A Low-Cost ECC Coprocessor for Smartcards

Harald Aigner[1], Holger Bock[2], Markus Hütter[2], and Johannes Wolkerstorfer[3]

[1] D. Swarovski & Co.,
6112 Wattens, Austria.
Ches04@Aigner.name
[2] Infineon Technologies,
Development Center Graz, Austria.
{Holger.Bock, Huetter.External}@infineon.com, http://www.infineon.com/
[3] Institute for Applied Information Processing and Communications,
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria.
Johannes.Wolkerstorfer@iaik.at, http://www.iaik.at/

Abstract. In this article we present a low-cost coprocessor for smartcards which supports all necessary mathematical operations for a fast calculation of the Elliptic Curve Digital Signature Algorithm (ECDSA) based on the finite field GF(2^m). These ECDSA operations are GF(2^m) addition, 4-bit digit-serial multiplication in GF(2^m), inversion in GF(2^m), and inversion in GF(p). An efficient implementation of the multiplicative inversion which breaks the 11:1 limit regarding multiplications makes it possible to use affine instead of projective coordinates for point operations on elliptic curves. A bitslice architecture allows an easy adaptation for different bit lengths. A small chip area is achieved by reusing the hardware registers for different operations.

Keywords: Elliptic Curve Cryptography (ECC), digital signature, multiplicative inverse, hardware implementation.

1 Introduction

Smartcards offer a high-quality identification method by means of digital signatures. This identification provides legally effective authenticity, confidentiality, integrity, and non-repudiation of transactions in e-business, e-government, m-commerce, and Internet applications.

The Digital Signature Algorithm based on elliptic curves (ECDSA) is commonly used for achieving authenticity. Elliptic curve cryptography allows to use short key sizes compared to other cryptographic standards such as RSA. Short keys are especially favourable for targeting smartcards because smartcards typically offer very limited resources. These limited resources also motivate usage of a coprocessor to accelerate the time-consuming calculations of ECDSA and other cryptographic operations.

This paper presents a coprocessor which can be integrated into the Infineon SLE66CXxxxP family and allows a significant speed-up of ECDSA calculation.

This is achieved by a fast and compact implementation of the underlying arithmetic operations. We identified mainly three operations that are crucial for performance. These operations are multiplication in the finite field $\mathrm{GF}(2^m)$ and the computation of the multiplicative inverses in $\mathrm{GF}(p)$ and $\mathrm{GF}(2^m)$.

In particular, accelerated $\mathrm{GF}(2^m)$ inversion, presented in this paper, allows to use affine coordinates instead of projective coordinates. Affine coordinates become attractive when the calculation of the $\mathrm{GF}(2^m)$ inversion requires less time than about 11 multiplications. This relation origins from the additional multiplications that become necessary when using projective coordinates. More details can be found in Section 3. Affine coordinates use simpler formulas for calculating EC operations. They consist of less finite field operations and require a smaller number of auxiliary variables. Therefore, the usage of affine coordinates saves memory, registers and reduces the number of bus transfers, all of which are scarce resources on smartcards.

The remainder of this article is structured as follows: the next section gives an overview over related work. Section 3 introduces the mathematical background of elliptic curve cryptography, point operations on elliptic curves, and the ECDSA. The target smartcard architecture and the coprocessor hardware is presented in Section 4. Section 5 summarizes implementation results of the coprocessor. Conclusions are drawn in Section 6.

2 Related Work

The recently published book *Guide to Elliptic Curve Cryptography* gives a comprehensive overview on the state-of-art of implementing elliptic-curve cryptosystems in hardware and in software [2]. In this article we will narrow our view on related hardware implementations. Unfortunately, none of the published hardware implementations is targeted towards an ECC coprocessor for 8-bit smartcards. This is unpleasant because the intended application has an enormous impact on the design of an optimized ECC hardware. The target application fixes many parameters for which a circuit can be optimized. For instance the parameter *throughput*: a server application might demand several thousand EC operations per second, whereas a smartcard may be contented with ten operations per second or even less. Other parameters influencing efficiency are scalability (the ability to adopt to other operand sizes or other finite fields), energy efficiency, the desired target technology (FPGA, ASIC, or ASSP), the amount of hardware resources required (gate count), and last-but-not-least security aspects (robustness against side channel attacks like timing attacks, SPA, and DPA).

