Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations

Erich Wenger and Michael Hutter

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{Erich.Wenger,Michael.Hutter}@iaik.tugraz.at

Abstract. In this paper, we answer the question whether binary extension field or prime-field based processors doing multi-precision arithmetic are better in the terms of area, speed, power, and energy. This is done by implementing and optimizing two distinct custom-made 16-bit processor designs and comparing our solutions on different abstraction levels: finite-field arithmetic, elliptic-curve operations, and on protocol level by implementing the Elliptic Curve Digital Signature Algorithm (ECDSA). On the one hand, our $\mathbb{F}_{2^m}$ based processor outperforms the $\mathbb{F}_p$ based processor by 19.7 % in area, 69.6 % in runtime, 15.9 % in power, and 74.4 % in energy when performing a point multiplication. On the other hand, our $\mathbb{F}_p$ based processor (11.6 kGE, 41.4 μW, 1,313 kCycles, and 54.3 μJ) improves the state-of-the-art in $\mathbb{F}_{p_{192}}$ ECC hardware implementations regarding area, power, and energy results. After extending the designs for ECDSA (signature generation and verification), the area and power-consumption advantages of the $\mathbb{F}_{2^m}$ based processor vanish, but it still is 1.5-2.8 times better in terms of energy and runtime.

Keywords: Hardware Implementation, Elliptic Curve Cryptography, ECC, ECDSA, Binary-Extension Field, Prime Field.

1 Introduction

Elliptic Curve Cryptography (ECC) has been introduced in the 1980s and is used nowadays in a variety of different applications. Every application has its own design criteria and raises special requirements for hardware designs. While contact-less powered devices have to meet low-power constraints, battery-powered devices need energy-aware implementations that consume as little energy as possible to increase the life-time of the battery.

The most fundamental decision concerning future hardware designs is whether to use a binary-extension field or a prime field as basis of the used elliptic curve. Most related work in dedicated hardware designs has been done in implementing ECC over binary fields using full-precision arithmetic. Only a few papers compared binary and prime fields in hardware. Wolkerstorfer [36] and Satoh [32] used full-precision dual-field hardware with bit-serial multipliers. We however

P. Laud (Ed.): NordSec 2011, LNCS 7161, pp. 256–271, 2012.

are interested in multi-precision designs, where the big integers are split and processed in small words. This design methodology has the advantage that the Central Processing Unit (CPU) can be reused to perform other work (*e.g.* protocol handling). In this paper we want to answer the following questions:

- What are the advantages and disadvantages of prime and binary-field processors in custom multi-precision hardware?
- How big are the differences when identical design methodologies and elliptic curves with similar security level are used?
- How does the performance of prime and binary-field processors scale in higher-level protocols?
- Does the speed advantage of carry-less operations makes up the additional need of prime-field arithmetics?

In this paper, we answer these questions by presenting two distinct custom 16-bit processors that leverage binary-field operations and prime-field operations and are based on [35] and [34]. Using a metric consisting of **area**, **speed**, **power**, and **energy**, we not only compare both designs in terms of finite-field operation and ECC point-multiplication performance, we also investigate a higher-level protocol. When performing an ECC-point-multiplication, the $\mathbb{F}_{2^m}$ based processor (9.3 kGE, 34.8 μW, 400 kCycles, and 13.9 μJ) is 3.3 times faster, 20 % smaller, uses 16 % less power, and needs 3.9 times less energy compared to the $\mathbb{F}_p$ based processor (11.6 kGE, 41.4 μW, 1,313 kCycles, and 54.3 μJ). Nevertheless our $\mathbb{F}_p$ based processor improves the state-of-the-art in area, power, and energy results for prime-field based ECC (doing point multiplication).

We further present two full hardware implementations of the Elliptic Curve Digital Signature Algorithm (ECDSA). It shows that the $\mathbb{F}_{2^m}$ based processor does not outperform the $\mathbb{F}_p$ based processor in every category of the metric. The $\mathbb{F}_{2^m}$ based processor is 4.4-5.5 % larger, needs up to 6.3 % more power, but still is 2.8 times faster and needs 2.8 times less energy when calculating a signature. The runtime and energy advantage drops down to a factor of 1.5 when the verification is done.

The paper is organized as follows. Section 2 discusses related work on ECC implementations. Section 3 gives an introduction to elliptic curve cryptography and introduces a metric. Whereas Section 4 gives a comparison, Section 5 thoroughly discusses all implementation results. Conclusions are given in Section 6.

2 Related Work on ECC-Hardware Implementations

There exist many hardware implementations of elliptic-curve cryptography. In the following, we consider only lightweight implementations that address embedded systems, wireless sensors, and contactless-powered applications. Most of the given implementations are based on either binary field, prime field, or dual-field arithmetic. A very tiny ECC processor over binary fields has been proposed by Y. K. Lee et al. [25] in 2008. They based their design on a compact architecture of a Modular Arithmetic Logic Unit (MALU) that has been first presented by

the work of L. Batina et al. [4] in 2006. The processor performs (full-precision) operations in $\mathbb{F}_{2^{163}}$ and calculates a scalar multiplication between about 80 000 and 300 000 clock cycles (depending on the digit size of the hardware multiplier). The final architecture needs about 12-20 kGEs of area. Similar results have been also reported by S. Kumar and C. Paar [24] who presented a generic binary-field processor over $\mathbb{F}_{2^{113-193}}$. The run-time and area requirements of the proposed processor is similar, needing between 170 000 and 560 000 clock cycles and 10-19 kGEs. D. Hein et al. [15] reported a low-resource co-processor for passive Radio Frequency Identification (RFID) applications. In contrast to the previous work, they applied multi-precision arithmetic over $\mathbb{F}_{2^{163}}$. Their ECC design needs about 300 000 clock cycles for one scalar multiplication and consumes about 11 kGEs of chip area. In view of power consumption, all described designs need between 8 and 30 μWs of power at 100 kHz and are thus well applicable to the targeted applications.

