depends on the scalar since, similar to the binary method, the addition step is simply omitted if the corresponding w bits of k are all 0. It is possible to reduce the number of pre-computed points to 2^{w-1} at the expense of point negation operations to be carried out "on-the-fly," resulting in a slight performance degradation. Our implementation of the comb method follows this avenue; in each step we process $w = 4$ bits of the scalar at once using $2^{w-1} = 8$ pre-computed points, which are negated on-the-fly if necessary.

A conventional implementation of the comb technique can leak information related to the scalar k since, as explained above, the number of point additions is not constant but depends on k. MoTE-ECC uses the comb method for fixed-point scalar multiplication on a twisted Edwards curve, which means we could exploit the completeness of the Edwards addition law and just add the neutral element $\mathcal{O}$ to achieve a (more) regular execution profile. Even though such an approach would foil timing attacks, it may still leak SPA-relevant information since the coordinates of $\mathcal{O}$ consist of the field elements 0 and 1. Multiplying an arbitrary field element by 0 or 1 causes less bit flips in the multiplier hardware and register file than a multiplication of two random elements of $\mathbb{F}_p$. Thus, we opted to not add $\mathcal{O}$ but represent the 4-bit digits processed in each step of the comb method using a signed digit-set that does not contain 0.

To foil SPA attacks, all operations involving bits of the secret scalar k need to be implemented in a highly regular way without conditional statements. In our case, this requirement boils down to the demand for a regular algorithm to convert a radix-2^4 integer with digits from the set $D = \{0, 1, 2, \ldots, 14, 15\}$ into an equivalent radix-2^4 representation using a "zero-free" digit set of the form $D' = \{\pm 1, \pm 3, \ldots, \pm 13, \pm 15\}$. Algorithms for this kind of digit-set conversion were proposed in e.g. [19, 14]. However, we use a different conversion technique that is more regular and easier to implement than the state-of-the-art. Our algorithm for digit-set conversion is based on the following observation: Any odd n-bit integer k given by $k = \sum_{i=0}^{n-1} k_i \cdot 2^i$ with $k_i \in \{0, 1\}$ for $0 < i < n-1$ and $k_{n-1} = k_0 = 1$ can be written in standard Binary Signed-Digit (BSD) form as $k = 2^{n-1} + \sum_{i=0}^{n-2} (2k_{i+1} - 1) \cdot 2^i$. The expression $2k_{i+1} - 1$ yields either 1 (when $k_{i+1} = 1$) or -1 (if $k_{i+1} = 0$), i.e. all digits of our BSD representation of k are non-zero. One can verify the correctness of this conversion as follows.

$$\begin{aligned} k \;&=\; 2^{n-1} + \sum_{i=0}^{n-2} (2k_{i+1} - 1) \cdot 2^i \;=\; 2^{n-1} - \sum_{i=0}^{n-2} 2^i + \sum_{i=0}^{n-2} 2k_{i+1} \cdot 2^i \;= \\ &=\; 1 + \sum_{i=0}^{n-2} k_{i+1} \cdot 2^{i+1} \;=\; 1 + \sum_{i=1}^{n-1} k_i \cdot 2^i \;=\; \sum_{i=0}^{n-1} k_i \cdot 2^i \;\text{ with }\; k_0 = 1 \end{aligned} \tag{8}$$

Equation (8) leads to a simple technique to convert an odd integer given in conventional binary form into a BSD representation consisting of only non-zero

Algorithm 2. Regular w-bit comb method for fixed-base scalar multiplication

Input: n-bit scalar $k = (k_{n-1}, \ldots, k_1, k_0)_2$ with $k_0 = 1$, point $P \in E(\mathbb{F}_p)$.
Output: $Q = k \cdot P$.

