Smart Elliptic Curve Cryptography for Smart Dust

Johann Großschädl[1], Matthias Hudler[2], Manuel Koschuch[2], Michael Krüger[2], and Alexander Szekely[3]

[1] University of Luxembourg,
Laboratory of Algorithmics, Cryptology and Security,
6, rue Richard Coudenhove-Kalergi, L–1359 Luxembourg, Luxembourg
johann.groszschaedl@uni.lu
[2] FH Campus Wien – University of Applied Sciences,
Competence Centre for IT-Security,
Favoritenstraße 226, A–1100 Vienna, Austria
firstname.lastname@fh-campuswien.ac.at
[3] Graz University of Technology,
Institute for Applied Information Processing and Communications,
Inffeldgasse 16a, A–8010 Graz, Austria
alexander.szekely@iaik.tugraz.at

Abstract. Wireless ad-hoc and sensor networks play a vital role in an ever-growing number of applications ranging from environmental monitoring over vehicular communication to home automation. Security and privacy issues pose a big challenge for the widespread adoption of these networks, especially in the automotive domain. The two most essential security services needed to maintain the proper functioning of a wireless network are authentication and key establishment; both can be realized with Elliptic Curve Cryptography (ECC). In this paper, we introduce an efficient ECC implementation for resource-restricted devices such as sensor nodes. Our implementation uses a 160-bit Optimal Prime Field (OPF) over which a Gallant-Lambert-Vanstone (GLV) curve with good cryptographic properties can be defined. The combination of optimized field arithmetic with fast group arithmetic (thanks to an efficiently computable endomorphism) allows us to perform a scalar multiplication in about $5.5 \cdot 10^6$ clock cycles on an 8-bit ATmega128 processor, which is significantly faster than all previously-reported ECC implementations based on a 160-bit prime field.

Keywords: Ad-hoc network, elliptic curve cryptography, performance evaluation, arithmetic in finite fields, endomorphism.

1 Introduction

The term "smart dust" refers to miniature computing devices with sensing and wireless networking capabilities [26]. Current-generation micro-sensors, such as the Crossbow MICA2DOT mote [4], have a volume of a few cubic-centimeters

X. Zhang and D. Qiao (Eds.): QShine 2010, LNICST 74, pp. 623–634, 2011.

including battery. However, it can be expected that progress in miniaturization will reduce the size of motes significantly over the next couple of years. These tiny devices can form a Wireless Sensor Network (WSN) and undertake certain tasks such as battlefield surveillance or the monitoring of weather and/or road conditions for a traffic-control system [1]. Kristofer Pister, who first coined the term "smart dust," forecasted in the mid-90s that "in 2010 MEMS sensors will be everywhere and sensing virtually everything" [18]. Today, much of Pister's vision from some 15 years ago has become reality; we may think of miniature sensor nodes used for environmental surveillance, home automation, or medical monitoring [1]. Widespread deployment of sensors for intelligent transportation systems is expected in the near future, e.g. WSNs to monitor traffic conditions and report these conditions to smart vehicles.

In October 1999, the Federal Communications Commission (FCC) allocated a 75 MHz band in the 5.9 GHz frequency range for exclusive use by automotive applications and intelligent transportation systems [5]. Dedicated Short Range Communications (DSRC) is a suite of protocols and standards for wireless networking within this 75 MHz band [2]. The main purpose of DSRC is to enable short-range communication among vehicles and between vehicles and roadside infrastructure (e.g. traffic condition sensors) with the goal of increasing safety on roads and improving traffic flow [13]. Higher-level protocols operating above DSRC (or IEEE 802.11p) are specified in the IEEE standard 1609 [24]. Typical fields of application of DSRC (resp. WAVE) include safety measures (e.g. road condition warning, collision avoidance), traffic management (e.g. variable speed limits, intelligent traffic lights), driver assistance (e.g. parking aids, cruise control), and electronic payment (e.g. toll collection, parking fees). Several of these applications pose significant challenges to security and privacy, as was pointed out in [12,15,19]. To address these issues, the IEEE 1609.2 standard contains a number of measures to ensure the confidentiality, integrity, and authenticity of messages exchanged over DSRC. The public-key cryptosystems specified in IEEE 1609.2 are based on Elliptic Curve Cryptography (ECC) [10] to achieve a balance between efficiency and security. More precisely, IEEE 1609.2 defines ECIES for asymmetric encryption and ECDSA as signature primitive [15].

