Twisted Edwards-Form Elliptic Curve Cryptography for 8-bit AVR-based Sensor Nodes

Dalin Chu
Shandong University, China
chudalin@gmail.com

Johann Großschädl
University of Luxembourg
johann.groszschaedl@uni.lu

Zhe Liu
University of Luxembourg
zhe.liu@uni.lu

Volker Müller
University of Luxembourg
volker.muller@uni.lu

Yang Zhang
University of Luxembourg
yang.zhang@uni.lu

ABSTRACT

Wireless Sensor Networks (WSNs) pose a number of unique security challenges that demand innovation in several areas including the design of cryptographic primitives and protocols. Despite recent progress, the efficient implementation of Elliptic Curve Cryptography (ECC) for WSNs is still a very active research topic and techniques to further reduce the time and energy cost of ECC are eagerly sought. This paper presents an optimized ECC implementation that we developed from scratch to comply with the severe resource constraints of 8-bit sensor nodes such as the MICAz and IRIS motes. Our ECC software uses Optimal Prime Fields (OPFs) as underlying algebraic structure and supports two different families of elliptic curves, namely Weierstraß-form and twisted Edwards-form curves. Due to the combination of efficient field arithmetic and fast group operations, we achieve an execution time of $5.8 \cdot 10^6$ clock cycles for a full 158-bit scalar multiplication on an 8-bit ATmega128 microcontroller, which is 2.78 times faster than the widely-used TinyECC library. Our implementation also shows that the energy cost of scalar multiplication on a MICAz (or IRIS) mote amounts to just 19 mJ when using a twisted Edwards curve over a 160-bit OPF. This result compares fairly well with the energy figures of two recently-presented hardware designs of ECC based on twisted Edwards curves.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption—*Public Key Cryptosystems*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Algorithms, Experimentation, Performance, Security

Keywords

MICAz Mote, AVR Processor, Twisted Edwards Curve

Cryptology ePrint Archive, Report 2012/730, http://eprint.iacr.org, 2013.

1. INTRODUCTION

In recent years, Wireless Sensor Networks (WSNs) have found widespread adoption in such areas as environmental monitoring, military surveillance, industrial control, home automation, and health care [1]. Many of said applications collect or process sensitive information, which initiated an extensive body of research on security and privacy aspects of WSNs. The special adversary models and threat scenarios of WSNs pose a multitude of unique research problems (see e.g. [22] for an overview), including some that are still not properly solved and, hence, need further consideration [16]. Wang et al [23] identify the following building blocks as essential for the design and implementation of a secure WSN: cryptography, key management, secure routing, secure data aggregation, and intrusion detection. One of the open research issues mentioned in [23] is to further improve the efficiency of Public-Key Cryptography (PKC) on small sensor nodes with limited computational power. They state that "public key cryptography can greatly ease the design of security in WSNs" [23, p. 19], but perceive overheads in execution time and energy consumption as limiting factors for the widespread deployment of PKC.

The benefits and drawbacks of using PKC in WSNs have been widely researched in the past ten years. Early work on the feasibility of PKC in WSNs includes that of Carman et al [6], who analyzed and compared the computation time and energy requirements of RSA, DSA, Diffie-Hellman and a few other public-key algorithms. The first really practical RSA implementation for an 8-bit sensor node, namely the prevalent MICAz mote [9], was presented by Gura et al in 2004 [13]. They also introduced highly-optimized software for Elliptic Curve Cryptography (ECC) on 8-bit AVR micro-controllers and reported an execution time of less than $6.5 \cdot 10^6$ clock cycles for a 160-bit scalar multiplication. This result set a new speed record for ECC on an 8-bit platform and has since then been generally regarded as the ultimate proof that strong PKC is feasible on resource-constrained sensor nodes. One of the most widely used ECC implementations for WSNs is TinyECC [19], whose first version was released in the late 2007. TinyECC is a highly configurable ECC library for wireless sensor nodes running TinyOS and supports Weierstraß curves over arbitrary prime fields. To increase efficiency, TinyECC contains special optimizations for standardized 128, 160, and 192-bit fields.