1: Pre-compute $R[j] = R[a_{w-2}, \ldots, a_1, a_0] = 2^{dw}P + (2a_{w-2} - 1)2^{(d-1)w}P + \ldots + (2a_1 - 1)2^wP + (2a_0 - 1)P$ for all bit-strings $j = (a_{w-2}, \ldots, a_1, a_0)$ of length $w - 1$
2: $Q \leftarrow R[k_{dw}, \ldots, k_{2d}, k_d]$
3: **for** $i = d - 1$ **downto** 1 **do**
4: $\quad Q \leftarrow 2Q$
5: $\quad Q \leftarrow Q + (2k_{(w-1)d+i} - 1) \cdot R[k_{(w-2)d+i}, \ldots, k_{d+i}, k_i]$
6: **end for**

digits, namely -1 and 1. We just have to shift the whole binary representation of k one bit to the right and insert a "1" at the vacant MSB position. Now this shifted bit-string is already exactly the BSD-form of k when we interpret all 0 bits as -1. A radix-2^4 representation can be obtained by dividing the bit-string into groups of 4-bit digits, each of which corresponds to an odd number in the range $[-15, 15]$. In this way, we get a signed radix-2^4 representation that does not contain zero digits. Similar to Joye et al's signed-digit recoding algorithm from [19, Sect. 3.2], our conversion technique requires k to be odd as otherwise the result will be off by 1. More precisely, when performing a scalar multiplication using the proposed digit-set conversion with an even k, the actual result is $(k - 1) \cdot P$ instead of $k \cdot P$, which means a final addition of P is required. However, such a final addition does not necessarily introduce an irregularity in the comb method since we can define private keys to be odd (or even) so that the final addition is either never or always executed. Unlike the recoding technique from [19], the execution time and power consumption profile of our conversion is independent of the position of the MSB of k since a leading bit-string of the form $000 \cdots 001$ becomes $1\bar{1}\bar{1} \cdots \bar{1}\bar{1}\bar{1}$ where $\bar{1}$ denotes -1. Hence, short scalars (i.e. scalars having less than n bits) are processed in precisely the same way as an n-bit scalar, which is not the case for the conversion proposed in [19].

Algorithm 2 shows a highly regular variant of the fixed-base comb method for point multiplication. We use the same notation as Sect. 3.3.2 in [15], which means w denotes the number of bits (i.e. length of the bit-string) processed in each iteration of the loop and $d = \lceil n/w \rceil$. Similar to the straightforward comb method specified in [15, Algorithm 3.44], our variant comprises an offline phase (Step 1) and an online phase. In the first phase, 2^{w-1} points are pre-computed and stored, all of which are linear combinations of P. Our implementation pre-computes eight points as we use $w = 4$ to achieve a balance between execution time and storage requirements. Note that an expression of the form $(2a_i - 1)$ in Step 1 yields either 1 (when $a_i = 1$) or -1 (if $a_i = 0$), thereby performing the digit-set conversion described above. The online phase consists of a simple loop that executes a doubling followed by an addition in each iteration. However, in contrast to the standard comb method, $w - 1$ bits (instead of w bits) of k are used to determine which of the 2^{w-1} pre-computed points is to be added, while a further bit (namely $k_{(w-1)d+i}$ in Step 5 of Algorithm 2) defines whether this

point is actually added or subtracted. To achieve a regular execution, we need a function that, depending on the value of a bit, assigns either a point R or the negative of that point (i.e. $-R$) to a destination. The negative of a point R in extended affine coordinates is $-R = (-x, y, -t)$ [16]; consequently, the problem of negating a point boils down to the negation of elements of $\mathbb{F}_p$, which can be realized through subtractions from p. MoTE-ECC performs the negation of an element $x \in \mathbb{F}_p$ depending on the value of a bit b as follows. First, we compute $x' = p - x$ via subtraction. Then, we use the bit b to derive a mask m, which is either an "all-1" byte (if $b = 1$) or an "all-0" byte (if $b = 0$), in the same way as described in [24]. Furthermore, we need a second mask m' that is the bit-wise complement of m, i.e. m' is 0 if m is an "all-1" byte and vice versa. After these preparations, we compute $(x_i' \,\&\, m) \mid (x_i \,\&\, m')$ for all bytes of x' and x (where & and | denote the bit-wise *and* and *or* operation, respectively) and assign the result to the corresponding byte of the destination. The field element we get in this way is either $-x = p - x$ (if $b = 1$, i.e. the negation is actually carried out) or simply x (if $b = 0$, i.e. no negation). In summary, our regular comb method executes always exactly $d - 1$ point additions and $d - 1$ doublings, irrespective of the actual value of the scalar bits and the index of the MSB.