Different design parameters will lead to different ECC implementations. The range of possible ECC implementations is large: starting from pure software implementations, instruction-set extensions (ISE) became popular for 16-bit and 32-bit platforms to accelerate ECC over $\mathrm{GF}(2^m)$ [3]. ISE are not useful for 8-bit platforms because slow data transport in 8-bit systems will deteriorate accelerated field operations. Alternatives are heavy-weight accelerators for complete EC operations [4, 5, 7, 8] or hardware-software co-design approaches where computational intensive tasks are done by an EC coprocessor [9]. These coprocessors can either calculate all finite field operations [12] or support only multiplication as the most demanding finite field operation [10, 11]. Circuits for calculating the multiplicative inverse in the finite fields $\mathrm{GF}(p)$ and $\mathrm{GF}(2^m)$ are rare [12, 13].

The most obvious operation to support in hardware is multiplication because multiplication contributes most to the runtime of EC operations. Fast multiplication even helps to speedup the calculation of the multiplicative inverse when using Fermat's theorem. Fermat's theorem allows to calculate the inverse by exponentiation. Exponentiation, in turn, can be calculated by repeated multiplications [1]. Even than, exponentiation takes more than 100 times longer than multiplication which makes the use of affine coordinates for EC operations unattractive. Useful multipliers which can operate both in $\mathrm{GF}(p)$ and $\mathrm{GF}(2^m)$ were presented by J. Großschädl [11] and E. Savaş et al. [10]. J. Großschädl's approach uses a dual-field bit-serial multiplier utilizing interleaved modular reduction. The achieved $\mathrm{GF}(p)$ performance is slower than the $\mathrm{GF}(2^m)$ performance. E. Savaş et al. approach bases on a Montgomery multiplier for both fields and allows to handle arbitrarily large operands due to a scalability feature which is achieved by a pipelined array of processing elements. Both approaches use a redundant representation for $\mathrm{GF}(p)$ results to circumvent critical-path problems caused by carry propagation in the $\mathrm{GF}(p)$ mode of operation.

Hardware accelerators for modular inversion usually base on the extended Euclidean algorithm or variants of it. The dual-field inversion circuit by A. Gutub et al. is no exception [13]. Their circuit is scalable which means it can calculate inverses of any length. This feature seems to come at a high price because performance is lower than attainable and the architecture seems to have interconnect penalties due to a large number of wide buses getting multiplexed. J. Wolkerstorfer manages to embed the inversion functionality for $\mathrm{GF}(p)$ and $\mathrm{GF}(2^m)$ into a dual-field arithmetic unit at negligible additional cost compared to the cost of a mere dual-field multiplication unit [12]. Nevertheless, inversion takes 70 times longer than multiplication.

Some implementations of EC processors have no hardware support for inversion [9]. For other implementations it remains unclear whether they have or not [6]. EC processors with very fast high-radix multipliers (which require substantial hardware resources) often lack dedicated inversion circuitry. They calculate inverses via Fermat's theorem to reuse the multiplier. The EC processor of G. Orlando et al. is an example for this [7]. A counter-example is the fastest known EC processor by N. Gura et al. [8]. This EC processor for server applications has a 256×64-bit multiplier and a separate inversion unit which calculates inverses in $2m$ clock cycles by running a variant of the extended Euclidean algorithm. In comparison, inversion calculated by exponentiation would take three times longer in the worst case. EC processors trimmed for energy-efficient operation have usually smaller multipliers with either bit-serial processing or a moderate degree of parallelization. Hence, inversion calculated via exponentiation would become slow too. Therefore, they often have hardware support for calculating the modular inverse using the extended Euclidean algorithm. An example is the so-called Domain-Specific Reconfigurable Cryptographic Processor by J. Goodman et al. [4] and the GF(2^{178})-EC-processor by R. Schroeppel et al. [5]. The latter can calculate inverses only in GF(2^{178}). The calculation of the GF(p) inverses for signature generation is avoided by using a modified signature scheme.