In this paper, we describe a carefully-optimized software implementation of ECC for 8-bit AVR-based sensor nodes like the MICAz and IRIS motes. The aim of our work is to advance the state-of-the-art in lightweight ECC for WSNs by exploring the potential of new families of elliptic curves and prime fields with special arithmetic properties. In contrast, most existing ECC libraries for 8-bit AVR processors (in particular TinyECC) are optimized for curves and fields that have been standardized by such bodies as the Natinal Institute of Standards and Technology (NIST) [21]. These so-called NIST curves were specified some 15 years ago and do not reflect the current state-of-the-art of ECC in terms of efficiency. Our implementation "departs" from these old standards and puts forward a novel approach for ECC on small sensor nodes that combines twisted Edwards curves (which provide very fast point arithmetic [4]) with so-called Optimal Prime Fields (which allow for efficient modular reduction [24]). Besides achieving high performance, we also aim for a "lightweight" implementation with low RAM and ROM footprint. Therefore, we use the conventional double-and-add method for scalar multiplication, even though one could reach better execution times at the cost of additional memory for storing multiples of the base point. Our results show that an 8-bit sensor node, such as the MICAz mote, is able to perform a full 160-bit scalar multiplication in some $5.8 \cdot 10^6$ clock cycles, which is about 2.78 times faster than the widely-used TinyECC library.

2. OPTIMAL PRIME FIELDS

The specific field we use for our implementation belongs to the family of *Optimal Prime Fields (OPFs)*, which were originally introduced in [11]. These fields are represented by "low-weight" primes of the form $p = u \cdot 2^k + v$, where u and v are relatively small compared to 2^k; in our case, u has a length of 16 bits so that it fits into two 8-bit registers of an AVR processor, and v is equal to 1. A concrete example is $p = 65356 \cdot 2^{144} + 1$, which is a 160-bit prime that looks as follows when written in hex notation.

0xFF4C000000000000000000000000000000000001

Primes of such a form are characterized by a low Hamming weight because only the two most significant bytes and the least significant byte are non-zero; all other "middle" bytes are zero. The low weight of p allows for optimization of the modular arithmetic as only the non-zero bytes of p need to be processed in the reduction operation. For example, the Montgomery multiplication [20] can be optimized for these special primes so that the modular reduction operation has linear complexity, similar to generalized-Mersenne primes like the 192-bit NIST prime $p = 2^{192} - 2^{64} - 1$ that is used in Lederer et al's ECC implementation for WSNs [18].

Zhang et al introduce in [24] an efficient OPF arithmetic library for 8-bit AVR processors end explain how to speed up modular reduction for low-weight primes. They perform the Montgomery multiplication according to the so-called "Finely Integrated Product Scanning" (FIPS) method (see e.g. [17, 12]) and take the low weight of primes of the form $p = u \cdot 2^k + 1$ into account. The ECC implementation we present in this paper uses Zhang et al's OPF library [24] as a building block for low-level field arithmetic. However, we had to write the Assembly code for inversion in OPFs from scratch since it was not part of the library. Our inversion is based on the binary version of the extended Euclidean algorithm [14] and features certain low-level optimizations like e.g. multi-bit shifting. All other arithmetic operations of the OPF library are described in full detail in [24].

3. TWISTED EDWARDS CURVES

In July 2007, Harold Edwards introduced a normal form for elliptic curves along with a simple, symmetric addition law [10]. Bernstein and Lange [5] established the relevance of Edwards' work for elliptic curve cryptography and came up with more efficient formulas for point addition and doubling using standard projective coordinates [14]. They also extended Edwards' curve definition to a more general form that covers a much larger class of elliptic curves. In formal terms, a so-called Edwards curve over a prime field $\mathbb{F}_p$ can be described by the equation[1]

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

with $d \in \mathbb{F}_p \setminus \{0, 1\}$. Edwards curves have some attractive properties for practical use, most notably efficiency of the point arithmetic and completeness of the addition law when d is not a square in $\mathbb{F}_p$. Completeness means the addition formula is valid for all $P, Q \in E(\mathbb{F}_p)$, including the special cases $P = Q$, $P = -Q$, $P = \mathcal{O}$, and $Q = \mathcal{O}$. Bernstein and Lange [5] also showed that every Edwards curve contains a point of order 4 and, thus, has a co-factor of $h \geq 4$.

In 2008, Bernstein et al [4] introduced twisted Edwards curves as a generalization of Edwards curves. Formally, a twisted Edwards curve over a prime field $\mathbb{F}_p$ is defined via the equation

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

where a and d are distinct, non-zero elements of $\mathbb{F}_p$. Bernstein et al observed empirically that the twisted Edwards form covers much more curves than the "original" Edwards form[2] based on Equation 1. Furthermore, as demonstrated in [4], every twisted Edwards curve over a non-binary field $\mathbb{F}_q$ is birationally equivalent over $\mathbb{F}_q$ to a Montgomery curve (i.e. every twisted Edwards curve can be transformed to a Montgomery curve, and vice versa). Bernstein et al [4] also presented explicit formulas for addition and doubling on a twisted Edwards curve; these formulas are complete if a is a square and d a non-square in the underlying field.