Prime-field based processors have been reported by, for example, E. Öztürk et al. [31] in 2004. They presented an ECC architecture over the prime field $\mathbb{F}_{2^{(167+1)}/3}$. Their design needs 545 440 clock cycles for one scalar multiplication and requires about 30 kGEs of area. Similar results have also been reported by F. Fürbass and J. Wolkerstorfer [11] in 2007. Their $\mathbb{F}_{p_{192}}$ processor needs 502 000 clock cycles and about 23 kGEs of area. Recently, M. Hutter et al. [16] presented an ECC processor over the same prime field needing about 750 000 clock cycles for a scalar multiplication and about 19 kGEs of area. E. Wenger et al. [34] reduced the area requirements even further to only about 12 kGEs but their design needs about 1.4 million clock cycles. The power consumption of most of the reported prime-field processors is about 20 to several hundred μWs of power at 100 kHz.

By the given related work, it seems that binary-field processors benefit from a more efficient computation for application-specific hardware implementations. However, it is impossible to make a fair comparison since the authors used different design techniques, synthesis tools, bit/word sizes, and EC parameters. This renders a comparison largely unfeasible. Nevertheless, there exist only a few publications that reported dual-field processors for ECC that give detailed comparison results. A. Satoh and K. Takano [32] presented a processor over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ supporting 160 to 256 bits. They show that the binary-field operations can be performed about six times faster than their prime-field opponents (1.21 ms vs. 0.19 ms for a 160-bit scalar multiplication). Furthermore, the area requirements for the prime-field controller is 1.47 times larger than the binary-field controller (6 606 GEs vs. 4 490 GEs for 8-bit word size and a 160-bit scalar). J. Wolkerstorfer [36] also presented a dual-field processor that supports 190 to 256 bits. One of his outcomes has been that binary-field operations can be performed about 1.58, 1.42, and 1.27 times faster than prime-field operations for 191/192, 233/224, and 283/256 bits respectively. However, he did not compare the hardware requirements of both types of supported fields and reported only the total area requirements of his processor which is between 24-31 kGEs.

In the following, we design both a binary and prime-field based ECC processor in order to compare them in a fair environment. In contrast to existing work, we consider not only scalar multiplication but evaluate and compare the performance also for higher-level protocols such as ECDSA. First of all, we give a brief introduction into ECC and define a metric to compare different criteria which is done in the next section.

3 Implementations of Elliptic-Curve Cryptography

Elliptic curves have been introduced by Koblitz [22] and Miller [28] in the 1980s and they have been thoroughly analyzed by the community throughout the last decades. They are based on the Weierstrass equation which can be written as

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

with $a_{i=1,2,3,4,6}, x, y \in K$. K defines the finite field. A point $P = (x, y)$ is a valid point on the elliptic curve if it fulfills the Weierstrass equation, *i.e.* Equation (1). The basic operations performed on the elliptic curve are point addition and point doubling. Using those operations, a point multiplication (often referred as scalar multiplication) $Q = k \times P$ can be calculated. The Elliptic Curve Discrete Logarithm Problem (ECDLP) states that finding k is a mathematical hard problem if the points P and Q are given. For a more detailed introduction into elliptic curves and its properties we refer the reader to [3,5,13,23].

Figure 1 shows the hierarchy of ECC implementations. All ECC operations are based on finite-field arithmetics. Higher-level protocols make use of the underlying ECC operations to provide various cryptographic services such as authentication, data integrity, non-repudiation, or confidentiality. Note that most of these protocols (such as ECDSA) require different operations over finite fields such as prime-field addition or multiplication.

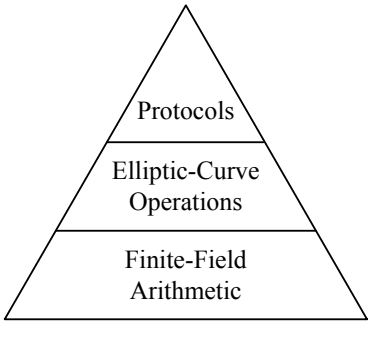

Fig. 1. Hierarchy of ECC implementations

Among the most commonly used types of finite fields are prime fields $\mathbb{F}_p$ and binary-extension fields $\mathbb{F}_{2^m}$. These types have different characteristics so that the Weierstrass equation can be simplified and different formulas for point addition and point doubling can be derived. Due to the differences of those fields, the performance of both software and hardware implementations can vary significantly.

In this paper, we compare two ECC implementations that are based on $\mathbb{F}_{2^{191}}$ and on $\mathbb{F}_{p_{192}}$. Both implementations use multi-precision arithmetic that means that all finite-field elements are split into smaller bit vectors of size W. Note that the one-bit difference does not have an impact in a relative comparison

of ECC implementations since we use the same metric for both implementations. As elliptic curves, we decided to use the recommended NIST prime-field curve P-192 [30] and the ANSI X9.62 compliant binary-field curve B-191 [1], *i.e.* `c2tnb191v1`. This is because we would like to compare curves with nearly identical bit sizes (191 vs. 192 bits). The one-bit difference between those two fields can be considered as negligible.