3 Mathematical Background

This section describes the point operations on elliptic curves and compares the use of affine coordinates with projective coordinates for point representation. It also gives an overview of the mathematical operations in the finite field GF(2^m). The section will end with a short description of the Elliptic Curve Digital Signature Algorithm (ECDSA).

The use of elliptic curves in cryptography was proposed first by Victor Miller [15] and Neal Koblitz [16] in 1985. The mathematical basis for the security of elliptic-curve cryptosystems is the computational intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP) leading to smaller key-sizes (compared to, e.g., RSA) which make elliptic curves attractive especially for smartcards where a small hardware implementation is desired.

3.1 Point Operations on Elliptic Curves

The points on an elliptic curve E together with the point at infinity $\mathcal{O}$ form an *abelian group* under an *addition* operation. Two distinct points P, $Q \in E$ can be added to $R = P + Q$. Performing this calculation involves several operations (addition, multiplication, and inversion) in the underlying field GF(2^m). In case $P = Q$, the addition turns into point *doubling* and uses slightly different formulas.

The *scalar multiplication* of a point $P \in E$ by an integer k is the sum

$$\overbrace{P + P + \cdots + P}^{k \text{ times}} = \sum_{k} P = kP \tag{1}$$

In cryptographic applications k can be very large (usually 163 or 191 bits) which would lead to an enormous computing time using repeated point addition. However, scalar multiplication can be performed more efficiently by the *double-and-add method* [17].

3.2 Point Representation on Elliptic Curves

There are two commonly used representations of points on elliptic curves: *affine coordinates* and *projective coordinates*. Various types of projective coordinates exist. Within this paper, the main focus is on *Jacobian* projective coordinates because they allow the fastest implementation of point doubling compared with other types like *standard* projective coordinates or *Chudnovsky* projective coordinates.

An affine point on an elliptic curve E is specified by a pair of finite field elements (x, y) which are called the *affine coordinates* for the point. The point at infinity $\mathcal{O}$ has no affine representation. It may be more efficient to compute numerators and denominators separately if division is expensive to calculate. For this reason, the affine coordinates are transformed into *projective coordinates* which consist of three elements (X, Y, Z).

The number of operations in the underlying finite field GF(2^m) for calculating point operations strongly depends on the chosen coordinate representation. Table 1 shows the number of additions, multiplications, and inversions in the finite field GF(2^m) and the number of auxiliary variables needed for an implementation according to [18].

	#Add.	#Mult.	#Inv.	#Var.
Point addition (affine)	9	3	1	2
Point doubling (affine)	6	3	1	2
Point addition (projective)	7	14	0	5
Point doubling (projective)	4	10	0	4

Table 1. Comparison of operations on elliptic curves over GF(2^m)

Table 1 shows that, e.g., a point addition takes 3 multiplications and 1 inversion in the underlying field with affine coordinates. It takes 14 multiplications using projective coordinates. Additions are not considered because they are very easy to calculate. Nearly all implementations of elliptic curves use projective coordinates. This leads to more multiplications but the costly calculation of the inverse can be avoided and calculation is still faster than using affine coordinates. However, calculating the multiplicative inverse at least as fast as $14 - 3 = 11$ multiplications makes it economical to use affine coordinates instead of projective coordinates with all advantages as described in the introduction. Affine coordinates become a little bit less attractive when they are compared with the projective version of Montgomery's ladder. This approach was proposed by Lopez and Dahab [14]. It uses only 11 multiplications for a combined point-addition and point-doubling operation. Thus, inversion has to be faster than 8 multiplications to make affine coordinates competitive. When comparing the affine version of Montgomery's method against the projective variant, inversion has to break a 5-to-1 limit.

3.3 Berlekamp's Variant of the extended Euclidean algorithm

E. Berlekamp introduced a variant of the binary extended Euclidean algorithm for calculating the multiplicative inverse in GF(2^m) in [20] along with a proposal for an efficient hardware implementation. A slight modification of this algorithm makes it possible to calculate the multiplicative GF(2^m) inverse in a constant time of $2m + 1$ clock cycles.