3.1 Curve Generation

The security of elliptic curve cryptosystems relies on the computational intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP) [14]. In order to ensure the hardness of the ECDLP, the elliptic curve E and the base point $P \in E(\mathbb{F}_p)$ must satisfy certain properties. First and foremost, the elliptic curve group $E(\mathbb{F}_p)$ needs to contain a large subgroup of prime order n [14]. More precisely, when writing the order of E over $\mathbb{F}_p$ as product of a prime n and a co-factor h, i.e. $\#E(\mathbb{F}_p) = h \cdot n$, then n has to have a bit-length of roughly 160 (or above) and h should be small; in the ideal case $h = 1$. Another important requirement is to avoid certain classes of curves that are "weak" in the sense that the ECDLP can be solved in relatively short time even if n is a 160-bit prime. A famous example are anomalous curves, i.e. curves over $\mathbb{F}_p$ with $\#E(\mathbb{F}_p) = p$ for which the ECDLP has just linear complexity [7]. Furthermore, curves having a "small" embedding degree need to be excluded to

[1]Note that Bernstein and Lange originally defined Edwards curves more generally over non-binary fields. However, in this paper we only consider prime fields.

[2]Of course, the conventional Edwards curves from [5] are a subset of twisted Edwards curves since an Edwards curve is nothing else than a twisted Edwards curve with $a = 1$.

prevent the Menezes-Okamoto-Vanstone (MOV) attack and other attacks based on the Weil and Tate pairing.

Determining whether a random curve E satisfies all the criteria mentioned above requires to find $\#E(\mathbb{F}_p)$, which, in turn, means to count the number of $\mathbb{F}_p$-rational points on E. There exist several computer algebra systems, such as Magma or Sage, that provide functions for calculating the order of an elliptic curve E given by a Weierstraß equation of the form $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$. Using Magma, the generation of a cryptographically strong Weierstraß curve is fairly easy. A straightforward approach is to fix the curve parameter a to -3, initialize the parameter b with 1, and increment b until a suitable curve is found. Fixing a to -3 (i.e. $a = p - 3$) benefits implementation efficiency by reducing the computational cost of point doubling as explained in [14, page 90]. We took this approach to generate secure Weierstraß curves over the 160-bit field given in Section 2 (i.e. the OPF defined by $p = 65356 \cdot 2^{144} + 1$). One of the curves we got is $E : y^2 = x^3 - 3x + 9048$ (i.e. $a = p - 3$ and $b = 9048$), which is suitable for cryptographic applications because it has a prime group order (i.e. $h = 1$) and satisfies all other security requirements discussed above.

Finding a "good" curve in twisted Edwards form is more complicated than the generation of a Weierstraß curve. As mentioned previously, our implementation is based on the prime field $\mathbb{F}_p$ with $p = 65356 \cdot 2^{144} + 1$, which belongs to the family of OPFs [24] and has a size of 160 bits. 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 $a, d \in \mathbb{F}_p$ can be expressed via a non-simplified Weierstraß equation as follows.

$$a_2 = 2(a + d),\ a_4 = (a - d)^2, \text{ and } \ a_1 = a_3 = a_6 = 0 \quad (3)$$

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 [4]. The Magma script we wrote to generate a twisted Edwards curve contains a simple loop in which the parameter d (initially set to 1) gets incremented each iteration until a suitable curve is found. We fixed the parameter a to -1 (i.e. $a = p - 1$) to take advantage of the fast formulas for point addition and doubling presented in [15]. Furthermore, our Magma script only considers values of d that are non-square in $\mathbb{F}_p$ so as to ensure completeness of the addition formula. In each iteration of the loop, three main steps are carried out. First, the twisted Edwards curve defined by a and d is transformed into a Weierstraß curve via Equation (3). Next, we determine the number of $\mathbb{F}_p$-rational points on this curve using Magma's `Order` function and check whether it is four times a prime (i.e. whether its co-factor h is 4). If this is the case then the final step is to carry out some further checks to guarantee the ECDLP is hard. Following the outlined approach, we eventually found the curve $E : -x^2 + y^2 = 1 + 31145x^2y^2$ (i.e. $a = p - 1$ and $d = 31145$), which has an order of $\#E(\mathbb{F}_p) = h \cdot n =$

$$4 \cdot 364371875798791851509551807137352597688979500323,$$

whereby the latter factor is a 158-bit prime. In addition to the already-mentioned requirements for the hardness of the ECDLP, we also checked a couple of other criteria such as "twist security." The quadratic twist E' of our curve E has a small co-factor of 8, which helps to prevent certain forms of implementation attack [3].