In this paper, we introduce an optimized ECC implementation for wireless ad-hoc and sensor networks. Our implementation uses an Optimal Prime Field (OPF) as underlying algebraic structure to facilitate fast modular reduction on different platforms [7]. Furthermore, we take advantage of a Gallant-Lambert-Vanstone (GLV) elliptic curve with an efficiently computable endomorphism to accelerate the scalar multiplication [6]. The focus of our implementation lay on high performance, low memory footprint, and low register usage. We evaluated the performance of our ECC software on a Crossbow Micaz mote featuring an 8-bit ATmega128 processor clocked at 7.37 MHz. The combination of fast field arithmetic with fast curve arithmetic allowed us to perform a full scalar multiplication over a 160-bit OPF in roughly $5.5 \cdot 10^6$ clock cycles, which represents a new speed record for 160-bit ECC on 8-bit processors. Our ECC software can be easily ported to other platforms and achieves excellent performance also on

processors with few general-purpose registers as we do not rely on the hybrid multiplication technique [9]. The results we present in this paper demonstrate that strong public-key cryptography is feasible for resource-constrained devices such as sensor nodes.

2 Elliptic Curve Cryptography

In this section (which is largely based on our previous work [16]), we introduce the basic concepts of ECC with a focus on implementation aspects. In short, an elliptic curve E over a prime field $\mathbb{F}_p$ can be formally defined as the set of all tuples $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ satisfying a Weierstraß equation of the form

$$y^2 = x^3 + ax + b \quad \text{with} \quad a, b \in \mathbb{F}_p \tag{1}$$

These tuples are called *points* with x and y referred to as coordinates. The set of points together with a special point $\mathcal{O}$ (the so-called *point at infinity*) allows one to form a commutative group with $\mathcal{O}$ being the identity element. The group operation is the addition of points, which can be performed through arithmetic operations (addition, subtraction, multiplication, squaring, and inversion) in the underlying field $\mathbb{F}_p$ according to well-defined formulae (see e.g. [10]). Adding a point $P = (x, y)$ to itself is referred to as point doubling and can also be done through a well-defined sequence of operations in $\mathbb{F}_p$. In general, point doubling requires fewer field operations than the addition of two points.

The *order* of an elliptic curve group $E(\mathbb{F}_p)$ is the number of $\mathbb{F}_p$-rational points on the curve E, plus one for the point at infinity. It is well known from Hasse's theorem that $\#E(\mathbb{F}_p)$ has the following bounds:

$$p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p} \tag{2}$$

For cryptographic applications, $\#E(\mathbb{F}_p)$ should have a large prime factor; in the ideal case it is a prime. Before ECIES encryption (or any other elliptic curve scheme) can be carried out, the involved parties have to agree upon a common set of so-called *domain parameters*, which specify the finite field $\mathbb{F}_p$, the elliptic curve E (i.e. the coefficients $a, b \in \mathbb{F}_p$ defining E according to Equation (1)), a base point $P \in E(\mathbb{F}_p)$ generating a cyclic subgroup of large order, the order n of this subgroup, and the co-factor $h = \#E(\mathbb{F}_p)/n$. Consequently, elliptic curve domain parameters over $\mathbb{F}_p$ are simply a sextuple $D = (p, a, b, P, n, h)$ [10]. In elliptic curve cryptography, a private key is an integer k chosen randomly from the interval $[1, n-1]$. The corresponding public key is the point $Q = k \cdot P$ on the curve. Given k and P, the point $Q = k \cdot P$ can be obtained by means of an operation called *scalar multiplication* [10]. Numerous algorithms for scalar multiplication have been proposed; the simplest way to compute $k \cdot P$ is to perform a sequence of point additions and doublings, similar to the square-and-multiply algorithm for modular exponentiation.

While a scalar multiplication of the form $Q = k \cdot P$ can be calculated quite efficiently, the inverse operation, i.e. finding k when P and Q are given, is a hard

mathematical problem known as the *Elliptic Curve Discrete Logarithm Problem (ECDLP)*. To date, the best algorithm known for solving the ECDLP requires fully exponential time if the domain parameters were chosen with care [10]. In contrast, the best algorithm for solving the Discrete Logarithm Problem (DLP) in $\mathbb{Z}_p^*$ or the Integer Factorization Problem (IFP) has a sub-exponential running time. As a consequence, elliptic curve cryptosystems can use much shorter keys compared to their "classical" counterparts based on the DLP or IFP. A common rule of thumb states that a properly designed 160-bit ECC scheme is about as secure as 1024-bit RSA.