Using a bit-serial $GF(2^m)$ multiplier, a multiplication takes m clock cycles. With a 4-bit digit-serial multiplier this value is reduced to $\lceil \frac{m}{4} \rceil$ clock cycles. Thus, it is possible to perform an inversion faster than 11 multiplications both, with a bit-serial and a 4-bit digit-serial multiplier. This allows the use of affine coordinates instead of projective coordinates which avoids the use of coordinate transformations and reduces the number of auxiliary variables. Using the affine version of Montgomery's ladder is preferable when a bit-serial multiplier or a 2-bit-digit serial multiplier is used. Otherwise, the projective version will be faster.

3.4 Elliptic Curve Digital Signature Algorithm

Algorithm 1 shows the creation of an elliptic-curve digital signature. The inputs of the algorithm are the so called domain parameters (see [21]), a message m, and the key pair (d, Q). Random number generation and the SHA-1 hash-function are also needed but are usually calculated within a dedicated coprocessor and, therefore, are not considered in the following.

Algorithm 1 Elliptic Curve Digital Signature Algorithm - generation

Require: Message m, domain parameters
Ensure: Signature (r, s) of m
1: Select a random integer k, $1 \leq k \leq n - 1$.
2: Compute $kP = (x_1, y_1)$.
3: Compute $r = x_1 \bmod n$. If $r = 0$ go to step 1.
4: Compute $k^{-1} \bmod n$.
5: Compute $e = \text{SHA-1}(m)$.
6: Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ go to step 1.
7: **return** (r, s)

The remaining two main operations are the scalar multiplication (line 2 of Algorithm 1) which is calculated by means of addition, multiplication, and inversion in the finite field $GF(2^m)$ and $GF(p)$ inversion (line 4 of Algorithm 1). The coprocessor provides these functions and therefore allows a fast calculation of the ECDSA.

4 Architecture

This section introduces the SLE66 smart card family of Infineon and shows how we extended the existing architecture with our new elliptic-curve coprocessor. We designed the elliptic-curve module to optimally fit into the given architecture and to achieve maximum speed when calculating digital signatures. Very low area requirements account for low cost.

Section 4.1 presents our target architecture, the SLE66XxxxP smartcard family. Section 4.2 shows the new ECC coprocessor architecture in detail.

4.1 Target Smartcard Architecture

Figure 1 shows the block diagram of the Infineon SLE66XxxxP smartcard [19]. A multiplexed address and data bus connects various modules like memories (ROM, XRAM, NVRAM), a Random Number Generator (RNG) or the UART to the CPU. In the actual design also a RSA coprocessor called Advanced Crypto Engine(ACE) is used to accelerate cryptographic operations. However, the ACE requires much resources since it is designed to operate with key lengths of 1024 bits and beyond. With our design we target low-cost elliptic-curve applications that rely on much smaller key lengths. Typical key lengths in such a scenario are 163 or 191 bits.

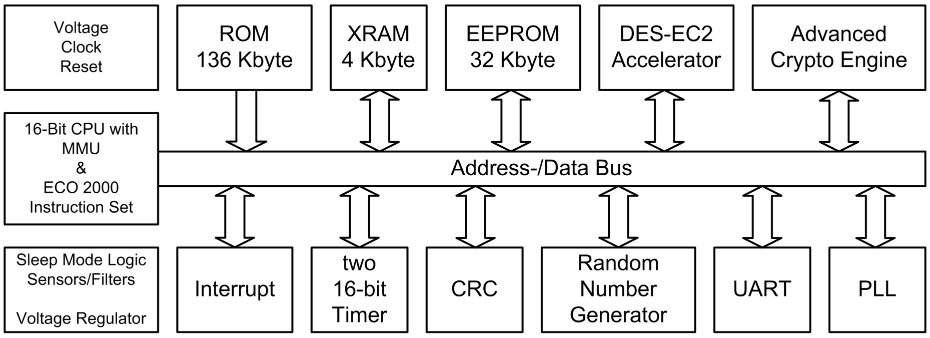

Fig. 1. Block diagram of the Infineon SLE66XxxxP smartcard family

According to this overall architecture we designed the coprocessor to communicate via the bus with the ECO 2000 CPU. The 8-bit CPU bus uses time multiplexing for address and data transport. It is able to deliver maximum 4 data bits within each clock cycle. To achieve maximum throughput we designed our architecture to process 4 bits in each clock cycle to avoid any wait states. We designed a 4-bit serial parallel multiplier to process 4 bits in each clock cycle.