3.2 Point Arithmetic

The most efficient way of performing point arithmetic on a twisted Edwards curve is to use the extended coordinates proposed by Hişil et al in [15]. When using this coordinate system, a point $P = (x, y)$ is represented by the quadruple $(X : Y : T : Z)$ where $x = X/Z$, $y = Y/Z$, $xy = T/Z$, and $Z \neq 0$. Such extended twisted Edwards coordinates can be seen as homogenous projective coordinates $(X : Y : Z)$, augmented with a fourth coordinate T that corresponds to the product xy in affine coordinates. The point at infinity $\mathcal{O}$ is represented by $(0 : 1 : 0 : 1)$ and the negative of a point in extended coordinates is $(-X : Y : -T : Z)$. A point given in affine coordinates as (x, y) can be converted to extended coordinates by simply setting $X = x$, $Y = y$, $T = xy$, and $Z = 1$. The re-conversion is done in the very same way as for homogenous projective coordinates through calculation of $x = X/Z$ and $y = Y/Z$, which costs an inversion in the underlying field.

In the following, we briefly explain the dedicated addition and doubling formulas using extended coordinates as given by Hişil et al in [15, Section 3.2 and 3.3]. Let P_1 and P_2 be two distinct points on a twisted Edwards curve represented in extended coordinates of the form $(X_1 : Y_1 : T_1 : Z_1)$ and $(X_2 : Y_2 : T_2 : Z_2)$ with $Z_1 \neq 0$, $Z_2 \neq 0$. When $a = -1$ (as is the case for our curve from Subsection 3.1), a dedicated addition $P_3 = P_1 + P_2 = (X_3 : Y_3 : T_3 : Z_3)$ consists of the following sequence of operations.

$$\begin{gathered} A \leftarrow (Y_1 - X_1) \cdot (Y_2 + X_2),\quad B \leftarrow (Y_1 + X_1) \cdot (Y_2 - X_2), \\ C \leftarrow 2Z_1 \cdot T_2,\quad D \leftarrow 2T_1 \cdot Z_2,\quad E \leftarrow D + C, \\ F \leftarrow B - A,\quad G \leftarrow B + A,\quad H \leftarrow D - C, \\ X_3 \leftarrow E \cdot F,\quad Y_3 \leftarrow G \cdot H,\quad T_3 \leftarrow E \cdot H,\quad Z_3 \leftarrow F \cdot G \end{gathered} \quad (4)$$

The computational cost of this point addition amounts to eight multiplications (8M) and some less costly operations like field additions. When P_2 is given in affine coordinates (i.e. $Z_2 = 1$), the addition (which is then actually a mixed addition) can be performed with only seven multiplications (7M) in $\mathbb{F}_p$. Hişil et al also introduced a fairly fast formula for doubling a point $P_1 = (X_1 : Y_1 : T_1 : Z_1)$ such that the result $P_3 = 2P_1$ is in turn a point in extended coordinates [15]. For twisted Edwards curves with $a = -1$, the sequence of operations to perform a point doubling is as follows.

$$\begin{gathered} A \leftarrow X_1^2,\quad B \leftarrow Y_1^2,\quad C \leftarrow 2Z_1^2,\quad D \leftarrow -A, \\ E \leftarrow (X_1 + Y_1)^2 - A - B, \\ G \leftarrow D + B,\quad F \leftarrow G - C,\quad H \leftarrow D - B, \\ X_3 \leftarrow E \cdot F,\quad Y_3 \leftarrow G \cdot H,\quad T_3 \leftarrow E \cdot H,\quad Z_3 \leftarrow F \cdot G \end{gathered} \quad (5)$$

A point doubling via Equation (5) costs four multiplications (4M) and four squarings (4S) in the underlying field. Unlike to the complete addition formula from [4] and [15, Section 3.1], the doubling via Equation (5) and the point addition formula quoted above do not use the curve parameter d as input. The main disadvantage of the 7M addition formula is that it is not complete, even when a is a square and d a non-square in $\mathbb{F}_p$, i.e. there exist some exceptional cases as explained in [15, Section 2]. However, exceptional cases can only occur in scalar multiplication if the base point P does not have a prime order, which normally never happens in an ECDH protocol for sensor nodes. Of course, a malicious node could try to use a public ECDH key that triggers an

