Fast Arithmetic on ATmega128 for Elliptic Curve Cryptography

Anton Kargl and Stefan Pyka and Hermann Seuschek*

Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 München, Germany
{anton.kargl|stefan.pyka|hermann.seuschek}@siemens.com

Abstract. Authentication protocols are indispensable in wireless sensor networks. Commonly they are based on asymmetric cryptographic algorithms. In this paper we investigate all categories of finite fields suitable for elliptic curve cryptography on the ATmega128 microcontroller: $\mathbb{F}_p$, $\mathbb{F}_{2^d}$, and $\mathbb{F}_{p^d}$. It turns out that binary fields enable the most efficient implementations.

Key words: Optimal Extension Field, Binary Fields, Embedded Security.

1 Introduction

Elliptic curves are very attractive for cryptosystems implemented on constrained devices, because of their short key lengths, small system parameters, and high performance. Elliptic curve cryptosystems can be used for authentication, digital signatures, and public-key encryption [18]. In particular, authentication schemes based on elliptic curves enable public-key authentication on low-cost devices like RFIDs [7] or wireless sensor nodes [25].

Recently a lot of authors studied the efficiency of elliptic curve cryptosystems on microcontrollers which are used in wireless sensor motes. They focus on the TI MSP430 and the ATmega128 developed by Atmel. Gura et al. first reported the efficiency of elliptic curve cryptosystems defined over prime fields on the ATmega128 in [16]. They used curves published in the recommendations of SEC [8]. The execution time of one ECDSA signature generation is 810 ms at 8 MHz. Work by Uhsadel and Scott [22, 26] further improves speed results on the ATmega128.

Malan refers in [20] to an ECC implementation based on binary fields on the ATmega128 microcontroller. It was implemented using a C++ and Java library. The running times were quite slow, a complete point multiplication was executed in 34 seconds. Shi and Yan showed in [24], that a sophisticated implementation of the inversion algorithm enables an improvement for binary curves. Their run times for a scalar multiplication was 12.2 seconds. In 2005 Blaß and Zitterbart implemented finite fields of characteristic two on the ATmega128 [6] clocked at

* This work is being carried out in the context of the SMEPP project (Secure Middleware for Embedded Peer-to-Peer Systems, FP6 IST-5-033563).

7MHz, too. The elements of the finite field of degree 113 were represented by normal bases and the multiplication of a fixed base point took about 6.7 seconds. In 2008 Szczechowiak et al. and Seo et al. reported new speed records for elliptic curve realizations concerning binary fields [25, 23]. While Szczechowiak consumes 2.16 seconds for a scalar multiplication on a binary Koblitz curve, Shi and Yan applied the Frobenius endomorphism and they were able to decrease the run time to 1.13 seconds.

In this paper we will study the efficiency of optimal extension fields, prime fields as well as binary fields for the ATmega128. We will show, that the run times for a multiplication in $GF(2^{167})$ takes less than 1 ms, if the memory accesses are decreased as much as possible. Furthermore we will take side channel attacks into account and measure the run times of elliptic curve scalar multiplication resistant to simple side channel attacks for several finite fields of different characteristic. It turns out, that binary fields are the optimal choice for elliptic curve cryptography if the Montgomery-Ladder can be used.

2 The Target Platform — ATmega128 Microcontroller

Depending on the requirements for computing power and peripheral interfaces, different types of microcontrollers are embedded into sensor nodes. The used microcontroller architectures range from low-cost 8-bit to advanced 32-bit platforms. For our work we employed the ATmega128 [3], a high-performance low-power 8-bit microcontroller provided by the Atmel corporation. The ATmega128 is a derivative of the AVR family which is based on a modern highly structured RISC design. The different AVR microcontrollers are equipped with various peripheral units and memory sizes. They are widely available and low-priced. Beside the MSP430 from Texas Instruments and ARM implementations from different manufacturers, AVR microcontrollers are very popular in sensor network environments.

The ATmega128 offers 133 powerful instructions, where most of them are single clock cycle instructions. This allows nearly a maximum throughput of 16 MIPS at 16 MHz. There are two different memory spaces for data and code, so AVR controllers implement a Harvard architecture. The AVR pipeline is very simple and has got only two stages, one for fetching the instruction and another one for the execution. One of the most interesting features is the large register set of 32×8-bit general purpose working registers. The physical memory is split up in three parts. The program memory is a 128 kB in system programmable flash memory. The data memory is a 4 kB internal SRAM which is expandable with external memory up to 64 kB. Additionally there is a internal EEPROM of 4 kB which isn't directly addressable but offers an interface using special function registers.

There are many development tools available for the Atmel AVR family. Atmel offers the free *AVR Studio* development environment including an assembler. Furthermore there are many third party suppliers. For an up to date list see [2].

3 Efficient Finite Fields for ATMega128

There are three categories of finite fields, which have been proposed in the literature suitable for elliptic curve cryptosystems. We will investigate their advantages in terms of speed for our target platform.