2.1 Scalar Multiplication

The computationally expensive part of virtually all elliptic curve cryptosystems is scalar multiplication, an operation of the form $k \cdot P$ where k is an integer and P is a point on the curve. A scalar multiplication can be performed by means of repeated point additions and point doublings, both of which, in turn, involve a sequence of arithmetic operations (i.e. addition, multiplication, squaring, and inversion) in the underlying finite field. Inversion is by far the most expensive operation in prime fields [10]. However, it is possible to add points on an elliptic curve without the need to perform costly inversions, e.g. by representing the points in *projective coordinates* [10]. In Section 2 we described the conventional (i.e. affine) coordinate system in which a point P is associated with an x and a y coordinate, i.e. a tuple $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$. By contrast, in projective coordinate systems, a point is represented by a triplet (X, Y, Z), which corresponds to the affine coordinates $(X/Z^u, Y/Z^v)$ when $Z \neq 0$ (u and v depend on the specific coordinate system chosen). For example, the projective point $P = (X, Y, Z)$ in Jacobian coordinates corresponds to the affine point $P = (X/Z^2, Y/Z^3)$. It is also possible to add two points when one is given in projective coordinates and the other in affine coordinates [10]. In fact, such mixed coordinates often lead to very efficient point addition formulae. For example, adding a point in Jacobian coordinates to an affine point requires eight multiplications and three squarings in $\mathbb{F}_p$ (but no inversion). Doubling a point given in Jacobian coordinates takes four multiplications and four squarings.

The *double-and-add* algorithm performs a scalar multiplication via repeated point additions and doublings, analogous to the multiply-and-square algorithm for modular exponentiation. It uses the binary expansion of the integer k and computes $k \cdot P$ as follows: For each bit k_i of k, the current intermediate result is doubled, and the base point P is added if bit $k_i = 1$ (no addition is performed when $k_i = 0$). Given an l-bit scalar k, the double-and-add algorithm executes exactly l point doublings, whereas the number of point additions depends on the Hamming weight of k. In the average case $l/2$ additions are carried out; the worst-case number of additions is l. The conventional double-and-add method can be easily improved by using a signed-digit representation of k. One option is the *non-adjacent form (NAF)*, which reduces the number of additions (of either P or $-P$) to $l/3$ in the average case and $l/2$ in the worst case [10]. However, the number of point doublings remains the same.

2.2 Arithmetic in Prime Fields

As mentioned before, the execution time of a scalar multiplication depends on the efficiency of the arithmetic in the underlying finite field. The elements of a prime field $\mathbb{F}_p$ are the integers $0, 1, \ldots, p-1$, and the arithmetic operations are addition and multiplication modulo p. In ECC, primes of a length of between 160 and 512 bits are typically used. Consequently, the field elements can not be directly processed, but must be represented by arrays of single-precision words (e.g. arrays of unsigned n-bit integers when working on an n-bit processor). In our case, i.e. 160-bit ECC on an 8-bit processor, each field element is stored in an array of 20 bytes. Arithmetic in $\mathbb{F}_p$ is similar to that in $\mathbb{Z}_p^*$ as needed for the implementation of RSA and other "classical" public-key schemes. Therefore, all algorithms for modular arithmetic in $\mathbb{Z}_p^*$ are directly applicable to $\mathbb{F}_p$ as well (e.g. Montgomery reduction [14]). However, it is possible and common practice to use special primes in ECC for which optimized modular reduction methods exist; a typical example are generalized-Mersenne (GM) primes [10].