Throughout the paper, we used the following notation. For prime fields with modulo p, $n = \lceil log_2(p) \rceil$ bits are required to represent a number. For binary fields with $f(z) = z^m + r(z)$ denoting an irreducible binary polynomial of degree m, a bit-vector with m entries can be used to represent any binary polynomial. Consequently the number of needed words to represent a $\mathbb{F}_p$ number is $N = \lceil n/W \rceil$ and number of needed words to represent a $\mathbb{F}_{2^m}$ polynomial is $M = \lceil m/W \rceil$.

3.1 Comparison Metric and Criteria

The efficiency of ECC-hardware implementations depends on different criteria. In order to make a fair comparison, we introduce the following metric consisting of four main attributes:

- The **area** requirement of a chip is important for any cost-sensitive application. This is because the area largely determines the chip costs at fabrication.
- Embedded systems require **low-power** and
- **low-energy** designs. This is an important issue especially in battery-powered environments.
- **Speed** of computation is important for many applications to be applicable in practice. The most neutral unit for measurement is the number of cycles it takes to perform a certain operation.

The maximum **frequency** that can be used to clock a design has a direct impact on the resulting execution time (speed) of any algorithm. But the previously mentioned applications heavily constrain the maximum frequency anyways, so we do not include the frequency measure into our metric.

Because the energy $W = Pt$ is defined as product of the electrical power P and time t, its properties are not handled explicitly within Section 4.

4 Comparing ECC-Hardware Designs over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$

In this section, we compare ECC hardware designs and the respective algorithms over $\mathbb{F}_{2^m}$ and over $\mathbb{F}_p$. We describe the differences of the finite-field operations, the respective elliptic-curve group operations, and compare the hardware designs of both types of fields regarding cryptographic protocols like ECDSA.

Algorithm 1. Prime-field addition

Require: Two integers $a, b \in [0, p-1]$ and modulus p.
Ensure: $c = (a + b) \pmod{p}$.
1: $(\varepsilon, C[0]) \leftarrow A[0] + B[0]$.
2: **for** i from 1 to $N - 1$ **do**
3: $\quad(\varepsilon, C[i]) \leftarrow A[i] + B[i] + \varepsilon$.
4: **end for**
5: **if** $\varepsilon = 1$ or $c \geq p$ **then**
6: $\quad(\varepsilon, C[0]) \leftarrow C[0] - P[0]$.
7: $\quad$**for** i from 1 to $N - 1$ **do**
8: $\quad\quad(\varepsilon, C[i]) \leftarrow C[i] - P[i] - \varepsilon$.
9: $\quad$**end for**
10: **end if**
11: Return(c).

Algorithm 2. Binary-field addition

Require: Binary polynomials $a(z), b(z)$ with maximum degree m-1.
Ensure: $c(z) = a(z) + b(z)$.
1: **for** i from 0 to $M - 1$ **do**
2: $\quad C[i] \leftarrow A[i] \oplus B[i]$.
3: **end for**
4: Return(c).

4.1 Finite-Field Arithmetics

Modular Addition and Subtraction. The most basic finite-field algorithms are addition and subtraction. Algorithm 1 and Algorithm 2 show modular-addition algorithms over $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$. The major difference of those algorithms is the carry propagation ε. The polynomial addition is a simple XOR operation that does not incorporate a carry. A $\mathbb{F}_p$ addition, in contrast, needs up to three times more operations: the actual addition, a comparison of the result c with the prime p, and a modular reduction afterwards. By extending the range of the integers a, b, c form $[0, p-1]$ to $[0, 2^{N*W} - 1]$, the comparison operation ($c \geq p$) can be avoided, which reduces the total number of arithmetic operations by about a third. Notice that for this partial reduction, all other operations handling a, b, c must be prepared for their extended range. A modular subtraction works similar as the modular addition.

Modular Multiplication. Modular multi-precision multiplications are usually realized by following an operand-scanning or product-scanning multiplication approach. A multiply-accumulate unit (cf. [12,15,16]) can be used to increase