4 Implementation and Evaluation

We implemented MoTE-ECC for the 8-bit AVR platform (e.g. ATmega128 [3]) and assessed its execution time and memory footprint using ephemeral ECDH key exchange as example. The main idea of our ECDH protocol is to exploit the birational equivalence between Montgomery and twisted Edwards curves [5] to improve the overall performance. Assume two sensor nodes, named $\mathcal{A}$ and $\mathcal{B}$ in the following, want to establish a shared secret key, whereby the set of domain parameters (a, d, A, B, G, p) has already been agreed upon. Here, a and d are the parameters of a twisted Edwards curve E_E, while A and B characterize the birationally-equivalent Montgomery curve E_M. G is a point of prime order on E_E, and p defines the underlying OPF. One round of our ECDH key exchange protocol can be divided into two stages as follows:

1. Node $\mathcal{A}$ generates a private key d_A and computes the corresponding public key $Q = d_A \cdot G$. This scalar multiplication is done on the twisted Edwards curve E_E using generator G. Then, node $\mathcal{A}$ converts the point $Q = (x_q, y_q)$ to a point $M = (x_m, y_m)$ on the birationally equivalent Montgomery curve E_M and sends the x-coordinate x_m of M to node $\mathcal{B}$. Node $\mathcal{B}$ performs the same steps with private key d_B and sends its x-coordinate to $\mathcal{A}$.
2. After node $\mathcal{A}$ has received the x-coordinate from $\mathcal{B}$, it computes the scalar multiplication $S = d_A \cdot M$ (whereby M consists of only an x coordinate) on the Montgomery curve E_M. Node $\mathcal{B}$ does the same with the x coordinate it received from node $\mathcal{A}$.

Both node $\mathcal{A}$ and node $\mathcal{B}$ have to carry out two scalar multiplications to obtain the shared secret key $S = d_A \cdot d_B \cdot G$. Since the base point G is fixed and known

Table 1. Execution time (in clock cycles) of field arithmetic operations for operands of a length of 160, 192, 224, and 256 bits

Operation	160 bits	192 bits	224 bits	256 bits
mod_add	530	631	732	833
mod_sub	530	631	732	833
mod_mul	3237	4500	5971	7650
mod_sqr	2901	3909	5058	6347
mod_inv	571916	830823	1163655	1491839

in advance, we can speed up the execution of the first scalar multiplication with help of the fixed-base comb method using a window width of $w = 4$ and eight pre-computed points as described in Section 3.3.

MoTE-ECC adopts the "extended" coordinate system for twisted Edwards curves introduced in [16], which means we obtain the point Q resulting from the first scalar multiplication in extended projective coordinates. A straightforward conversion of a point Q on a twisted Edwards curve E_E into a point M on the birationally-equivalent Montgomery curve E_M can be executed in the following way. We firstly convert the projective point $Q = (X_q, Y_q, T_q, Z_q)$ on E_E to its affine representation $Q = (x_q, y_q)$ and then calculate the equivalent point $M = (x_m, y_m)$ on E_M via $x_m = (1 + y_q)/(1 - y_q)$ and $y_m = (1 + y_q)/((1 - y_q) \cdot x_q)$ as specified in [5]. However, when doing so, we have to execute an inversion in the affine-to-projective conversion to get $1/Z_q$ and another inversion as part of the Edwards-to-Montgomery transformation (to obtain $1/[(1 - y_t) \cdot x_t]$). To reduce the computational overhead caused by two inversions, we directly transform the point $Q = (X_q, Y_q, T_q, Z_q)$ to the point $M = (x_m, y_m)$ as follows.

$$x_m = (1 + y_q)/(1 - y_q) = (1 + Y_q/Z_q)/(1 - Y_q/Z_q) = (Z_q + Y_q)/(Z_q - Y_q) \quad (9)$$

$$y_m = (1 + y_q)/(x_q \cdot (1 - y_q)) = (Z_q^2 + Y_q Z_q)/(X_q Z_q - X_q Y_q) \quad (10)$$

In this way, we only need one inversion to compute $1/(X_q Z_q - X_q Y_q)$, which we just have to multiply by X_q to get $1/(Z_q - Y_q)$.

4.1 Execution Time

As explained in Section 2, we implemented the OPF inversion from scratch and used the OPF library from [24] for all other arithmetic operations. Table 1 lists the simulated execution times for the ATmega128. The modular multiplication only takes 3237, 4500, 5971 and 7650 clock cycles for 160, 192, 224 and 256-bit operands, respectively. As stated in [24], these timings represent speed records for modular multiplication on an 8-bit AVR processor. For 256-bit operands, the OPF multiplication is even faster than the multiplication of the NaCl software for AVR [17]. As also shown in Table 1, our regular Itoh-Tsujii inversion needs 571916 clock cycles (in a 160-bit OPF), which is about 1.36 times faster than the unprotected inversion of the well-known TinyECC library [23].