3.1 Prime fields

Finite fields $\mathbb{F}_p$ of large characteristic $p \sim 2^{160}$ are usually used in software solutions. Efficient long integer libraries like GMP [14] or MIRACL [1] enable very fast implementations on common PC platforms. In addition, modern smart cards are equipped with coprocessors to accelerate modulo arithmetic with respect to long moduli. Recently, Gneysu et al. reported in [15], that prime fields can be implemented on modern FPGA-boards very efficiently, such that the performance of FPGA- and PC-realizations are quite similar.

In previous articles Scott, Uhsadel and Gura [16, 22, 26] describe implementation results of prime field arithmetic. All authors implemented the modular multiplication for the prime number $p_1 = 2^{160} - 2^{31} - 1$ with dedicated reduction. The modular reduction is easy to achieve in this finite field, because it consists of two shifts and four additions. Generic reduction methods like Barret- or Montgomery reduction are not necessary.

Since we want to compare the security levels with our other implementations, we have implemented the arithmetic for two finite fields $\mathbb{F}_{p_1}$ and $\mathbb{F}_{p_2}$, where $p_2 = 2^{168} - 2^8 - 1$. Both libraries are based on the recommendations of Scott in [22]. Caused by the additional byte of the elements on $\mathbb{F}_{p_2}$ we have modified the multiplication in $\mathbb{F}_{p_2}$ in a intuitive manner (see Figure 1). Reducing elements in $\mathbb{F}_{p_2}$ is rather simple, because shifting operations can be discarded.

The running times of the two libraries are quite similar. The modular multiplication in $\mathbb{F}_{p_1}$ took 3177 clock cycles on average. The multiprecision multiplication 160×160 was performed in 2593 clock cycles and is about 60 cycles faster than [22] (see Table 1). The difference might be caused by some saved memory accesses, but there is no fundamental new approach. So we skip the investigation of that acceleration. In $\mathbb{F}_{p_2}$ the multiplication counted 3338 clock cycles. The ratio of the two routines of 95% coincides with the ratio of the binary lengths of the prime numbers. In both libraries dedicated squaring routines lead to a speed up of the squaring by at least 26%.

Table 1: Comparison of 160×160 bit multiprecision multiplication

Source	# Clock cycles
Gura et al.[16]	3103
Uhsadel et al. [26]	2881
Scott et al. [22, 25]	2651
this work	2593

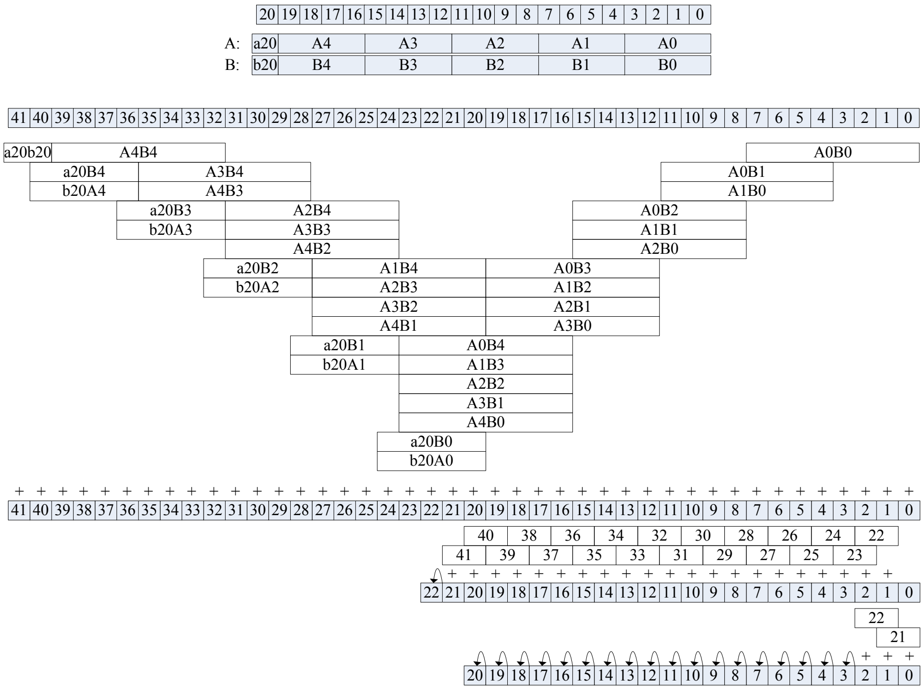

Fig. 1: Multiplication in $GF(2^{168} - 2^8 - 1)$.

Note, that our numbers for modular multiplication involve also the clock cycles for saving and recovering the registers prior to the function call or prior to the return jump. These necessary cycles are discarded in the previous papers [16, 22, 26].