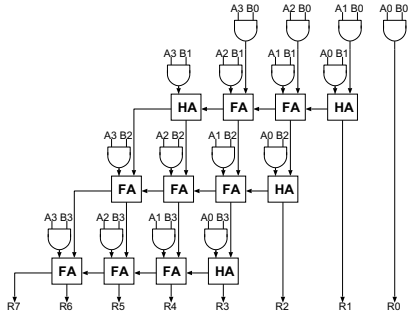

Fig. 2. 4-bit integer multiplier for $\mathbb{F}_p$

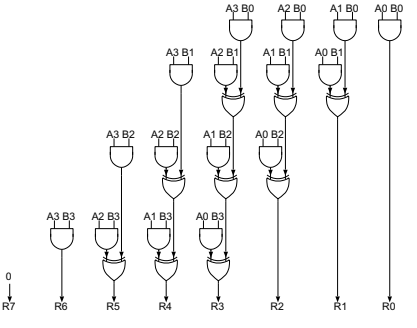

Fig. 3. 4-bit carry-less multiplier for $\mathbb{F}_{2^m}$

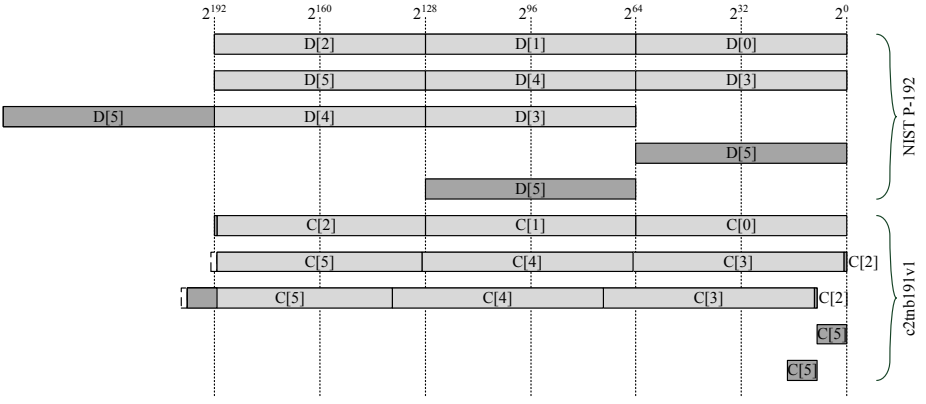

Fig. 4. Reduction using $p = 2^{192} - 2^{64} - 1$ on the top. Reduction using $f(z) = z^{191} + z^9 + 1$ on the bottom. In both cases, the product must be shifted and summed up.

the efficiency of the product-scanning method. Such a multiply-accumulate unit can be designed for $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$. Figures 2 and 3 show the internal structure of 4-bit multipliers for integers and polynomials. The biggest advantage of the carry-less multiplier for $\mathbb{F}_{2^m}$ are the shorter critical path and the smaller area requirement (logical XOR cells are used instead of full-adder standard cells). Thus, the difference between a $\mathbb{F}_{2^m}$ and a $\mathbb{F}_p$ multiplication module can be up to 40 % in terms of area requirement. Also the power consumption for a multiplier designed out of XORs instead of full-adders is lower. However the execution times (in cycles) for an integer or binary-polynomial multiplication using the product-scanning method are equivalent.

A finite-field multiplication always needs a reduction. There exist many ways to realize modular reduction in hardware. One efficient way is to apply a (fast) reduction method using special primes, so called Mersenne-like primes, which are often used for recommended and standardized elliptic curves (e.g. the NIST recommended curves [30]). Figure 4 shows how intermediate multiplication results can be reduced using this fast reduction method for primes and polynomials over the curves NIST P-192: $p = 2^{192} - 2^{64} - 1$ and ANSI X9.62 `c2tnb191v1`: $f(z) = z^{191} + z^9 + 1$. The reduction can be performed with only shifts and additions. The for NIST P-192 necessary shift operations fit very well within the addressing scheme of 8-bit, 16-bit, or 32-bit architectures. The shift operations required by `c2tnb191v1` do not fulfill this property. However, in cases where the shift operations are smaller than W, an additional hardcoded reduction logic can be used. In terms of area, this reduction logic is very cheap (about the size of a $\mathbb{F}_{2^m}$ addition).

Modular Squaring. Modular squaring is equivalent to a modular multiplication with two identical operands. Thus, an explicit implementation is often not necessary, especially in implementations where low area is a stringent requirement. However, if implemented it improves the performance since it is typically faster than modular multiplications [13].

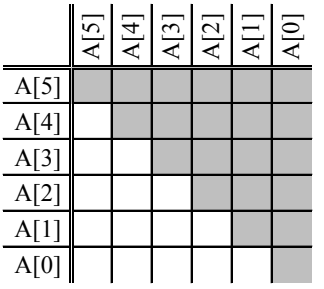

Fig. 5. Prime-field squaring operation. The necessary intermediate multiplications are shaded.

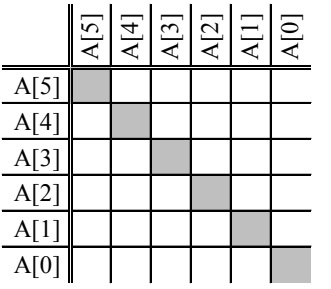

Fig. 6. Binary-field squaring operation. The necessary intermediate multiplications are shaded.

During a prime-field squaring operation, the two intermediate products $A[i] \times A[j]$ and $A[j] \times A[i]$, $\forall i \neq j \in [0, N-1]$, are identical. Figure 5 shows the operands of a 6-word squaring operation where only the necessary operations (multiplications) are shaded. Thus, the squaring operation can be up to two times faster than a multiplication.

Squarings over binary fields, as opposed, have the nice property that $a_i \times a_j + a_j \times a_i = 0$, $\forall i \neq j \in [0, m-1]$. If $a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0$, then $a(z)^2 = a_{m-1}z^{2m-2} + \cdots + a_2z^4 + a_1z^2 + a_0$. Thus, zero values are simply inserted between two consecutive bits a_i. Utilizing the binary multiplier from Figure 3, only M multiplications $A[i] \times A[i]$ are required to perform a binary-field squaring operation. As it can also be seen in Figure 6, the squaring operation is M times faster than a binary field multiplication. It can be performed with a similar runtime complexity as a modular addition.

In terms of runtime and lines-of-code, a $\mathbb{F}_{2^m}$ squaring can be up to $\frac{N^2}{2M}$ times faster than a $\mathbb{F}_p$ squaring.