Algorithm 1. Point addition on a TED curve with $a = -1$

Input: Projective point $P_1 = (X_1 : Y_1 : E_1 : H_1 : Z_1)$ with $E_1 H_1 = T_1$, and affine point $P_2 = (U_2 : V_2 : W_2)$ with $U_2 = Y_2 + X_2$, $V_2 = Y_2 - X_2$, $W_2 = 2T_2 = 2X_2Y_2$.
Output: Sum $P_3 = P_1 + P_2 = (X_3 : Y_3 : E_3 : H_3 : Z_3)$.

$T_1 \leftarrow E_1 \cdot H_1$
$E_3 \leftarrow Y_1 - X_1$
$H_3 \leftarrow Y_1 + X_1$
$X_3 \leftarrow E_3 \cdot U_2$
$Y_3 \leftarrow H_3 \cdot V_2$
$E_3 \leftarrow Z_1 \cdot W_2$
$Z_3 \leftarrow 2T_1$
$T_1 \leftarrow Y_3 + X_3$
$Y_3 \leftarrow Y_3 - X_3$
$H_3 \leftarrow Z_3 - E_3$
$E_3 \leftarrow Z_3 + E_3$
$X_3 \leftarrow E_3 \cdot Y_3$
$Z_3 \leftarrow Y_3 \cdot T_1$
$Y_3 \leftarrow T_1 \cdot H_3$
return $(X_3 : Y_3 : E_3 : H_3 : Z_3)$

Algorithm 2. Point doubling on a TED curve with $a = -1$

Input: Projective point $P_1 = (X_1 : Y_1 : E_1 : H_1 : Z_1)$ with $E_1 H_1 = T_1$.
Output: Double $P_3 = 2 \cdot P_1 = (X_3 : Y_3 : E_3 : H_3 : Z_3)$.

$E_3 \leftarrow {X_1}^2$
$H_3 \leftarrow {Y_1}^2$
$T_1 \leftarrow E_3 - H_3$
$H_3 \leftarrow E_3 + H_3$
$X_3 \leftarrow X_1 + Y_1$
$E_3 \leftarrow {X_3}^2$
$E_3 \leftarrow H_3 - E_3$
$Y_3 \leftarrow {Z_1}^2$
$Y_3 \leftarrow 2Y_3$
$Y_3 \leftarrow T_1 + Y_3$
$X_3 \leftarrow E_3 \cdot Y_3$
$Z_3 \leftarrow Y_3 \cdot T_1$
$Y_3 \leftarrow T_1 \cdot H_3$
return $(X_3 : Y_3 : E_3 : H_3 : Z_3)$

exception; in this case, the scalar multiplication produces a wrong result and no shared secret is established.

As noted in Section 4.3 of [15], it is possible to save one multiplication in the doubling formula by mixing extended with "conventional" (i.e. non-extended) coordinates. This cost reduction is simply based on the observation that the auxiliary coordinate T_1 of P_1 is only used in the point addition, but not in the point doubling described by Equation (5). Therefore, the computation of T_3 in Equation (5) can be omitted if a point doubling is followed by another point doubling. Something similar holds for point addition since T_3 in Equation (4) does not need to be computed when the subsequent operation is a point doubling. A straightforward realization of this idea requires the scalar multiplication to take the next operation into account and to incorporate the computation of T_3 into an `if-then` clause that is executed only when needed. Both makes scalar multiplication more complicated and also increases (binary) code size, which is undesirable for resource-constrained embedded devices.