To achieve low cost the new coprocessor needs to be small in terms of area. Having our new coprocessor we are able to omit the actual RSA coprocessor that can handle up to 1024 bit multiplication including registers of the same size.

The RSA coprocessor efficiently performs arithmetic operations in GF(p). The calculation of an ECDSA as it was described in section 3.4 requires efficient calculation of an inversion in GF(p). Therefore, we need to support inversion in GF(p) in our new architecture to be able to omit the actual RSA architecture. Section 4.2 shows the implementation of our architecture.

Our new architecture supports all operations required to build a low cost smartcard system based on elliptic curve cryptography. The presented architecture is very competitive in terms of area and performance.

4.2 Elliptic-Curve Coprocessor

The coprocessor integrates four basic operations: GF(2^m) addition, 4-bit digit-serial multiplication in GF(2^m), and calculation of the multiplicative inverse in GF(2^m) and GF(p). Figure 2 shows the overall system structure. The coprocessor consists of three major parts:

– Bus Decoder: The bus decoder is the interface between the multiplexed address and data bus (X-bus) of the SLE66 CPU and the coprocessor.
– Data Path: The main part of the data path consists of *leaf cells* which integrate the basic functionality of multiplication, addition, and calculation of the inverse. It also contains an adder and an up/down counter which performs GF(p) addition (adder) and is used for Berlekamp's version of the Euclidean Algorithm (counter).
– Control Logic: The control logic is a core component of the coprocessor. Its state machine generates the control signals for the data path to implement the two algorithms for GF(p) inversion and GF(2^m) inversion and sets the proper functions of the leaf cells.

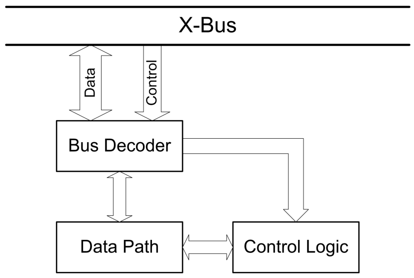

Fig. 2. Overall structure

The leaf cell (shown in Figure 3) is the main part of the data path. It is instantiated 192 times (24 slices of eight cells each, see Figure 4). The cell consists of four registers (`A` to `D`), combinational logic for achieving the necessary functionality (e.g. inversion, multiplication), and multiplexing. The grey box marks the 4-bit digit-serial multiply and reduce part. The implemented functions are as follows:

– Each register can perform a *shift-left* operation. This is essential for a fast GF(2^m) inversion.
– Register `C` can perform the *shift-right* operation necessary for the binary extended Euclidean algorithm.
– `C` and `D` can both do an *8-bit shift left* and an *8-bit shift right* which is used for loading register `C` (with bus values or addition result) and reading register `D`.