Table 2. Execution time (in clock cycles) of point arithmetic operations over 160, 192, 224, and 256-bit OPFs

Operation	160 bits	192 bits	224 bits	256 bits
Mo point add	19479	25890	33207	41428
Mo point dbl	15950	21072	26884	33390
TE point add	27355	36903	47907	60367
TE point dbl	25421	33848	43463	54262

We wrote the functions for point arithmetic in ANSI C and determined the execution time of point addition and point doubling on both twisted Edwards and Montgomery curves. The timings are reported in Table 2 for OPFs of sizes ranging between 160 and 256 bits. Taking the 160-bit OPF as example, it turns out that addition and doubling on a Montgomery curve require exactly $19,479$ and $15,950$ clock cycles, respectively. On the other hand, adding two points on a twisted Edwards curve needs $27,355$ clock cycles, while a doubling operation costs $25,421$ cycles. Our simulation results are exactly in line with the analysis in Section 3. For example, the point addition on a Montgomery curve requires only $3M + 2S$, which is clearly more efficient than the $7M$ for adding points on a twisted Edwards curve. Thus, it is not surprising that the point arithmetic on the Montgomery curve is much faster than on the twisted Edwards curve. The addition and doubling operation of our implementation for Montgomery curves outperform that of the TinyECC library by a factor of more than three. On the other hand, the point addition and doubling on the twisted Edwards curve are roughly 2.1 and 1.9 times faster than TinyECC, respectively.

Table 3. Execution time (in clock cycles) of scalar multiplication over 160, 192, 224 and 256-bit OPFs

Operation	160 bits	192 bits	224 bits	256 bits
Scalar mul. Mo curve	6276630	9964549	14856446	21118778
Scalar mul. TE curve	2767454	4412519	6603888	9420788
Full MoTE-ECDH	9044084	14377068	21460334	30539566

The execution times of a full scalar multiplication using the two curve shapes over 160, 192, 224 and 256-bit OPFs are summarized in Table 3. Each run of our ECDH key exchange protocol consists of two scalar multiplications; the first one is performed on a twisted Edwards curve, while the second is carried out on the birationally-equivalent Montgomery curve. When using a fixed-base comb method as described in Section 3.3, the first stage of the ECDH protocol can be executed in about $2.76 \cdot 10^6$ clock cycles over a 160-bit OPF (i.e. 0.37 s at the typical sensor-node frequency of 7.37 MHz), which already includes the Edwards-to-Montgomery conversion. The second stage of the ECDH protocol is more expensive than the first one since it involves a scalar multiplication by an

elliptic-curve point that is neither fixed nor known a priori. MoTE-ECC uses a simple ladder on a Montgomery curve for this second scalar multiplication and achieves an execution time of roughly $6.27 \cdot 10^6$ cycles in the 160-bit case. The complete computational cost of an ephemeral ECDH key exchange amounts to about $9.04 \cdot 10^6$ clock cycles when using a 160-bit OPF as underlying algebraic structure, which corresponds to an execution time of 1.22 s at 7.37 MHz.

4.2 Memory Footprint and Code Size

Besides performance, run-time memory consumption is an important criterium for WSN applications since a typical AVR-based sensor node features only 4 kB RAM. Our comb method for scalar multiplication on a twisted Edwards curve requires to store eight points given in extended affine coordinates. However, as these points are pre-computed "off-line," we can store them in ROM or in flash memory. In this way, we only need to transfer the point that is required for the current iteration of the comb method from ROM or flash memory to RAM. As a consequence, a full ephemeral ECDH key exchange supporting elliptic curves over 160, 192, 224, and 256-bit OPFs (without re-compilation) occupies a mere 556 bytes in RAM, which includes besides all global and local variables also the stack. A stripped-down variant of MoTE-ECC supporting only fields of a size of up to 160 bits has a RAM footprint of just 380 bytes.