Modular Inversion. What the inversion operations for prime and binary fields have in common is the very slow execution time. There are two common inversion methods. One is based on the extended Euclidean algorithm and one is based on Fermat's little theorem ($a = a^{2^m} \bmod f(z)\ \forall a \in \mathbb{F}_{2^m}$). For this paper the Montgomery inversion technique by Kalinski *et al.* [20] has been used for prime field inversion operations. Using Fermat's little theorem [18] for binary field inversions, with $a^{-1} \equiv a^{2^m-2} \bmod f(z)$, a field inversion can be performed by using $m-1$ squarings and several multiplications. In the case of `c2tnb191v1`, 190 squarings and 12 multiplications are necessary. Because of the fast squaring operations within binary fields, the runtime of this method exceeds any Euclidean-based algorithm. [14] gives a comparison of different algorithms for an inversion within the NIST B-163 field.

4.2 Elliptic-Curve Operations

The performance of EC-group operations over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ differ significantly. We used formulae that reflect the state of the art in efficient ECC implementations.

For binary-field arithmetic, we applied the formulae proposed by J. López and R. Dahab [27]. Their formulae need six finite-field multiplications, five squarings, and three additions per key bit. For prime-field arithmetic, we applied the formulae of M. Hutter et al. [17] needing 12 multiplications, four squarings, and 16 additions (incl. subtractions). Both formulae have been applied within the Montgomery powering ladder scalar multiplication [19]. By comparing the formulae, it clearly shows that the binary formulae need 50 % less multiplications than the formulae over prime-field arithmetic. This is one of the most advantageous properties that encourages the use of $\mathbb{F}_{2^m}$ operations in ECC-hardware implementations. Note that both formulae use projective coordinates that means that no modular inversion is needed throughout the scalar multiplication[1].

4.3 Cryptographic Protocols

After the basic elliptic-curve operation of a scalar multiplication, we compare the performance of $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ processors in terms of higher-level protocols. In particular, we implemented ECDSA [30] on both types of (binary and prime-field based) processors. The main additional operations needed to support ECDSA is the SHA-1 [29] algorithm[2] to calculate the message digest of the message m and some prime-field operations, *i.e.* modular addition, multiplication, and inversion to calculate the digital signature $(r, s) = (k \times P, k^{-1}(\text{SHA-1}(m) + rd))$, where d represents the used private key.

For a more efficient ECDSA-verify algorithm, we additionally implemented a different methodology for calculating point multiplications. First, we applied Shamir's trick [8,9] to improve the performance of multiple point multiplication. Second, we used different formulae to perform the verification using Jacobian-projective coordinates [13] for the prime-field processor and López-Dahab coordinates [13,26] for the binary-field processor.

5 Comparison Results

For a fair comparison of binary field and prime-field ECC implementations, it is important to select a common controlling engine, common development tools, the same process technology, and elliptic curves of nearly the same bit size.

As a controller, we decided to use our own 16-bit microcontroller called Neptun [33,34,35] that is especially optimized for elliptic-curve cryptography. The processor comes with twelve special-purpose registers and uses a Harvard architecture with separated program and data memory. The usually area consuming data memory is made from a very area-efficient single-port RAM macro[3]. The program memory is a synthesized lookup table stored as Read-Only Memory

[1] Inversion is only needed after the calculation of $k \times P$ to convert the projective coordinates back to affine coordinates.

[2] Included in the design. 16-bit CPU is used to calculate the hash.

[3] All following area-related results would be different if latch or register-based RAM's are used.

Table 1. Prime-field vs. binary-field operations of our ECC-hardware architecture

	Cycles		Lines of Code		Operations/key-bit	
	$\mathbb{F}_{p192}$	$\mathbb{F}_{2^{191}}$	$\mathbb{F}_{p192}$	$\mathbb{F}_{2^{191}}$	$\mathbb{F}_{p192}$ [17]	$\mathbb{F}_{2^{191}}$ [27]
Addition/Subtraction	64	38	64	38	16	3
Multiplication	329	265	329	265	12	6
Squaring	190	45	190	45	4	5
Inversion	46,560	14,611	397	117	-	-

(ROM). In fact, the area requirements of this lookup table is proportional to the number of lines-of-code (LOC) stored within the program memory. The central processing unit (CPU) is capable of the most basic arithmetic operations such as addition/subtraction, logic operations (AND, OR, XOR), and shift operations.

As target technology, we selected a 130nm low-leakage CMOS technology by UMC. This technology needs fewer power compared to larger 180 nm and 350 nm technologies and has a lower power leakage than smaller (*e.g.* 90 nm) technologies. The standard-cell library has been provided by Faraday Technology. The RAM-macro blocks have been generated using the Standard Memory Compiler FSA0A Memaker 200901.1.1 by the Faraday Technology Corporation [10]. For synthesis we used the Cadence RTL compiler [7] Version v08.10. For power-simulations we used Cadence First Encounter Version v08.10.

5.1 Finite-Field Arithmetic

The finite-field algorithms have been implemented as described in Section 4.1. All algorithms (except the algorithms for modular inverses) have been unrolled and optimized for our custom microcontroller instruction set (Assembler language). All results are summarized in Table 1.