There are two basic algorithms for multi-precision multiplication: one is the *operand-scanning method* (also called row-wise multiplication) and the other is the *product-scanning method* (column-wise multiplication) [9,10]. Both require the same number of single-precision multiplications (i.e. `mul` instructions on an ATmega128), namely 400 in our case of 160-bit operands, but they differ in the number of memory accesses and single-precision additions. We first describe the original operand and product scanning methods, which operate on 8-bit words (i.e. bytes) when implemented on an ATmega processor. Then, we sketch the hybrid multiplication method of Gura et al [9], which combines the advantages of operand scanning and product scanning to reduce the total number of load instructions. The operand-scanning method has a nested-loop structure with a relatively simple inner loop. Each iteration executes an operation of the form $a \cdot b + c + d$ with a, b, c, d denoting 8-bit words (i.e. bytes). On an ATmega this operation requires one `mul` instruction to produce the partial product $a \cdot b$, and a total of four `add` (resp. `adc`) instructions to add the two bytes c and d to the 16-bit quantity $a \cdot b$. Furthermore, two load (`ld`) instructions and a store (`st`) are executed in each iteration. On the other hand, the product-scanning method performs a multiply-accumulate operation in its inner loop, i.e. two bytes are multiplied and the 16-bit partial product is added to a cumulative sum held in three registers. The product-scanning method also executes two `ld` instructions per iteration, but no store.

The execution time of the conventional product-scanning technique can be vastly improved when the processor features a large number of general-purpose registers, which is the case with the ATmega128 [3]. The *hybrid* multiplication method, introduced by Gura et al in [9], works similar as the product-scanning technique, but processes $d \geq 2$ bytes of the operands at a time, which reduces the number of required loop iterations by a factor of d. In each iteration of the inner loop, $d \geq 2$ bytes of the operands are loaded from RAM, then multiplied together using d^2 `mul` instructions, and added to a running sum. However, the hybrid method can not be applied on processors with few registers.

3 Our Implementation

In this section, we describe our ECC implementation in detail and analyze its performance on an 8-bit ATmega128 processor [3]. Our implementation differs from previous work (e.g. [9,16,17,22,23,25]) in two main aspects. First, we combine efficient finite-field arithmetic with fast group arithmetic; the former is achieved by using an Optimal Prime Field (OPF) [7] as "underlying" algebraic structure, while the scalar multiplication profits from an efficiently computable endomorphism provided by a so-called Gallant-Lambert-Vanstone (GLV) curve [6]. The basic concepts of OPFs (including a formal definition of OPFs, how to do arithmetic in OPFs, and how to construct a GLV curve over an OPF) can be found in [7]; here in this paper we show the applicability of OPFs to secure ad-hoc and sensor networks. A second difference to related work is the concrete implementation of the multi-precision multiplication. We do not use the hybrid multiplication method from [9], but apply a special loop unrolling technique to achieve high performance on a wide variety of platforms, including processors with a small number of general-purpose registers. The hybrid multiplication, on the other hand, can only be implemented when a large number of registers is available, which is not always the case. Even though the ATmega128 features a large register file with 32 registers (which means that hybrid multiplication is possible), we decided to use it as experimental platform for our implementation in order to allow for direct comparison with previous work. The ATmega128 is a simple 8-bit RISC processor based on the AVR instruction set, i.e. the usual arithmetic and logical instructions are supported, including an integer-multiply instruction with a 2-cycle latency [3].

3.1 Optimal Prime Field (OPF)

An Optimal Prime Field (OPF) [7] is a finite field defined by a prime p of the form $u \cdot 2^k + v$, where u and v are constants that fit into a single register of the target processor (or, more precisely, $0 < u, v < 2^w$ where w denotes the word size of the target processor). In our case, both u and v are 8-bit constants since our implementation is optimized for an 8-bit processor. The specific prime we chose is $p = 232 \cdot 2^{152} + 99$, which happens to be a 160-bit prime that looks as follows when written in hex notation:

$$p = \text{0xE800000000000000000000000000000000000063} \tag{3}$$

OPFs are characterized by a low Hamming weight [7]. In particular, when p is stored in an array of w-bit words, only the most and least significant words are non-zero; all words between them are zero. The low weight of these primes allows for efficient software implementation of the modular reduction operation because only the two non-zero words of p need to be processed [7]. Well-known modular reduction methods, including Montgomery and Barrett reduction, can be well optimized for low-weight primes such that the reduction operation has linear complexity, similar to generalized-Mersenne (GM) and pseudo-Mersenne

(PM) primes [10,20]. A particular advantage of OPFs over GM prime fields is their flexibility; there exist a large number of OPFs for a given bitlength, while the number of GM primes with "good" arithmetic properties is rather limited [20]. The large number of OPFs for a given bitlength facilitates the construction of a GLV curve with an efficiently computable endomorphism suitable for the implementation of ECC, which is not possible with the GM primes specified by the NIST (see [7] for further details).