3.2 Finite Field Extensions of Odd Characteristic Greater 3.

In [4] Bailey and Paar propose finite extension fields of odd characteristic p, where p is adapted to the word size of the target processor. Similar to finite fields of characteristic 2 the elements of the finite field are represented by polynomials with coefficients in $\mathbb{F}_p$. Bailey and Paar define so called optimal extension fields which have the following two properties:

1. The characteristic p of the extension field can be written as $p = 2^n - c$, where $c^2 < n$.
2. The field extension is defined by an irreducible binomial, i.e.

$$\mathbb{F}_{p^d} \cong \mathbb{F}_p[X]/(X^d - \omega), \quad \omega \in \mathbb{F}_p.$$

Since we want to investigate the efficiency of elliptic curves over those extension fields, we restrict the extension degree d to be a prime number greater than

7. Diem shows in [11] that those finite fields prevent the attack based on the Weil descent presented by Gaudry et al. in [13] for finite fields of characteristic 2.

It is easy to see, that OEFs cannot be defined for every arbitrary prime p. Bailey and Paar mention, that the extension degree d has to divide the order e of ω in $\mathbb{F}_p$ but it must not divide the number $(p-1)/e$. This sufficient condition restricts the number of possibilities.

The arithmetic in OEF is derived from the usual arithmetic in polynomial rings. The addition of two polynomials is done by the addition of their coefficients modulo p and the multiplication is computed by a complete multiplication of two polynomials and the subsequent reduction modulo the irreducible polynomial.

Depending on the constant term ω it might be advantageous to choose a trinomial of the form $X^d \pm X^m \pm 1$, because all its coefficients have the absolute value 1 and hence the reduction consists of additions in $\mathbb{F}_p$ only. In particular, if the smallest possible value for ω involves a multiplication and if the multiplication on the target microcontroller is much more expensive than an addition, the trinomial might reduce the computation time compared to the binomial.

Our implementation is based on the finite field $\mathbb{F}_{p^{11}}$, where $p = 2^{16} - 129$ and the extension is defined by $X^{11} - 3$. Although our target platform has an 8-bit architecture the 16 bit construction is much more efficient than choosing an 8-bit prime number.

Choosing an 8-bit prime number results in an extension degree $d = 23$ in order to obtain an elliptic curve which provides at least 160 bit security. There is only one possibility $p = 139$ with irreducible polynomial $X^{23} - 2$. In the multiplication of polynomials of degree 23 we have to reveal 23 partial products for every coefficient of the product. Those partial products are summed up in an accumulator of 24 bits and therefore we need 3 additions for every partial product. If we assume, that we have to load the two factors into the CPU we can conclude that the computation of one coefficient of the product consumes $23 \cdot (2 + 3 + 4)$ clock cycles (multiplication + additions + loading) without reduction modulo p and $X^{23} - \omega$. So a lower limit for the clock cycles of one multiplication is $23 \cdot 207 = 4761$.

If p is a 16-bit prime number, then we have exactly 11 multiplications of 16-bit numbers for every coefficient. The multiplication of two 16-bit numbers results in a 32-bit value which has to be added to a 40 bit accumulator. The two factors $a = a_1 2^8 + a_0$ and $b = b_1 2^8 + b_0$ are multiplied in the following way (see Figure 2): First the partial products $a_1 b_1$ and $a_0 b_0$ are computed and added to the accumulator. In order to avoid checking the carry bits for the whole accumulator the carry bit occurring during the addition of the higher byte of $a_0 b_0$ is stored in an extra register. Then the products $a_1 b_0$ and $a_0 b_1$ are computed and added to the accumulator, such that the carry bit of the higher byte is catched again in a further register. All in all the multiplication of two 16-bit numbers can be realized within 20 cycles. Including the 8 cycles loading the factors into the CPU the multiplication of the two polynomials of degree 11 consume 3390 clock cycles without any reduction. This is an acceleration of almost 30% and justifies the choice of 16-bit prime numbers.

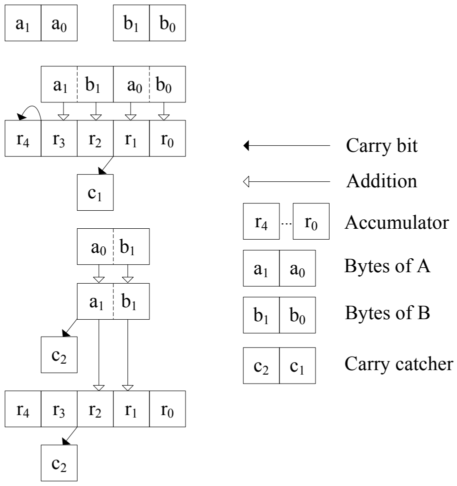

Fig. 2: 16×16-bit multiplication with 40 bit accumulator and carry catchers.