It shows that our processor performs the binary-field addition about 40.6 % faster than the prime-field addition (the same holds for modular subtraction). Binary-field multiplication is 19.5 % faster than its prime-field counterpart because of the extra reduction logic provided to take advantage of the Mersenne-like irreducible polynomial. However, it shows that even when multi-precision arithmetic is used, the biggest advantage of binary-field operations is within the squaring operation. Its runtime is 4.2 times faster than the prime-field squaring operation. Finally, the two very distinct inversion techniques, discussed in Section 4.1, result in very different runtime and LOC results. The binary-field inversion implementation is 3.19 times faster and needs only 29.5 % LOC. It reuses the squaring and multiplication methods and subsequently only works for a single irreducible polynomial. The prime-field inversion, in contrast, works for any prime. The main reason for the higher code size are actually the additional utility functions (addition, subtraction, multiplication with 2, division by 2) that had to be implemented.

Table 2. Comparison of prime field vs. binary-field ECC implementations

Algorithm	**Q = k × P**		**ECDSA Sign**		**ECDSA Verify**	
	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{2^{191}}$	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{2^{191}}$	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{2^{191}}$
Integer multiplier	required	–	required	required	required	required
Carry-less multiplier	–	required	–	required	–	required
Cycles	**1,312,616**	**399,635**	**1,393,523**	**494,983**	**1,417,422**	**892,124**
RAM entries	100	90	112	103	174	162
Program entries	1,207	699	1,662	1,689	1,519	1,875
Constants	61	60	100	129	100	141
Area requirements [GE]						
CPU	4,041	3,653	4,049	4,393	4,066	4,422
Program memory	4,494	2,683	7,203	7,432	7,589	8,031
Data memory	3,040	2,963	3,390	3,412	4,088	4,160
Total area	**11,579**	**9,301**	**14,644**	**15,293**	**15,747**	**16,618**
Power consumption @ 1 MHz [μW]						
CPU	20.40	19.99	18.36	18.48	17.93	20.38
Program memory	8.95	4.01	7.89	8.22	8.89	8.16
Data memory	10.59	8.92	11.91	10.86	11.80	12.52
Total power	**41.37**	**34.78**	**39.54**	**39.47**	**40.55**	**43.12**
Energy consumption [μJ]						
Energy	**54.30**	**13.90**	**55.10**	**19.53**	**57.48**	**38.47**

5.2 Elliptic-Curve Operations

Table 2 compares the absolute values and Table 3 compares the relative differences of the implemented prime-field and binary-field ECC implementations. The relative differences shown in Table 3 have been calculated using the Formula $\frac{Param(\mathbb{F}_{2^m})}{Param(\mathbb{F}_p)} - 1$. In the following, we separately consider point multiplication as well as signature generation and verification of the higher-level protocol of ECDSA.

In view of point multiplication, it shows that the binary-field based implementation is 3.28 times faster than the prime-field based opponent. The area requirement is 19.7 % better and the power consumption is 15.9 % lower for the binary-field processor. This results in an energy consumption which is 3.91 times lower than the calculation over prime fields. Note that the area difference mostly comes from the size of the program memory, the used multiplier within the CPU and the size of the necessary RAM macro. Even note that in both designs, about 50 % of the total power is consumed within the CPU.

Table 3. Relative difference between $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ based implementations

Algorithm	**Q = k × P**	**ECDSA Sign**	**ECDSA Verify**
Runtime	−69.6 %	−64.5 %	−37.1 %
Area	−19.7 %	+4.4 %	+5.5 %
Power	−15.9 %	±0.0 %	+6.3 %
Energy	−74.4 %	−64.6 %	−33.1 %

Table 4. Comparison of different ECC implementation with related work

	ECC Curve	Area [GE]	Cycles [kCycles]	Power[a] μW	Energy μJ	VLSI technology
Auer 2009 [2]	$\mathbb{F}_{p_{192}}$	24,750	1,031	613.65	632.67	AMS C35
Fürbass 2007 [11]	$\mathbb{F}_{p_{192}}$	23,656	500	1,692.11	846.06	AMS C35
Wolkerstorfer 2005 [36]	$\mathbb{F}_{p_{192}}$	23,818	678	500.00	340.00	350nm
This work 2011	$\mathbb{F}_{p_{192}}$	**11,579**	**1,313**	**41.37**	**54.30**	**UMC L130**
Lee 2008 d=4 [25]	$\mathbb{F}_{2^{163}}$	15,356	79	37.39	2.95	UMC L130
Lee 2008 d=1 [25]	$\mathbb{F}_{2^{163}}$	12,506	276	32.42	8.95	UMC L130
Batina[b] 2006 d=4 [4]	$\mathbb{F}_{2^{163}}$	14,816	95	27.00	2.57	130nm
Batina[b] 2006 d=1 [4]	$\mathbb{F}_{2^{163}}$	13,104	354	27.00	9.56	130nm
Bock 2008 d=8 [6]	$\mathbb{F}_{2^{163}}$	16,247	47	148.76	6.99	INF SRF55V01P
Bock 2008 d=1 [6]	$\mathbb{F}_{2^{163}}$	10,392	280	54.31	15.21	INF SRF55V01P
Hein 2008 [15]	$\mathbb{F}_{2^{163}}$	11,904	296	101.87	30.15	UMC L180
Kumar 2006 [24]	$\mathbb{F}_{2^{163}}$	15,094	430	-	-	AMI C35
Kumar 2006 [24]	$\mathbb{F}_{2^{193}}$	17,723	565	-	-	AMI C35
Wolkerstorfer 2005 [36]	$\mathbb{F}_{2^{191}}$	23,818	426	500.00	213.00	350nm
This work 2011	$\mathbb{F}_{2^{191}}$	**9,301**	**399**	**34.78**	**13.90**	**UMC L130**

[a] All reference values were scaled to 1 MHz.
[b] RAM approximated with 4,890 GE. Power-consumption values do not include RAM.