We implemented multiplication and squaring in OPFs using Montgomery's modular reduction method. More precisely, we optimized the Finely Integrated Operand Scanning (FIPS) method [14] for Montgomery multiplication with respect to the low weight of our OPF so that only the non-zero bytes $p_{19} = 232$ and $p_0 = 99$ of the prime $p = 232 \cdot 2^{152} + 99$ are processed. An implementation of the FIPS technique for "arbitrary" primes executes $2s^2 + s$ single-precision multiplications (i.e. `mul` instructions) when the operands consist of s words. In our case of processing 160-bit operands on an 8-bit processor (i.e. $s = 20$), this amounts to 820 `mul` instructions for a single Montgomery multiplication. However, after optimizing the FIPS technique taking into account that $s - 2$ bytes of p are 0, only only $s^2 + 3s$ single-precision multiplications have to be carried out, which results in 460 `mul` instructions for 160-bit operands. A conventional multiplication of two s-word operands (without reduction operation) requires s^2 `mul` instructions [10,14]; consequently, the overhead of modular reduction in our OPF is $3s$ `mul` instructions, i.e. scales linearly with the operand length.

The optimized FIPS method for Montgomery multiplication in an OPF has a nested-loop structure with a simple inner loop in which a multiply-accumulate operation is performed, i.e. two bytes are multiplied and the 16-bit product is added to a running sum held in three registers. More precisely, each iteration of the inner loop executes two `ld`, one `mul`, one `add`, and two `adc` instructions on an Atmega128, which takes nine clock cycles altogether. Furthermore, three clock cycles of loop overhead (i.e. increment or decrement of a loop counter and branch instruction) contribute to the execution time. Since this overhead constitutes 25% of the overall execution time of the inner loop, it makes sense to apply techniques for reducing loop overhead, such as loop unrolling or hybrid multiplication. The basic idea of hybrid multiplication, as described in [9], is to process $d > 1$ bytes of the operands in each iteration of the inner loop (instead of just a single byte), which reduces the overall number of loop iterations by a factor of d. However, hybrid multiplication is only advantageous on processors featuring a large number of registers; for example, when $d = 4$ (i.e. four bytes of each operand are processed per loop iteration), 14 registers are necessary to store the bytes of the operands and running sum [9].

As already mentioned, we did not apply the hybrid multiplication technique since we aimed for a flexible implementation that can be easily ported to various different platforms, including processors with a small number of registers. Instead, we reduced the loop overhead of our optimized FIPS method for OPFs by unrolling the inner loop. In general, loop unrolling is done by replicating the loop body a certain number of times and adjusting (i.e. reducing) the iteration

Table 1. Execution times (in clock cycles) of arithmetic operations in OPFs of length 128, 144, 160, 176, and 192 bits on an ATmega128 processor

Operation	128 bit	144 bit	160 bit	176 bit	192 bit
Addition	506	560	614	668	722
Subtraction	514	562	610	672	731
Multiplication	3,598	4,358	5,239	6,143	7,070
Squaring	2,967	3,488	4,086	4,642	5,277
Inversion	110,414	133,698	157,840	185,361	214,687

count accordingly. Loop unrolling improves the performance since the overhead for updating and testing the loop counter and branching back to the beginning of the loop is executed less frequently. However, this gain in performance comes at the expense of an increase in code size. To achieve a proper balance between these two metrics of interest, we decided to unroll only the inner loop. Unfortunately, the number of inner-loop iterations of both the standard FIPS method and our optimized OPF-variant is not constant, but varies between 1 and s. An efficient unrolling technique for such loops is Duff's device [11], as was shown in [8] for the standard FIPS method. We applied the Duff device from [8] to the optimized FIPS method for our 160-bit OPF, which means that we replicated the body of the inner loop 20 times. This approach for loop unrolling achieves almost the same performance as "full" loop unrolling (i.e. unrolling of both the inner and the outer loop), but does so at a faction of the code size.

Table 1 summarizes the execution time of addition, subtraction, multiplication, squaring, and inversion in OPFs of different size, ranging from 128 to 192 bits. The addition in an OPF is performed by first calculating the sum of two field elements, followed by a conditional subtraction of the prime p. Hence, the addition time (and also the subtraction time) is not constant but depends on whether or not a subtraction (resp. addition) of p is carried out. Also both the multiplication and squaring may require a final subtraction of p, which impacts the execution time. Our OPF multiplication with unrolled inner loop is a little slower than the hybrid multiplication from [9], but occupies only five registers for operands and the running sum, while the hybrid technique with $d = 4$ needs 14 registers.