Even though ROM (resp. flash) usage is, in general, less critical than RAM footprint, it is still important to analyze the (binary) code size. The arithmetic library for OPFs used by MoTE-ECC is implemented in a parameterized form with rolled loops (so as to support operands of varying length [24]), which has the side-benefit of compact code size. Besides the field arithmetic, also the concrete implementation of the fixed-base comb method has a large impact on the ROM (resp. flash) requirements of MoTE-ECC. Our choice of $w = 4$ with eight pre-computed points represents a fair trade-off between performance and code size. The total ROM/flash footprint of MoTE-ECC supporting Montgomery as well as twisted Edwards curves is 14.7 kB, which constitutes some 11.5% of the 128 kB flash memory that is available on a typical AVR-based sensor node.

4.3 Comparison with Related Work

Many ECC implementations for the 8-bit AVR platform have been reported in the literature. However, most of them were solely optimized for speed and did not properly consider the limited resources of 8-bit sensor nodes. We compare our MoTE-ECC library with previous work in three main aspects: performance (i.e. execution time of fixed-point and variable-point scalar multiplication, and execution time of both together), RAM footprint, and ROM requirements. The key figures of MoTE-ECC and previous ECC implementations can be found in Table 4, whereby all timings are specified for an ATmega128 processor clocked at a frequency of 7.37 MHz. We take the implementations using a 160-bit field as example to demonstrate the advantages of MoTE-ECC. Our implementation achieves the best execution time for ephemeral ECDH key exchange among all

Table 4. Comparison of ECC libraries for 160, 192, 224, and 256-bit prime fields

Reference	Field	Fixed P.	Rand. P.	RAM	ROM	ECDH
Liu [23]	160 bit	2.05 s	2.30 s	1174 B	19.0 kB	4.35 s
Wang [33]	160 bit	1.24 s	1.35 s	3200 B	15.8 kB	2.59 s
Szczech. [31]	160 bit	1.27 s	1.27 s	1800 B	46.1 kB	2.54 s
Gura [13]	160 bit	0.88 s	0.88 s	282 B	3.7 kB	1.76 s
Ugus [32]	160 bit	0.57 s	1.03 s	543 B	3.6 kB	1.60 s
Großschädl [12]	160 bit	0.74 s	0.74 s	n/a	n/a	1.48 s
MoTE-160	160 bit	0.37 s	0.85 s	556 B	14.7 kB	1.22 s
Liu [23]	192 bit	2.90 s	2.90 s	1510 B	19.0 kB	5.80 s
Gura [13]	192 bit	1.35 s	1.35 s	336 B	4.0 kB	2.70 s
Lederer [21]	192 bit	0.71 s	1.67 s	1398 B	23.0 kB	2.38 s
MoTE-192	192 bit	0.60 s	1.35 s	556 B	14.7 kB	1.95 s
Gura [13]	224 bit	2.38 s	2.38 s	422 B	4.8 kB	4.76 s
MoTE-224	224 bit	0.90 s	2.01 s	556 B	14.7 kB	2.91 s
Hutter [17]	255 bit	3.80 s	3.80 s	922 B	17.4 kB	7.60 s
Hutter [17]	255 bit	3.11 s	3.11 s	681 B	28.9 kB	6.22 s
MoTE-256	256 bit	1.28 s	2.86 s	556 B	14.7 kB	4.14 s

prime-field based ECC libraries documented in the literature. In particular, we require just about 28% of the execution time of TinyECC [23]. However, since MoTE-ECC supports two curve shapes, it occupies slightly more RAM and is larger in terms of code size than the implementations from [13] and [32]. When compared with all known implementations using 160-bit fields (including also binary-field libraries, which are not specified in Table 4), our implementation is only slower than the work of Aranha et al [2]. However, their software employs a carefully-optimized multiplication technique for binary fields, which achieves high performance at the expense of a RAM footprint of 2.8 kB and a code size of 32.0 kB. Furthermore, it should be noted that MoTE-ECC contains countermeasures against SPA attacks, which is not the case for all other ECC libraries listed in Table 4 except NaCl [17] and the work introduced in [21]. NaCl is the only implementation with a high level of SPA resistance similar to ours. While NaCl is fast and small in terms of code size[5], it supports only a single curve. In contrast, our implementation is highly scalable as it supports fields and curves of various size (e.g. from 160 to 256 bits) without re-compilation.