5.3 Cryptographic Protocols

For ECDSA, only 455 lines of code (38 %) have to be added to the prime-field ECC processor to support all operations to sign data. This and the small increase of necessary RAM entries increased the total area requirement by 26.5 %. The execution time is increased by only 6.2 %. The differences in power and energy consumption are hardly noticeable. The changes to the binary-field ECC processor are much more significant. The CPU had to be extended with a small 8-bit integer multiply-accumulate unit, making it capable of prime and binary-field operations, increasing the area requirements of the CPU by 20 %. Adding all those algorithms increased the size of the program memory by 177 % and the total area of the processor by 64 %. Also the power and energy consumption increased by 13.5 % and 40.5 %. However, the runtime of the binary-field based ECDSA processor is still 2.82 times faster than the runtime of the prime-field based ECDSA processor. Even though the area and power consumption are approximately identical, the binary-field ECDSA processor needs 2.82 times less energy than the prime-field ECDSA processor.

The ECDSA verification needs one additional point multiplication compared to the ECDSA-signature generation algorithm which needs only one. Cause of Shamir's trick the runtime for the prime-field based algorithms differ by only 2 %. The area differs by 7.5 % and the power and energy results are almost identical. The ECDSA-signature verification algorithm over binary fields does not handle the two point multiplications as well. Whereas the area increased by only 8.7 %, the runtime increased by 80 %. This doubles the required energy needed for an ECDSA-signature verification compared to an ECDSA-signature generation.

Table 5. Comparison of our ECDSA implementations with related work

	ECC Curve	Area [GE]	Cycles [kCycles]	Power[a] μW	Energy μJ	VLSI technology
Kern 2010 [21]	$\mathbb{F}_{p160}$	18,247	512	860.00	440.32	AMS C35
Hutter 2010 [16]	$\mathbb{F}_{p192}$	19,115	859	1,507.79	1,295.19	AMS C35
Wenger[b] 2010 [34]	$\mathbb{F}_{p192}$	11,686	1,377	113.86	156.79	UMC L180
This work[b] 2011	$\mathbb{F}_{p192}$	**14,644**	**1,394**	**39.54**	**55.10**	**UMC L130**
This work 2011	$\mathbb{F}_{2^{191}}$	**15,293**	**495**	**39.47**	**19.53**	**UMC L130**

[a] All reference values have been scaled to 1 MHz.

[b] Nearly identical designs were used. The differences in area and power come from the different technologies and synthesizers used.

5.4 Comparison with Related Work

Table 4 gives a comparison with related work. All power results have been scaled to 1 MHz. The first five rows give related work over prime fields. The remaining rows contain related work over binary fields. Our $\mathbb{F}_p$ processor is 51 % smaller than the best related design by Wolkerstorfer [36]. In terms of cycles this processor is above average. Only the energy requirement by Öztürk [31] design is lower, but their design is not based on NIST P-192. The area results of the math processor are 10.4 % smaller than the smallest related implementation. Our speed, power, and energy results are larger than many other designs, but it should be noted that those designs have an advantage cause of the smaller elliptic curve used.

Table 5 summarizes related work regarding low-resource ECDSA-hardware implementations. In terms of power and energy consumption, we outperform existing solutions. The area requirements are lower than the work of Kern [21] and Hutter [16] but are higher than the work of Wenger [34].

6 Conclusion

In this paper, we compared the performance of two distinct ECC-hardware implementations that are based on prime-field (NIST P-192) and binary-field (ANSI `c2tnb191v1`) arithmetic. The comparison of the finite-field algorithms showed us the clear runtime advantage of the squaring (4.2 times) and addition (1.7 times) operations within the binary-extension field. When doing point multiplications, the $\mathbb{F}_{2^m}$ based processor outperforms the $\mathbb{F}_p$ based processor by 19.7 % in area, 69.6 % in runtime, 15.9 % in power, and 74.4 % in energy. In addition to these outcomes, we analyzed the impact of higher-level protocols on the finite-field processors. The implementation of both digital-signature generation and verification using ECDSA had led us to interesting findings. It was shown that the area and power advantages for the $\mathbb{F}_{2^m}$ based processor vanish while it still is 1.5-2.8 times faster and consequently more energy efficient than the $\mathbb{F}_p$ based processor.

These results can be applied to any future design of an ASIC ECC processor that is integrated in an area, power, or energy constrained device.

Acknowledgements. The work has been supported by the European Commission through the ICT program under contract ICT-SEC-2009-5-258754 (Tamper Resistant Sensor Node - TAMPRES) and by Austrian Science Fund (FWF) under grant number P22241-N23.