3.2 Gallant-Lambert-Vanstone (GLV) Curve

In order to speed up the scalar multiplication $k \cdot P$, we use a so-called Gallant-Lambert-Vanstone (GLV) curve as introduced in [6]. This family of curves can be described by a Weierstraß equation of the form $y^2 = x^3 + b$ (i.e. $a = 0$ and $b \neq 0$) or $y^2 = x^3 + ax$ (i.e. $a \neq 0$ and $b = 0$) over a prime field $\mathbb{F}_p$ and features an efficiently computable endomorphism ϕ whose characteristic polynomial has small coefficients. Such an endomorphism provides the possibility to calculate a scalar multiplication as $k \cdot P = k_1 \cdot P + k_2 \cdot \phi(P)$, which is more efficient than a straightforward calculation of $k \cdot P$ since k_1 and k_2 have typically only half the

bitlength of k and the two multiplications $k_1 \cdot P$ and $k_2 \cdot \phi(P)$ can be performed simultaneously (i.e. in an interleaved fashion) using Shamir's trick as explained in [10, Section 3.3.3]. However, as already mentioned, a GLV curve with good cryptographic properties can not be constructed over any prime field; in particular, it is not possible to find a GLV curve of prime order over the GM prime fields standardized by the NIST (see [7] for further details).

Since there exist a large number of OPFs, it is not hard to find an OPF of a given bitlength such that a GLV curve with good cryptographic properties can be constructed over it. Our implementation uses the GLV curve

$$E : y^2 = x^3 + 3 \tag{4}$$

(i.e. $a = 0$ and $b = 3$) over the prime field $\mathbb{F}_p$ defined by $p = 232 \cdot 2^{152} + 99$. The group of points on this curve has prime order, namely

$$\#E(\mathbb{F}_p) = n = 1324485858831130769622088630463083182986367428713 \tag{5}$$

and satisfies all properties listed in [21, Section 3.1.1.1], which means that this curve can be securely used for the implementation of ECC. Consequently, this specific pair of OPF and GLV curve offers good arithmetic and cryptographic properties. In the following, we summarize some basic facts about our special curve, analogously to [10, page 125]. Since $a = 0$ and $p \equiv 1 \bmod 3$, our curve is of the type described in Example 4 in [6]. The underlying field $\mathbb{F}_p$ contains an element β of order 3 (since $p \equiv 1 \bmod 3$); our implementation uses

$$\beta = 10393645858606913233375911664120954873303254970 64 \tag{6}$$

According to [6, Example 4], the map $\phi : E \to E$ defined by

$$\phi : (x, y) \mapsto (\beta x, y) \quad \text{and} \quad \phi : \mathcal{O} \mapsto \mathcal{O} \tag{7}$$

is an endomorphism of E defined over $\mathbb{F}_p$. The characteristic polynomial of ϕ is $\lambda^2 + \lambda + 1$. In order to exploit this endomorphism for scalar multiplication, we need a root modulo n of the characteristic polynomial, i.e. we need a solution to the equation $\lambda^2 + \lambda + 1 \equiv 0 \bmod n$; for our implementation we use

$$\lambda = 89368587362047950552629335219870406524271965560 9 \tag{8}$$

The solution λ has the property that $\phi(P) = \lambda P$ for all $P \in E(\mathbb{F}_p)$ [6,10]. Note that computing $\phi(P)$ for a point $P = (x, y)$ requires only one multiplication in $\mathbb{F}_p$, namely $\beta \cdot x$.

When using a GLV curve [6], the common strategy for computing $k \cdot P$ is to decompose the scalar k into two "half-length" integers k_1 and k_2 (referred to as balanced length-two representation of k) such that $k = k_1 + k_2\lambda \bmod n$. This decomposition of k into k_1 and k_2 is described in detail in [6] and [10]. Because $k \cdot P = k_1 \cdot P + k_2 \cdot \lambda \cdot P = k_1 \cdot P + k_2 \cdot \phi(P)$, the result of $k \cdot P$ can be obtained by first computing $\phi(P)$ (which takes just a single field multiplication) and then using simultaneous multiple point multiplication ("Shamir's trick") to perform

Table 2. Execution times (in million clock cycles) of 160-bit scalar multiplication