The reduction of the higher coefficients c_i, $i \in \{11, \dots, 20\}$, is performed within the calculation of the coefficients c_{i-11}. Reducing the coefficient c_i results in an addition of the carry bits (4 clock cycles), copying, shifting and adding the accumulator (15 further cycles). All in all the reduction of a higher coefficient, i.e. the multiplication by 3, is done in 19 cycles. Since the shift instruction is only twice as fast as the 8×8 multiplication on the ATmega128, the reduction of the accumulator modulo $2^{16} - 129$ is built on 7 multiplications. The total count of clock cycles for one field multiplication in $\mathbb{F}_{p^d}$ is 4188. Furthermore we implemented a particular squaring function in order to avoid the double computation of the mixed terms. The acceleration is about 46%. In Table 2 we give a detailed analysis of the instruction counts computing the product of two OEF-elements. Note, that clock cycles for storing or recovering of registers are neglected in the table.

3.3 Finite Fields of Characteristic Two.

Finite extension fields of characteristic two are very attractive for hardware accelerated elliptic curve operations, since their field arithmetic consists of binary operations only.

Usually finite fields of even characteristic are represented by polynomials over $\mathbb{F}_2$ of degree less than d, where d is a prime number and the degree of the field extension (see [17]). The arithmetic in those fields is performed modulo

Table 2: Instruction counts for OEF-multiplication.

Instruction	Counts
Load/Store	506
Mul	561
Add	1686
Shift	61
other	181
total	2995
Clock cycles	4062

an irreducible polynomial $f(X)$. To decrease the computation time spent in the reduction of polynomials, $f(X)$ should be sparse, i.e. a trinomial or pentanomial.

Binary polynomials are represented with a binary vector of length d, which can be stored in $\lceil d/w \rceil$ processor words. Hankerson et al. propose several techniques implementing the modular multiplication of $A(X)$, $B(X) \in \mathbb{F}_{2^d}$ in software [17]. The basic multiplication method — called *Shift-and-Add* — is derived from the well-known Horner scheme and performs $\deg(A) - 1$ left shifts and an addition of $B(X)$ for every non-zero coefficient of A. If the left shift causes a polynomial of degree greater than d, a reduction is executed. Hankerson presents several variants of the Shift-and-Add algorithm. The most efficient one is the comb multiplication with windows of width four.

To explain this method and its implementation on the ATmega128, we write $A(X)$ in the following way

$$A(X) = \sum_{i=0}^{d-1} a_i X^i = \sum_{i=0}^{\lceil (d-1)/4 \rceil} \underbrace{\left(\sum_{j=0}^{3} a_{4i+j} X^j \right)}_{\alpha_i} X^{4i}.$$

The basic algorithm described by Hankerson et al. [18] works as follows:

Algorithm 1: Comb method with windows of width 4.

Input: $A(X), B(X) \in \mathbb{F}_{2^d}$
Output: $C(X) = A(X)B(X)$

1. Compute $T[u] = u(X)B(X)$ for all $u \in \mathbb{F}_2[X]$, $\deg(u) < 4$
2. $C = 0$
3. For $i = \lceil (d-1)/8 \rceil$ to 0 do
   $C(X) = C(X) + T[\alpha_{2i+1}]X^{8i}$
4. $C(X) = C(X)X^4$
5. For $i = \lceil (d-1)/8 \rceil$ to 0 do
   $C(X) = C(X) + T[\alpha_{2i}]X^{8i}$
6. return $C(X)$

Seo, Han and Hong try to optimize the number of memory accesses to the variable C by combining several windows α_i [23]. They explain their technique for two windows α_i and α_{i+1} and show that a lot of redundant memory accesses can be discarded. But their implementation is not written in assembly language and therefore, the CPU registers are not used in an optimal way. In particular, the overhead generated by the C-compiler affects the runtime negatively. So we pursue the strategy, processing the windows α_i sequentially but using all registers of the processor. Of course this strategy results in a bigger code size, but we will see, that the code size is still acceptable.

In our approach the comb multiplication with windows of width four is also executed in two loops following the algorithm given above. Since the ATmega128 has an 8-bit architecture, the multiplication by X^{8i} is replaced by an address computation, but the most important problem is the handling of the polynomial $C(X)$. Our implementation is based on the observation, that nearly all registers, which are affected by the addition of the multiple $\alpha_i B(X)$ are altered again during the addition of $\alpha_{i-2}B(X)$. Only one single register after the addition of $\alpha_i B(X)$ keeps untouched. On the other hand, adding $\alpha_{i-2}B(X)$ changes a new register of the variable C, so we have to load this part of the memory into the CPU. In order to avoid unnecessary copy instructions, these two bytes are exchanged. This results in a total number of 42 memory accesses processing C inside both loops (see Figure 3).