5 Conclusions

The main contributions of this paper can be recapitulated as follows: First, we extended Liu et al's [24] parameterized yet efficient arithmetic library for OPFs

[5] The ROM size of NaCl given in Table 4 is for the complete library, which includes besides the Curve25519-based ECC part also SHA-512, Salsa20, and Poly-1305.

with a Fermat-based inversion that is robust against SPA attacks. Second, we presented a highly regular implementation of the comb method so as to reduce the SPA-leakage of fixed-base scalar multiplication. Third, we described a new way of performing ephemeral ECDH key exchange by combining the individual computational benefits of Montgomery and twisted Edwards curves. Fourth, we discussed how these curves have to be generated to satisfy both efficiency and security criteria. The former three contributions have been implemented and evaluated in MoTE-ECC, an efficient, scalable, and SPA-resistant ECC library for AVR processors such as the ATmega128. MoTE-ECC is able to perform the two scalar multiplications of an ephemeral ECDH key exchange in a little more than $9 \cdot 10^6$ clock cycles altogether when the underlying OPF has a size of 160 bits, which significantly advances the state-of-the-art in prime-field based ECC on an 8-bit processor. Another advantage of MoTE-ECC is its scalability since it supports fields and curves of different size (e.g. 160, 192, 224, and 256 bits) without re-compilation. The RAM footprint of MoTE-ECC for OPFs of up to 256 bits is 556 bytes, which is less than 15% of the available RAM of a typical AVR-based sensor node. A stripped-down variant of MoTE-ECC that supports OPFs of a size of up to 160 bits occupies only 380 bytes in RAM.

Acknowledgements. We thank the anonymous reviewers of CHES 2013 and ACNS 2014 for their valuable comments and suggestions.

The research described in this paper was supported in part by the Austrian Research Promotion Agency (FFG) under grant 836628 (SeCoS) and the Fonds National de la Recherche (FNR) Luxembourg under AFR grant 1359142.

References

1. I. F. Akyildiz and M. C. Vuran. *Wireless Sensor Networks*. John Wiley and Sons, 2010.
2. D. F. Aranha, R. Dahab, J. C. López, and L. B. Oliveira. Efficient implementation of elliptic curve cryptography in wireless sensors. *Advances in Mathematics of Communications*, 4(2):169–187, May 2010.
3. Atmel Corporation. 8-bit ARV® Microcontroller with 128K Bytes In-System Programmable Flash: ATmega128, ATmega128L. Datasheet, available for download at `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`, June 2008.
4. D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *Public Key Cryptography — PKC 2006*, vol. 3958 of *Lecture Notes in Computer Science*, pp. 207–228. Springer Verlag, 2006.
5. D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. In *Progress in Cryptology — AFRICACRYPT 2008*, vol. 5023 of *Lecture Notes in Computer Science*, pp. 389–405. Springer Verlag, 2008.
6. H. Brody. 10 emerging technologies that will change the world. *Technology Review*, 106(1):33–49, Feb. 2003.
7. CertiVox Corporation. CertiVox MIRACL SDK. Source code, available for download at `http://www.certivox.com`, June 2012.