Implementation	Field type	Fixed P.	Rand. P.	Notes
Wang and Li [25]	PM prime	9.14	9.95	Sliding window
Szczechowiak et al. [22]	GM prime	9.38	9.38	Comb method
Ugus et al. [23]	PM prime	5.09	7.59	MOV, 1 prec. point
Liu and Ning [17]	PM prime	15.05	15.05	Sliding window
Gura et al. [9]	PM prime	6.48	6.48	NAF
Our implementation	OPF	5.48	5.48	GLV curve

these two half-length scalar multiplications in an interleaved fashion. A conventional computation of $k \cdot P$ using the double-and-add approach requires a total of l point doublings and on average $l/2$ point additions, whereby l refers to the bitlength of k. The number of additions can be reduced to $l/3$ by representing the scalar k in Non-Adjacent Form (NAF). On the other hand, our GLV curve allows us to obtain $k \cdot P$ with only $l/2$ doublings and $l/4$ additions when the two half-length scalars k_1 and k_2 are represented in Joint Sparse Form (JSF) as described in [10]. Consequently, the endomorphism of our GLV curve halves the number of point doublings and reduces the number of point additions by some 8.3% on average.

Our implementation performs the point (i.e. curve) arithmetic using mixed Jacobian-affine coordinates as detailed in [10, Section 3.2.2]. More precisely, we implemented the point addition and doubling according to Algorithm 3.2.2 and Algorithm 3.2.1, respectively, whereby we optimized the latter with respect to $a = 0$ such that a point doubling can be carried out with three multiplications in $\mathbb{F}_p$ instead of four [7]. The addition of points over our 160-bit OPF requires roughly 57,760 clock cycles on an ATmega128 processor, while a point doubling is executed in about 35,450 cycles. Our implementation of the point arithmetic is written in ANSI C and directly calls the Assembly-language functions for the OPF arithmetic. A scalar multiplication executes in $5.48 \cdot 10^6$ clock cycles on an ATmega128 processor when exploiting Shamir's trick and representing the two half-length scalars in JSF. The comparison with related work in Table 2 shows that our implementation sets a new speed record for ECC on an ATmega128.

4 Conclusions

In this paper, we introduced an efficient ECC implementation for DSRC-based ad-hoc and sensor networks that realize security services (e.g. authentication) according to the IEEE standard 1609.2. Our implementation is able to perform a 160-bit scalar multiplication on an ATmega128 processor in slightly less than $5.5 \cdot 10^6$ clock cycles, which establishes a new performance record for ECC on 8-bit platforms. Compared to previous work, our implementation advances the state-of-the-art in two main aspects. First, we combine efficient field arithmetic (thanks to the use of an OPF) with fast scalar multiplication on a GLV curve

by exploiting an efficiently computable endomorphism. Second, we do not use Gura's hybrid multiplication technique, but unroll the inner loops of our field multiplication and squaring operations following "Duff's device" to reduce the execution time. Therefore, our ECC software can be easily ported to various other platforms, even to processors with very few general-purpose registers on which hybrid multiplication would not work.

Acknowledgements. The authors are grateful to Stefan Mendel for his contributions to the Assembly-language implementation of the OPF arithmetic.

Manuel Koschuch, Matthias Hudler and Michael Krüger have been supported by the MA27 – EU-Strategie und Wirtschaftsentwicklung – in the course of the funding programme "Stiftungsprofessuren und Kompetenzteams für die Wiener Fachhochschul-Ausbildungen."

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Çayirci, E.: Wireless sensor networks: A survey. Computer Networks 38(4), 393–422 (2002)
2. ASTM International: ASTM E2213-03 Standard Specification for Telecommunications and Information Exchange Between Roadside and Vehicle Systems — 5 GHz Band Dedicated Short Range Communications (DSRC) Medium Access Control (MAC) and Physical Layer (PHY) Specifications (2003)
3. Atmel Corporation: 8-bit ARV® Microcontroller with 128K Bytes In-System Programmable Flash: ATmega128, ATmega128L. Datasheet (June 2008), `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`
4. Crossbow Technology, Inc.: MICA2DOT Wireless Microsensor Mote. Data sheet (January 2006), `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2DOT_Datasheet.pdf`
5. Federal Communications Commission (FCC): FCC Allocates Spectrum in 5.9 GHz Range for Intelligent Transportation Systems Uses. News release (October 1999), `http://www.fcc.gov/Bureaus/Engineering_Technology/News_Releases/1999/nret9006.html`
6. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Heidelberg (2001)
7. Großschädl, J., et al.: Optimal prime fields for use in elliptic curve cryptography (2010) (preprint, submitted for publication)
8. Großschädl, J., Tillich, S., Szekely, A.: Performance evaluation of instruction set extensions for long integer modular arithmetic on a SPARC V8 processor. In: Proceedings of the 10th Euromicro Conference on Digital System Design (DSD 2007), pp. 680–689. IEEE Computer Society Press, Los Alamitos (2007)
9. Gura, N., Patel, A., Wander, A.S., Eberle, H., Chang Shantz, S.: Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In: Joye, M., Quisquater, J.J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 119–132. Springer, Heidelberg (2004)
10. Hankerson, D.R., Menezes, A.J., Vanstone, S.A.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2004)