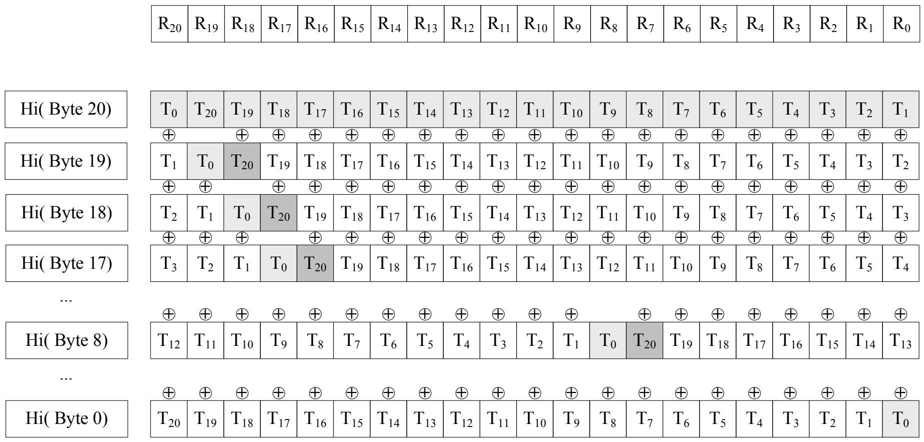

Fig. 3: First part of GF(2^{167}) comb multiplication with windows of width 4. T0 to T20 denote the 21 bytes of the multiple of $B(X)$ determined by the nibble "Hi(Byte XX)". The bytes without filling are added to the registers. Bytes filled with light-gray are loaded into the registers. The dark-gray bytes are stored in the RAM after the addition (except T20 of Hi(Byte 20)).

In the following table we listed the instructions for the comb method with windows of width four. The precomputation phase, the two loops as well as the multiplication of the polynomial by X^4 is separated to provide more details on the implementation. Instructions calculating the addresses or clearing registers are summed up in a separate column. It turns out, that the number of memory accesses is lower than the reported number of load and store instructions given by Seo, Han and Hong [23].

Table 3: Instruction counts for comb method with window width 4.

Phase	Load/Store	XOR	Shifts	other	Total
Precomputation	481	157	146	185	969
Loop 1	501	400	0	169	1070
Multiplication by X^4	43	0	176	25	244
Loop 2	544	441	0	188	1173
Total	1569	998	322	567	3456
Clock Cycles	3138	998	322	599	5057

The reduction of the polynomial $C(X)$ is then performed byte-wise. Hankerson presents in [17] an efficient reduction method for sparse reduction polynomials. We adapt this methodology to our reduction polynomial $X^{167} + X^6 + 1$. We reduce one complete register by the congruence $X^{168} \equiv X^7 + X$, such that we have to perform two left shift, two right shift and four XOR instructions. The implementation of the multiplication in assembly language as well as the faster reduction caused by the different reduction polynomial lead to an efficient multiplication in $\mathrm{GF}(2^{167})$ (see Table 4). While our implementation computes the product of two finite field elements in 0.67 ms, Seo et al. consume about 2.9 ms. This is an improvement by a factor 4.

Additionally, we composed a particular function to square a polynomial in $\mathbb{F}_{2^d}$, because squaring in characteristic 2 is just a spreading of the coefficients [5]:

$$A(X)^2 = \left(\sum_{i=0}^{d-1} a_i X^i \right)^2 = \sum_{i=0}^{d-1} a_i X^{2i} \mod f.$$

As recommended in [17], the spreading is achieved by a table look-up. In order to avoid a result $C(X) = A(X)^2$ of double length, we composed a second look-up table storing the coefficients of the polynomials $u(X)^2(X^7 + X)$ in two bytes per entry. The polynomials are used to include the reduction into the spreading of the upper half.

Finally, we want to illustrate the advantage of the comb method with windows of width 4 and so we have implemented the binary Shift-and-Add as well as two window methods on the ATmega128. We have chosen the finite field $\mathbb{F}_2[X]/(f(X))$, where $f(X) = X^{167} + X^6 + 1 \in \mathbb{F}_2[X]$. In Table 4 we give the clock cycles and timings of all implemented multiplication routines in $\mathbb{F}_2[X]/(X^{167} + X^6 + 1)$.

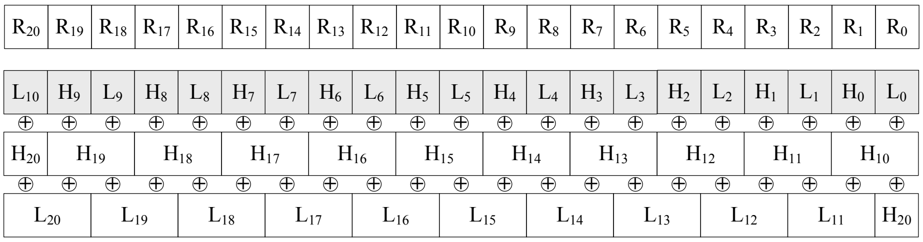

Fig. 4: Squaring in GF(2^{167}) including reduction. The light-gray bytes are copied into the registers of the microcontroller. The reduced bytes $A(X)^2(X^7+1)$ ($H_{10} \dots H_{20}$ and $L_{11} \dots L_{20}$) are xored with the registers.