References

1. American National Standards Institute (ANSI). American National Standard X9.62-2005. Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA) (2005)
2. Auer, A.: Scaling Hardware for Electronic Signatures to a Minimum. Master thesis, University of Technology Graz (October 2008)
3. Avanzi, R.M., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. Chapman & Hall/CRC (2005)
4. Batina, L., Mentens, N., Sakiyama, K., Preneel, B., Verbauwhede, I.: Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks. In: Buttyán, L., Gligor, V.D., Westhoff, D. (eds.) ESAS 2006. LNCS, vol. 4357, pp. 6–17. Springer, Heidelberg (2006)
5. Blake, I.F., Seroussi, G., Smart, N.P.: Elliptic Curves in Cryptography. London Mathematical Society Lecture Notes Series, vol. 265. Cambridge University Press, Cambridge (1999)
6. Bock, H., Braun, M., Dichtl, M., Hess, E., Heyszl, J., Kargl, W., Koroschetz, H., Meyer, B., Seuschek, H.: A Milestone Towards RFID Products Offering Asymmetric Authentication Based on Elliptic Curve Cryptography. Invited talk at RFIDsec 2008 (July 2008)
7. Cadence Design Systems, Inc., San Jose, California, United States (2011). The Cadence Design Systems Website, `http://www.cadence.com/`
8. de Rooij, P.: Efficient Exponentiation Using Precomputation and Vector Addition Chains. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 389–399. Springer, Heidelberg (1995)
9. El Gamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)
10. Faraday Technology Corporation. Faraday FSA0A_C 0.13 μm ASIC Standard Cell Library (2004), `http://www.faraday-tech.com`
11. Fürbass, F., Wolkerstorfer, J.: ECC Processor with Low Die Size for RFID Applications. In: Proceedings of 2007 IEEE International Symposium on Circuits and Systems. IEEE (May 2007)
12. Großschädl, J., Savaş, E.: Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m). In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 133–147. Springer, Heidelberg (2004)
13. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2004)
14. Hein, D.: Elliptic Curve Cryptography ASIC for Radio Frequency Authentication. Master thesis, Technical University of Graz (April 2008)
15. Hein, D., Wolkerstorfer, J., Felber, N.: ECC Is Ready for RFID – A Proof in Silicon. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 401–413. Springer, Heidelberg (2009)

16. Hutter, M., Feldhofer, M., Plos, T.: An ECDSA Processor for RFID Authentication. In: Ors Yalcin, S.B. (ed.) RFIDSec 2010. LNCS, vol. 6370, pp. 189–202. Springer, Heidelberg (2010)
17. Hutter, M., Joye, M., Sierra, Y.: Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 170–187. Springer, Heidelberg (2011)
18. Itoh, T., Tsujii, S.: Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$. Electronic Letters 24(6), 334–335 (1988)
19. Joye, M., Yen, S.-M.: The Montgomery Powering Ladder. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 291–302. Springer, Heidelberg (2003)
20. Kaliski, B.: The Montgomery Inverse and its Applications. IEEE Transactions on Computers 44(8), 1064–1065 (1995)
21. Kern, T., Feldhofer, M.: Low-Resource ECDSA Implementation for Passive RFID Tags. In: Proceedings of 17th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2010), Athens, Greece, December 12-15, pp. 1236–1239. IEEE (2010)
22. Koblitz, N.: Elliptic Curve Cryptosystems. Mathematics of Computation 48, 203–209 (1987)
23. Koblitz, N.: A Course in Number Theory and Cryptography. Springer, Heidelberg (1994) ISBN 0-387-94293-9
24. Kumar, S.S., Paar, C.: Are standards compliant Elliptic Curve Cryptosystems feasible on RFID? In: Workshop on RFID Security (RFIDSec 2006), Graz, Austria, July 12-14 (2006)
25. Lee, Y.K., Sakiyama, K., Batina, L., Verbauwhede, I.: Elliptic-Curve-Based Security Processor for RFID. IEEE Transactions on Computers 57(11), 1514–1527 (2008)
26. López, J., Dahab, R.: Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n). In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 201–212. Springer, Heidelberg (1999)
27. López, J., Dahab, R.: Fast Multiplication on Elliptic Curves over GF(2^m). In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 316–327. Springer, Heidelberg (1999)
28. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
29. National Institute of Standards and Technology (NIST). FIPS-180-3: Secure Hash Standard (October 2008), `http://www.itl.nist.gov/fipspubs/`
30. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS) (2009), `http://www.itl.nist.gov/fipspubs/`
31. Öztürk, E., Sunar, B., Savaş, E.: Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 92–106. Springer, Heidelberg (2004)
32. Satoh, A., Takano, K.: A Scalable Dual-Field Elliptic Curve Cryptographic Processor. IEEE Transactions on Computers 52(4), 449–460 (2003)
33. Wenger, E., Feldhofer, M., Felber, N.: A 16-Bit Microprocessor Chip for Cryptographic Operations on Low-Resource Devices. In: Proceedings of Austrochip 2010, Villach, Austria, October 6, pp. 55–60 (2010) ISBN 978-3-200-01945-4
34. Wenger, E., Feldhofer, M., Felber, N.: Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices. In: Chung, Y., Yung, M. (eds.) WISA 2010. LNCS, vol. 6513, pp. 92–106. Springer, Heidelberg (2011)

35. Wenger, E., Hutter, M.: A Hardware Processor Supporting Elliptic Curve Cryptography for Less Than 9kGEs. In: Proceedings of the Tenth Smart Card Research and Advanced Application Conference, CARDIS 2011, Leuven, Belgium, September 15-16 (2011)
36. Wolkerstorfer, J.: Is Elliptic-Curve Cryptography Suitable for Small Devices? In: Workshop on RFID and Lightweight Crypto, Graz, Austria, July 13-15, pp. 78–91 (2005)