11. Holly, R.: A reusable Duff device. Dr. Dobb's Journal 30(8), 73–74 (2005)
12. Hubaux, J.P., Ĉapkun, S., Luo, J.: The security and privacy of smart vehicles. IEEE Security & Privacy 2(3), 49–55 (2004)
13. Jiang, D., Taliwal, V., Meier, A., Holfelder, W., Herrtwich, R.G.: Design of 5.9 GHz DSRC-based vehicular safety communication. IEEE Wireless Communications 13(5), 36–43 (2006)
14. Koç, Ç.K., Acar, T., Kaliski, B.S.: Analyzing and comparing Montgomery multiplication algorithms. IEEE Micro 16(3), 26–33 (1996)
15. Laurendeau, C., Barbeau, M.: Threats to security in DSRC/WAVE. In: Kunz, T., Ravi, S.S. (eds.) ADHOC-NOW 2006. LNCS, vol. 4104, pp. 266–279. Springer, Heidelberg (2006)
16. Lederer, C., Mader, R., Koschuch, M., Großschädl, J., Szekely, A., Tillich, S.: Energy-efficient implementation of ECDH key exchange for wireless sensor networks. In: Markowitch, O., Bilas, A., Hoepman, J.H., Mitchell, C.J., Quisquater, J.J. (eds.) WISTP 2009. LNCS, vol. 5746, pp. 112–127. Springer, Heidelberg (2009)
17. Liu, A., Ning, P.: TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In: Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008), pp. 245–256. IEEE Computer Society Press, Los Alamitos (2008)
18. Pister, K.S.: My view of sensor networks in 2010 (2010), `http://robotics.eecs.berkeley.edu/~pister/SmartDust/in2010`
19. Raya, M., Hubaux, J.P.: The security of vehicular ad hoc networks. In: Atluri, V., Ning, P., Du, W. (eds.) Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN 2005), pp. 11–21. ACM Press, New York (2005)
20. Solinas, J.A.: Generalized Mersenne numbers. Tech. Rep. CORR-99-39, Centre for Applied Cryptographic Research (CACR), University of Waterloo, Waterloo, Canada (1999)
21. Standards for Efficient Cryptography Group (SECG): SEC 1: Elliptic Curve Cryptography. Working draft, version 1.7 (November 2006), `http://www.secg.org/download/aid-631/sec1_1point7.pdf`
22. Szczechowiak, P., Oliveira, L.B., Scott, M., Collier, M., Dahab, R.: NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In: Verdone, R. (ed.) EWSN 2008. LNCS, vol. 4913, pp. 305–320. Springer, Heidelberg (2008)
23. Ugus, O., Westhoff, D., Laue, R., Shoufan, A., Huss, S.A.: Optimized implementation of elliptic curve based additive homomorphic encryption for wireless sensor networks. In: Wolf, T., Parameswaran, S. (eds.) Proceedings of the 2nd Workshop on Embedded Systems Security (WESS 2007), pp. 11–16 (2007), `http://arxiv.org/abs/0903.3900`
24. U.S. Department of Transportation: IEEE 1609 – Family of standards for wireless access in vehicular environments (WAVE). ITS standards fact sheet (September 2009), `http://www.standards.its.dot.gov/fact_sheet.asp?f=80`
25. Wang, H., Li, Q.: Efficient implementation of public key cryptosystems on mote sensors. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 519–528. Springer, Heidelberg (2006)
26. Warneke, B., Last, M., Liebowitz, B., Pister, K.S.: Smart dust: Communicating with a cubic-millimeter computer. Computer 34(1), 44–51 (2001)