8. Crossbow Technology, Inc. MICAz Wireless Measurement System. Data sheet, available for download at `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`, Jan. 2006.
9. G. de Meulenaer and F.-X. Standaert. Stealthy compromise of wireless sensor nodes with power analysis attacks. In *Mobile Lightweight Wireless Systems — MOBILIGHT 2010*, vol. 45 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 229–242. Springer Verlag, 2010.
10. P.-A. Fouque, R. Lercier, D. Réal, and F. Valette. Fault attack on elliptic curve Montgomery ladder implementation. In *Proceedings of the 5th International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2008)*, pp. 92–98. IEEE Computer Society Press, 2008.
11. J. Großschädl. TinySA: A security architecture for wireless sensor networks. In *Proceedings of the 2nd International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2006)*, pp. 288–289. ACM Press, 2006.
12. J. Großschädl, M. Hudler, M. Koschuch, M. Krüger, and A. Szekely. Smart elliptic curve cryptography for smart dust. In *Quality of Service in Heterogeneous Networks — QSHINE 2010*, vol. 74 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 548–559. Springer Verlag, 2010.
13. N. Gura, A. Patel, A. S. Wander, H. Eberle, and S. Chang Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Cryptographic Hardware and Embedded Systems — CHES 2004*, vol. 3156 of *Lecture Notes in Computer Science*, pp. 119–132. Springer Verlag, 2004.
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 Verlag, 2004.
16. H. Hişil, K. K.-H. Wong, G. Carter, and E. Dawson. Twisted Edwards curves revisited. In *Advances in Cryptology — ASIACRYPT 2008*, vol. 5350 of *Lecture Notes in Computer Science*, pp. 326–343. Springer Verlag, 2008.
17. M. Hutter and P. Schwabe. NaCl on 8-bit AVR microcontrollers. In *Progress in Cryptology — AFRICACRYPT 2013*, vol. 7918 of *Lecture Notes in Computer Science*, pp. 156–172. Springer Verlag, 2013.
18. T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Information and Computation*, 78(3):171–177, Sept. 1988.
19. M. Joye and M. Tunstall. Exponent recoding and regular exponentiation algorithms. In *Progress in Cryptology — AFRICACRYPT 2009*, vol. 5580 of *Lecture Notes in Computer Science*, pp. 334–349. Springer Verlag, 2009.
20. M. Joye and S.-M. Yen. The Montgomery powering ladder. In *Cryptographic Hardware and Embedded Systems — CHES 2002*, vol. 2523 of *Lecture Notes in Computer Science*, pp. 291–302. Springer Verlag, 2003.
21. C. Lederer, R. Mader, M. Koschuch, J. Großschädl, A. Szekely, and S. Tillich. Energy-efficient implementation of ECDH key exchange for wireless sensor networks. In *Information Security Theory and Practice — WISTP 2009*, vol. 5746 of *Lecture Notes in Computer Science*, pp. 112–127. Springer Verlag, 2009.
22. J. Lee, S. H. Son, and M. Singhal. Design of an architecture for multiple security levels in wireless sensor networks. In *Proceedings of the 7th International Conference on Networked Sensing Systems (INSS 2010)*, pp. 107–114. IEEE, 2010.

23. 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)*, pp. 245–256. IEEE Computer Society Press, 2008.
24. Z. Liu, J. Großschädl, and D. S. Wong. Low-weight primes for lightweight elliptic curve cryptography on 8-bit AVR processors. In *Information Security and Cryptology — INSCRYPT 2013*, vol. ?? of *Lecture Notes in Computer Science*, pp. ??–??. Springer Verlag, 2014.
25. J. Lopez and J. Zhou. *Wireless Sensor Network Security*, vol. 1 of *Cryptology and Information Security Series*. IOS Press, 2008.
26. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, Apr. 1985.
27. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, Jan. 1987.
28. D. Naccache, N. P. Smart, and J. Stern. Projective coordinates leak. In *Advances in Cryptology — EUROCRYPT 2004*, vol. 3027 of *Lecture Notes in Computer Science*, pp. 256–267. Springer Verlag, 2004.
29. K. Okeya, H. Kurumatani, and K. Sakurai. Elliptic curves with the Montgomery-form and their cryptographic applications. In *Public Key Cryptography — PKC 2000*, vol. 1751 of *Lecture Notes in Computer Science*, pp. 238–257. Springer Verlag, 2000.
30. S. Slijepcevic, M. Potkonjak, V. Tsiatsis, S. Zimbeck, and M. B. Srivastava. On communication security in wireless ad-hoc sensor networks. In *Proceedings of the 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2002)*. IEEE Computer Society Press, 2002.
31. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In *Wireless Sensor Networks — EWSN 2008*, vol. 4913 of *Lecture Notes in Computer Science*, pp. 305–320. Springer Verlag, 2008.
32. O. Ugus, D. Westhoff, R. Laue, A. Shoufan, and S. A. Huss. Optimized implementation of elliptic curve based additive homomorphic encryption for wireless sensor networks. In *Proceedings of the 2nd Workshop on Embedded Systems Security (WESS 2007)*, pp. 11–16, 2007. Available for download at `http://arxiv.org/abs/0903.3900`.
33. H. Wang and Q. Li. Efficient implementation of public key cryptosystems on mote sensors. In *Information and Communications Security — ICICS 2006*, vol. 4307 of *Lecture Notes in Computer Science*, pp. 519–528. Springer Verlag, 2006.
34. Y. Zhang and J. Großschädl. Efficient prime-field arithmetic for elliptic curve cryptography on wireless sensor nodes. In *Proceedings of the 1st International Conference on Computer Science and Network Technology (ICCSNT 2011)*, vol. 1, pp. 459–466. IEEE, 2011.