In order to optimize the point doubling without sacrificing code size, we omit the multiplication that produces the auxiliary coordinate T_3 in Equation (5) and output the two factors E, H it is composed of instead. When doing so, the resulting point $P_3 = 2P_1$ comprises five coordinates instead of four, which means P_3 is actually represented in the form of a quintuple $(X_3 : Y_3 : E_3 : H_3 : Z_3)$. In other words, the auxiliary coordinate T is split up into factors E and H so that $EH = T = XY/Z$, thereby saving a multiplication in the point doubling. The subsequently-executed operation can recover T_3, when needed, by simply multiplying E and H. Of course, this optimization of the doubling requires to adapt the point addition accordingly. We implemented the addition formula shown in Equation (4) to output the two factors $E = D + C$ and $H = D - C$ instead of $T_3 = EH$. In this case, when executing an addition using P_1 represented by $(X_1 : Y_1 : E_1 : H_1 : Z_1)$ as input, the auxiliary coordinate $T_1 = E_1 H_1$ has to be computed first since it is needed as operand. However, this modification has no impact on the overall cost of the point addition because the computation of coordinate $T_3 = E_3 H_3$ is simply replaced by forming the product $T_1 = E_1 H_1$. Putting it all together, our optimized implementation of point doubling takes 3M and 4S in the underlying field, while a point addition still needs 7M.

Algorithm 1 details the sequence of field operations for a mixed addition, whereby P_1 is given in extended projective coordinates of the form $(X_1 : Y_1 : E_1 : H_1 : Z_1)$ so that the product $E_1 H_1 = T_1 = X_1 Y_1 / Z_1$. We derived this sequence of field operations directly from Equation (4) and made an effort to minimize the amount of temporary storage needed in the course of the computation of point P_3. In fact, since we use X_3, Y_3, E_3, H_3, Z_3 to store intermediate results, we only need a single temporary variable to accommodate the extra coordinate $T_1 = E_1 H_1$. Equation (4) contains three factors that depend solely on the coordinates of $P2$; these are $Y_2 + X_2$, $Y_2 - X_2$, and $2T_2 = 2X_2Y_2$. Since P_2 is often fixed (e.g. the base point of a scalar multiplication), we can pre-compute these factors to save an addition, a subtraction, and a multiplication by 2 in $\mathbb{F}_p$. Our implementation represents P_2 using extended affine coordinates of the form $(U_2 : V_2 : W_2)$ whereby $U_2 = Y_2 + X_2$, $V_2 = Y_2 - X_2$, and $W_2 = 2T_2 = 2X_2Y_2$. Algorithm 2 is derived from Equation (5) and specifies the sequence of operations in $\mathbb{F}_p$ needed to double a point P_1 given in extended projective coordinates as a quintuple $(X_1 : Y_1 : E_1 : H_1 : Z_1)$. Again, we aimed to minimize the memory requirements so that we just need a single temporary variable. Note that both Algorithm 1 and Algorithm 2 are specifically optimized for twisted Edwards curves with parameter $a = -1$ and do not work for curves with arbitrary a. On the other hand, both algorithms are independent of the second curve parameter d.

There exist a variety of algorithms for computing a scalar multiplication using point additions and doublings; see e.g. [14] for a general overview. The most basic technique is the well-known double-and-add method, which requires to perform n point doublings and about $n/2$ point additions for a scalar k consisting of n bits. The number of additions can be reduced to roughly $n/3$ when k is represented in Non-Adjacent Form (NAF). Of course, there exist several faster methods for computing $k \cdot P$, but they either rely on the pre-computation of multiples of P (which costs RAM) or can only be used if P is fixed and known a-priori. Since we aim for a lightweight implementation with low memory footprint, we decided to adopt the double-and-add method along with a NAF-representation of the scalar k.

4. IMPLEMENTATION RESULTS

We implemented scalar multiplication using a Weierstraß curve and twisted Edwards curve for 8-bit AVR processors such as the ATmega128. In this section we summarize the results for curves over 160 and 192-bit OPFs and compare them with TinyECC [19]. However, our implementation is fully parameterized and can also process fields and curves of larger order. We used Atmel's AVR Studio 4 to compile source codes and build an executable binary file, which we simulated to obtain cycle-accurate execution times for field and group (i.e. curve) operations. The ATmega128 serves as target device for our experiments and simulations since several 8-bit sensor nodes, including the MICAz and IRIS motes [9], are equipped with this specific processor.

When using a 160-bit OPF, the scalar multiplication on the twisted Edwards curve occupies roughly 1037 bytes in RAM. The size of the binary executable file is 18.4 kB.

4.1 Execution Time

We firstly simulated the main field arithmetic operations of our implementation and TinyECC for 160-bit as well as 192-bit operands. Our timings for the OPF operations are (up to a few cycles) the same as that given by Zhang et al in [24]. The execution time of addition, subtraction, multiplication, and squaring are listed in Table 1, along with the cycle count of OPF inversion, which we implemented from scratch since it was not included in Zhang et al's original library. Moreover, the timings of the arithmetic operations of TinyECC are also summarized in Table 1.