Table 4: Clock cycles of multiplication functions in $\mathbb{F}_2[X]/(X^{167} + X^6 + 1)$

Function	# Clock cycles	Time in ms
Double-and-Add	10852	1.36
Double-and-Add (Window width 4)	8789	1.10
Comb method (Window width 4)	5490	0.69
Squaring	663	0.08

4 Application to Elliptic Curves

In this section we apply our efficient arithmetic to elliptic curve cryptography. In the following the notation $\mathbb{F}_q$ will denote a finite field of characteristic $p > 3$ with q elements. The parameter q has not to be a prime number but a prime power $q = p^d$. If q is a prime power then $\mathbb{F}_q$ is a field extension over the prime field $\mathbb{F}_p$ of degree d. If the finite field has characteristic 2, then it will be explicitly denoted by $\mathbb{F}_{2^d}$.

Let E be an elliptic curve over an arbitrary finite field, then the elliptic curve can be represented by an shortened Weierstraß-equation

$$E : y^2 = x^3 + ax + b, \quad \text{with } a, b \in \mathbb{F}_q \text{ and } 4a^3 + 27b^2 \neq 0.$$

$$E : y^2 + xy = x^3 + ax^2 + b, \quad \text{with } a, b \in \mathbb{F}_{2^d} \text{ and } b \neq 0.$$

The set of solutions of the polynomials together with the point at infinity form an additive group. The addition law of the elliptic curve is built on operations in the finite field $\mathbb{F}_q$ or $\mathbb{F}_{2^d}$ resp. Therefore fast arithmetic in the finite fields is necessary for efficient elliptic curve cryptosystems. In [9] the authors report, that field inversions reduce the efficiency of elliptic curve based cryptosystems drastically, because the ratio of a field inversion to the field multiplication is rather high. For all finite fields we were able to confirm this statement, such that we avoid field inversions by switching to projective coordinates.

The transformation to the projective space is not unique. Several embeddings are proposed in the literature [10]. Cohen et al. point out, that for fields of odd characteristic the Jacobian representation

$$\mathbb{P}^2(\mathbb{F}_q) \to \mathbb{A}^2(\mathbb{F}_q)$$
$$[X:Y:Z] \mapsto \left(\frac{X}{Z^2}, \frac{Y}{Z^3}\right) \quad \text{for} \quad Z \neq 0.$$

of coordinates offers the fastest implementations. In characteristic 2 coordinates introduced by Lopez and Dahab are the optimal choice [19].

$$\mathbb{P}^2(\mathbb{F}_{2^d}) \to \mathbb{A}^2(\mathbb{F}_{2^d})$$
$$[X:Y:Z] \mapsto \left(\frac{X}{Z}, \frac{Y}{Z^2}\right) \quad \text{for} \quad Z \neq 0.$$

We recall addition laws for these projective representations in order to select our algorithms for scalar multiplication.

Table 5: Point addition and point doubling for Jacobian coordinates in $\mathbb{F}_q$

$P \neq Q$	$P = Q$
$X_3 = -CE - 2AE + D^2$	$X_3 = C$
$Y_3 = -BCE + D(AE - X_3)$	$Y_3 = -8Y_1^4 + B(A - C)$
$Z_3 = Z_1Z_2C$	$Z_3 = 2Y_1Z_1$
$A = X_1Z_2^2, B = Y_1Z_2^3,$	$A = 4X_1Y_1^2, B = 3X_1^2 + aZ_1^4,$
$C = X_2Z_1^2 - A, D = Y_2Z_1^3 - B, E = C^2$	$C = -2A + B^2$

The computational effort for adding two different points in Jacobian coordinates consists of 12 multiplications and 4 squarings. For the doubling of P we have to perform 4 multiplications and 6 squarings. Representing points in Jacobian coordinates is more efficient than affine coordinates if the multiplication is 12 times faster than an inversion.

Table 6: Point addition and point doubling for Lopez-Dahab coordinates in $\mathbb{F}_{2^d}$

$P \neq Q$	$P = Q$
$X_3 = A(H + D) + B(C + G)$	$X_3 = C^2 + B$
$Y_3 = (AJ + FG)F + (J + Z_3)X_3$	$Y_3 = (Y_1^2 + aZ_3 + B)X_3 + Z_3B$
$Z_3 = FZ_1Z_2$	$Z_3 = AC$
$A = X_1Z_2, B = X_2Z_1, C = A^2, D = B^2,$	$A = Z_1^2, B = bA^2, C = X_1^2$
$E = A + B, F = C + D, G = Y_1Z_2^2$	
$H = Y_2Z_1^2, I = G + H, J = IE$	

Using elliptic curves over binary fields representing points in Lopez-Dahab coordinates, we have to execute 3 multiplications and 4 squarings for point addition, whereas we only need 5 multiplications and 4 squarings in the case of point doubling.