Operation	160-bit operands		192-bit operands	
	OPF Lib	TinyECC	OPF Lib	TinyECC
mod_add	531	637	632	832
mod_sub	531	682	632	786
mod_mul	3588	6098	4845	8152
mod_sqr	3032	5725	4052	7493
mod_inv	356650	861901	476055	1305616

Table 1: Execution time (in cycles) of major arithmetic operations in 160 and 192-bit prime fields

As specified in Table 1, a multiplication (including modular reduction) in a 160-bit OPF takes exactly 3588 cycles on an ATmega128. OPF-squaring is about 15% faster than multiplication in an OPF, while addition and subtraction need only one sixth of the multiplication cycles. Inversion is more than 100 times slower than multiplication. Compared with the field arithmetic of TinyECC (which uses pseudo and generalized Mersenne primes), our OPF multiplication and squaring is 1.7 and 1.9 times faster, respectively.

Next, we determined the timings of the point arithmetic operations of our two implementations, one of which uses a Weierstraß curve and the other a twisted Edwards curve (both over the same 160-bit OPF). The cycle counts for a single point addition and point doubling are given in Table 2, along with the simulation results of TinyECC. Our Weierstraß implementation employs the same formulae for the point arithmetic as TinyECC, namely the ones specified in Algorithm 3.21 (doubling) and Algorithm 3.22 (addition) of [14]. Yet, our addition/doubling on the Weierstraß curve outperforms TinyECC significantly, which is primarily due to faster field arithmetic thanks to using an OPF instead of a Mersenne-like prime field. The performance advantage of our work becomes even more apparent when we compare the results on the twisted Edwards curve with TinyECC. In detail, the point addition function on our twisted Edwards curve outperforms TinyECC by a factor of 2.05, while the point doubling is about 1.87 times faster.

Operation	WEI curve	TED curve	TinyECC
Point addition	40222	28792	59070
Point doubling	31536	25994	48483

Table 2: Execution time (in cycles) of addition and doubling on a Weierstraß (WEI) curve and twisted Edwards (TED) curve over a 160-bit OPF

Scalar k	WEI curve	TED curve	TinyECC
Min. weight k	5528120	4359765	11104238
Random integer	7384579	5837612	16241387
Max. weight k	8409321	6658934	22952117

Table 3: Best-case, average, and worst-case execution time (in cycles) of 158-bit scalar multiplication

To evaluate the timings of scalar multiplication $k \cdot P$, we simulated our implementations (and TinyECC) using three different 158-bit values for the scalar. Our twisted Edwards implementation over the 160-bit OPF operates in a cyclic subgroup of $E(\mathbb{F}_p)$ of order n, where n is the 158-bit prime from Section 3.1. When using the double-and-add method the number of point additions is not constant but depends on the non-zero digits in the NAF representation of k. In the first simulation we used $k = 2^{157} = $ 0x200...00, which is the 158-bit scalar of minimal Hamming weight, yielding the best possible execution time. The second simulation was performed with a random 158-bit value of k such that the number of point additions is approximately one third of the bitlength of k, which allows one to determine the average execution time. Finally, we used $k =$ 0x2aa...aa (which is a $k < n$ of maximum NAF weight) as scalar to assess the worst-case time. All results are listed in Table 3.

Our simulation results show that, on average, the twisted Edwards implementation is roughly 2.78 times faster than TinyECC, whereby it must be taken into account that the evaluated version of TinyECC represents k in conventional (i.e. binary) form instead of NAF. Apparently, this overall improvement is bigger than the gain in performance due to the underlying field arithmetic. Consequently, not only the $\mathbb{F}_p$-arithmetic but also the point arithmetic on our twisted Edwards curve is more efficient than that of the Weierstraß curve used by TinyECC. The timings in Table 2 can serve as benchmark for the field arithmetic and show a difference by a factor of roughly 2.0 between our OPF library and the field-operations of TinyECC. Consequently, the field arithmetic contributes a factor of 2.0 (or 71.94%) to the overall performance gain of 2.78. The remaining 28.06% are due to more efficient scalar multiplication on the twisted Edwards curve (i.e. faster point addition and doubling).