The core of elliptic curve cryptosystems is the scalar multiplication, i.e. the iterated addition of a point to itself. It is well-known that cryptosystems resistant against theoretical attacks might be vulnerable to side-channel attacks. These attacks exploit information like power consumption, electromagnetic radiation, run time oscillation or handling of calculation faults. In particular, simple power analysis is a very dangerous and efficient threat. It assigns parts of the power consumption profile to operations on the elliptic curve, i.e. the power consumption profile is divided into additions and doublings on the curve. Therefore, it is indispensable to adjust the power consumption profile by dummy operations or to use scalar multiplication algorithms which process the key bits in a constant manner.

There are several proposals for scalar multiplications preventing simple side channel attacks. The most efficient algorithm is based on a window technique [18], but since we are interested in memory saving methods we restrict to algorithms, which do not need precomputed points. So, there are two remaining algorithms, which can be used in secure implementations of elliptic curve cryptosystems.

The Double-And-Add-Always (see Algorithm 2) is resistant against simple side chanel attacks, because an addition and a doubling is executed independently from the certain value of the key bit. Using mixed coordinates the algorithm consumes $12n$ multiplications and $9n$ squarings for $\mathbb{F}_q$ [10] ($12n$ multiplications and $10n$ squarings for $\mathbb{F}_{2^d}$ [18]). Additions in the finite fields are neglected, because they play not an important role for the total running time.

Algorithm 2: **Double-and-add-always**	**Algorithm 3:** **Montgomery ladder**
Input: $P \in E$, $k \in \mathbb{Z}$	*Input:* X_1
Output: $Q = kP$	*Output:* $A/B = X_k/Z_k$, $C/D = X_{k+1}/Z_{k+1}$
$Q[0] \leftarrow P$;	$A \leftarrow 1$; $B \leftarrow 0$; $C \leftarrow X_1$; $D \leftarrow 1$;
$j \leftarrow 0$;	$i \leftarrow n-1$;
While $j < n$	While $i > -1$;
$Q[0] \leftarrow P$;	If $k_i = 1$
$Q[1] \leftarrow Q[0] + P$;	add(A, B, A, B, C, D);
$Q \leftarrow Q[k_i]$;	double(C, D, C, D);
$P \leftarrow 2P$;	Else
$j \leftarrow j + 1$;	add(C, D, C, D, A, B);
Return Q;	double(A, B, A, B);
	$i \leftarrow i - 1$;
	Return;

An alternative to the Double-And-Add-Always is the so called Montgomery-Ladder (see Algorithm 3). It was published by Peter Montgomery [21] for fast factorization based on elliptic curves. It can be applied to elliptic curves defined over arbitrary finite fields and the concrete formulas are summarized in [12] and [18]. Montgomery determines the projective x-coordinates (X_3, Z_3), (X_4, Z_4) of points $P_3 = P_1 + P_2$, $P_4 = 2P_2$, such that the projective x-coordinates (X_1, Z_1) and (X_2, Z_2) of P_1, P_2 and the difference $P_2 - P_1$ are used only. Starting with the input points $P_1 = \mathcal{O}$ and $P_2 = P$, we can derive the multiple kP such that the same operations are executed for different values of the key bits. In $\mathbb{F}_{2^d}$ we need $6n$ multiplications and $4n$ squarings. For arbitrary elliptic curves defined over finite fields of characteristic $p > 3$ we have $5n$ squarings and $14n$ multiplications. Since in $\mathbb{F}_q$ squaring is not twice as fast as the multiplication, the Montgomery-Ladder is faster than the Double-And-Add-Always for every choice of the finite field.

In Table 7 we give the running times of the Montgomery-ladder on the ATmega128 for all finite fields implemented in the previous section. The used curves were cryptographically strong, such that the security levels of the curves are similar.

Table 7: Performance data of Montgomery-ladder without scalar randomization and without reconstruction of the y-coordinate on ATmega128@8MHz

Finite field	length of scalar	# Clock cycles	Running time	Code size
$\mathbb{F}_{p_1}$	160	10086676	1.261 s	7.7 kB
$\mathbb{F}_{p_2}$	168	10534273	1.317 s	9.6 kB
$\mathbb{F}_{p^{11}}$, $p = 2^{16} - 129$	176	13830892	1.729 s	9.8 kB
$\mathbb{F}_{2^{167}}$	165	6100462	0.763 s	11 kB

Table 7 shows, that the efficiency of the finite field is not the single argument for fast ECC implementations. In particular, for finite fields of characteristic two the short formulas of the Montgomery-Ladder enable scalar multiplications secured against simple side channel attacks, which are comparable to other finite fields. Although the parameters extension field $\mathbb{F}_{p^d}$ were adapted to the properties of the target platform large prime fields are not less efficient.

5 Conclusion

In this paper we investigated the efficiency of different finite fields on the ATmega128 microcontroller. We used prime fields, optimal extension fields and binary fields. Prime fields offer the fastest arithmetic, but involving the arithmetic into quick scalar multiplication algorithms secured against simple power analysis, binary curves outperform all other types of finite fields.

References

1. MIRACL, Multiprecision Integer and Rational Arithmetic C/C++ Library. `http://www.shamus.ie/`.
2. Atmel Corporation. AVR 8-Bit RISC Third Party. available at `http://www.atmel.com/products/AVR/thirdparty.asp`.
3. Atmel Corporation. Datasheet: ATmega128(L). available at `http://www.atmel.com/products/avr/`.
4. D.V. Bailey and C. Paar. Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO 98*, volume 1462 of *Lecture Notes in Computer Science*, pages 472–485. Springer-Verlag, 2000.
5. I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1999.
6. E.-O. Blaß and M. Zitterbart. Efficient Implementation of Elliptic Curve Cryptography for Wireless Sensor Networks. Technical Report TM-2005-1, Insitute of Telematics, University of Karlsruhe, 3 2005.
7. M. Braun, E. Hess, and B. Meyer. Using Elliptic Curves on RFID Tags. *IJCSNS International Journal of Computer Science and Network Security*, 8(2):1–9, 2008.
8. D. Brown. *SEC 1: Elliptic Curve Cryptography.* Certicom Research, February 2005. Working Draft Version 1.5.
9. M. Brown, D. Hankerson, J. Lopez, and A. Menezes. Software Implementation of the NIST Elliptic Curves over Prime Fields. Technical Report CORR 2000-58, Centre for Applied Cryptography, University of Waterloo, 2000.
10. H. Cohen, A. Miyaji, and T. Ono. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In K. Ohta and D. Pei, editors, *Advances in Cryptology - ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 51–65. Springer-Verlag, 1998.
11. C. Diem. The GHS Attack in Odd Characteristic. *J. Ramanujan. Math. Soc.*, 18(1):1–32, 2003.
12. W. Fischer, C. Giraud, E.W. Knudsen, and J.P. Seifert. Parallel scalar multiplication on general elliptic curves over F_p hedged against Non-Differential Side-Channel Attacks. Cryptology ePrint Archive, 2002/007.
13. P. Gaudry, F. Hess, and N.P. Smart. Constructive and Destructive Facets of Weil Descent on Elliptic Curves. Technical Report CSTR-00-016, Department of Computer Science, University of Bristol, 2000.
14. T. Granlund. *The GNU Multiple Precision Arithmetic Library*, September 2004. Available at `http://www.swox.com/gmp/`.
15. T. Güneysu and C. Paar. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 62–78. Springer-Verlag, 2008.
16. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In M. Joye and J. J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 119–132. Springer-Verlag, 2004.
17. D. Hankerson, J.L. Hernandez, and A. Menezes. Software implementation of elliptic curve cryptography over binary fields. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 200.

18. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer-Verlag, 2004.
19. J. López and R. Dahab. Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$. In S.E. Tavares and H. Meijer, editors, *Selected Areas in Cryptography — SAC '98*, volume 1556 of *Lecture Notes in Computer Science*, pages 201–212. Springer-Verlag, 1998.
20. D.J. Malan, M. Welsh, and M.D. Smith. A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography. In *Sensor and Ad Hoc Communications and Networks, 2004*, pages 71–80, 2004.
21. P.L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Math. Comp.*, 48:243–264, 1987.
22. M. Scott and P. Szczechowiak. Optimizing Multiprecision Multiplication for Public Key Cryptography. Cryptology ePrint Archive, Report 2007/299, 2007. `http://eprint.iacr.org/`.
23. S.C. Seo, D. G. Hand, and S. Hong. TinyECCK: Efficient Elliptic Curve Cryptography Implementation over $GF(2^m)$ on 8-bit MICAz Mote. Cryptology ePrint Archive, Report 2008/XXX, 2008. `http://eprint.iacr.org/`.
24. Z.J. Shi and H. Yan. Software Implementations of Elliptic Curve Cryptography. *International Journal of Network Security*, 7(2):157–166, 2008.
25. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. Nanoecc: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In R. Verdone, editor, *Wireless Sensor Networks — EWSN 2008*, volume 4913 of *Lecture Notes in Computer Science*, pages 305–320. Springer-Verlag, 2008.
26. L. Uhsadel, A. Poschmann, and C. Paar. Enabling Full-Size Public-Key Algorithms on 8-Bit Sensor Nodes. In F. Stajano, C. Meadows, S. Capkun, and T. Moore, editors, *Security and Privacy in Ad-hoc and Sensor Networks*, volume 4572 of *Lecture Notes in Computer Science*, pages 73–86. Springer-Verlag, 2007.