4.2 Energy Consumption

The MICAz mote from Crossbow [9] is equipped with an ATmega128 processor clocked at 7.3728 MHz and powered by two 1.5 V batteries. As per [9], this processor draws an average current of 8 mA if the supply voltage is 3 V. Given these figures, we can easily assess the energy consumption of our ECC implementation. A scalar multiplication on the twisted Edwards curve over a 160-bit OPF takes 5837612 clock cycles and, consequently, requires 19.0 mJ energy. We also simulated an implementation using a 192-bit OPF and got an execution time of 9341379 cycles, which corresponds to an energy consumption of 30.4 mJ. This result compares fairly well with recent hardware designs of 192-bit ECC on Edwards curves. For example, Baldwin et al [2] describe an FPGA implementation that consumes 1.623 mJ for a full 192-bit scalar multiplication when using a single ALU. The hardware/software co-design reported by Coyette [8] has an energy consumption of 119 mJ per scalar multiplication.

5. CONCLUSIONS

We introduced a highly-optimized ECC implementation for 8-bit AVR processors such as used in the MICAz mote and several other sensor nodes. Our software is able to perform a 158-bit scalar multiplication in $5.8 \cdot 10^6$ clock cycles on average (i.e. 0.79 seconds on a MICAz mote), which is roughly 2.78 times faster than TinyECC. We achieved this execution time by combining fast $\mathbb{F}_p$ arithmetic (thanks to using an OPF) with very efficient group arithmetic (made possible by using a twisted Edwards curve). TinyECC, on the other hand, is based on standardized fields and curves that were devised some 15 years ago and do not reflect the state-of-the-art. More than two third of the speed-up is due to the faster field arithmetic and about 28% is contributed by the more efficient point addition and doubling formulae of twisted Edwards curves. Going along with this massive performance gain is a reduction of the energy cost of ECC by the same factor. For example, an ECDH key exchange using our ECC software would require only one third (36% to be precise) of the energy of the ECDH implementation contained in TinyECC. As a consequence, our work makes ECDH key exchange much more attractive for WSNs since high energy consumption is generally considered to be the most significant drawback of ECDH and other ECC-based key establishment techniques.

6. REFERENCES

[1] I. F. Akyildiz and M. C. Vuran. *Wireless Sensor Networks.* John Wiley and Sons, 2010.

[2] B. Baldwin, R. Moloney, A. Byrne, G. McGuire, and W. P. Marnane. A hardware analysis of twisted Edwards curves for an elliptic curve cryptosystem. In *Reconfigurable Computing: Architectures, Tools and Applications — ARC 2009*, vol. 5453 of *Lecture Notes in Computer Science*, pp. 355–361. Springer Verlag, 2009.

[3] 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.

[4] 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.

[5] D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In *Advances in Cryptology — ASIACRYPT 2007*, vol. 4833 of *Lecture Notes in Computer Science*, pp. 29–50. Springer Verlag, 2007.

[6] D. W. Carman, P. S. Kruus, and B. J. Matt. Constraints and Approaches for Distributed Sensor Network Security. Technical Report #00-010, NAI Labs, Sept. 2000.

[7] H. Cohen and G. Frey (eds). *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapmann & Hall\CRC, 2006.

[8] A. Coyette. *Embedded Security for Car Telematics and Infotainment.* M.Sc. Thesis, Department of Electrical Engineering (ESAT), Katholieke Universiteit Leuven, Heverlee, Belgium, 2012.

[9] Crossbow Technology, Inc. MICAz Wireless Measurement System. Data sheet, available online at `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`, Jan. 2006.

[10] H. M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, July 2007.

[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 and G.-A. Kamendje. Architectural enhancements for Montgomery multiplication on embedded RISC processors. In *Applied Cryptography and Network Security — ACNS 2003*, vol. 2846 of *Lecture Notes in Computer Science*, pp. 418–434. Springer Verlag, 2003.

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

[15] 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.

[16] M. K. Jain. Wireless sensor networks: Security issues and challenges. *International Journal of Computer and Information Technology*, 2(1):62–67, July 2011.

[17] Ç. K. Koç, T. Acar, and B. S. Kaliski. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, June 1996.

[18] 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.

[19] 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.

[20] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, Apr. 1985.

[21] National Institute of Standards and Technology (NIST). Recommended Elliptic Curves for Federal Government Use. White paper, available online at `http://csrc.nist.gov/encryption/dss/ecdsa/NISTReCur.pdf`, July 1999.

[22] A. Perrig, J. A. Stankovic, and D. Wagner. Security in wireless sensor networks. *Communications of the ACM*, 47(6):53–57, June 2004.

[23] Y. Wang, G. Attebury, and B. Ramamurthy. A survey of security issues in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 8(2):2–23, Apr. 2006.

[24] 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.