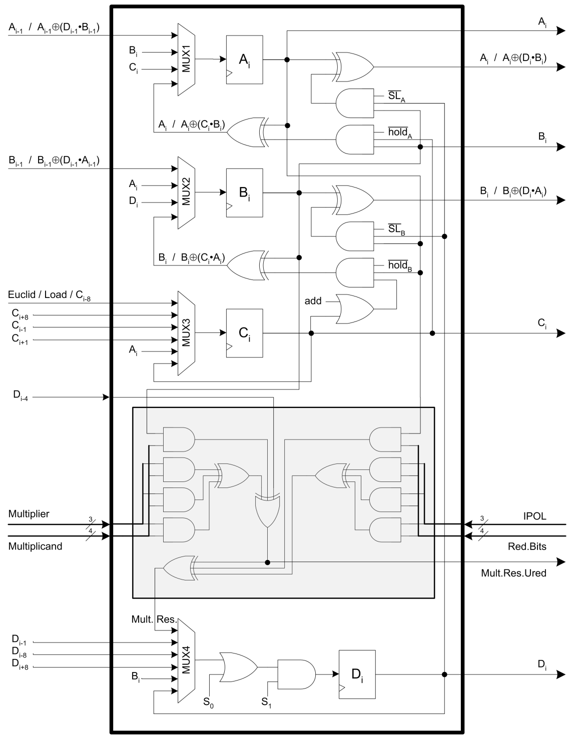

Fig. 3. Leaf cell

- Since only register `C` can be loaded and do a shift right, register contents must be distributed. So `A` can load the values of `B` or `C`, `B` can load the values of `A` or `D`, `C` the value of `A`, and `D` the value of `B`. This allows each register contents to be loaded to each other register.
- `A` and `B` store the calculation results of the GF(2^m) inversion.
- `B` is used to store the GF(2^m) addition result and
- `D` is used to store the GF(2^m) multiplication result.
- Of course, each register can *hold* its actual value.

The presented architecture is fully scalable with regard to operand length. The VHDL model of the coprocessor was carefully developed to support various operand lengths. This can be achieved by inserting additional slices to the architecture which is possible by simple parameter adjustment in the VHDL model.

As a countermeasure against side channel attacks it is possible to implement the leaf cells using a secure logic style. Such a full custom implementation of the comparably small leaf cell together with a generator tool for placement and routing of m leaf cells can be used to implement the whole architecture using a secure logic style.

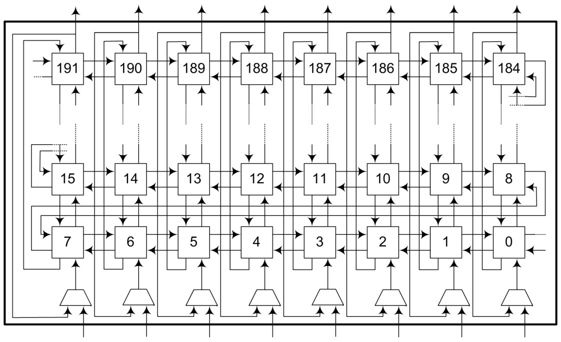

Fig. 4. Leaf cell array

5 Results

The coprocessor has been synthesized on a 0.13 µm CMOS process from Infineon. The synthesis was done with worst-case speed parameters and clock constraints of 10 MHz. The resulting chip area is 0.16 mm^2. Table 2 gives a more detailed overview of the area allocation. The values in the data path row include the leaf cell array, the adder/counter, and the bus decoder. The total size corresponds to a gate count of approximately 25,000 NAND gates. A leaf cell without support for $GF(2^m)$ inversion would have a size of 496.0 μm^2 which saves about 30% area.

Part	Area in μm^2	%
(Leaf cell	692.8	0.4)
Control unit	10,649.6	6.7
Data path	148,784.2	93.3
Total	159,433.8	100.0

Table 2. Chip area of the coprocessor

All performance results are based on the finite field $GF(2^{191})$ on a hardware implementation of 192 leaf cells (24 slices of 8 cells each). To get a reasonable performance estimation some assumptions must be made:

- The scalar multiplication has average-case characteristics (190 point doubling and 95 point addition) using the double-and-add method.
- A software overhead of 30% for scalar multiplication is added
- $GF(p)$ inversion cannot be calculated in constant time. Therefore, an average value obtained from numerous simulations is taken.
- A software overhead of 5% for $GF(p)$ inversion is added.

– Other operations needed for ECDSA calculation (besides GF(p) inversion and scalar multiplication) are not considered because they denote only a very small part of the whole algorithm.

The 'software overhead' results are based on Infineon-internal experiences. The overhead covers operations like loading operands or storing intermediate results which are necessary for an assembler implementation in the smartcard.

Operation	clock cycles
Scalar Multiplication	341,430
30% overhead	102,429
GF(p) inversion	24,310
5% overhead	1,216
Total	**469,385**

Table 3. ECDSA performance for 191 bit

Table 3 summarizes the run time of the main parts of an ECDSA calculation. A comparison with Infineon's SLE66 smartcard family shows that the coprocessor can achieve a speed-up of 4.13 compared to smartcards with the Advanced Crypto Engine (ACE) and 7.44 on smartcards without ACE.

6 Conclusion

In this article we presented a low-cost ECC smartcard coprocessor which allows a fast calculation of the Elliptic Curve Digital Signature Algorithm (ECDSA) over the finite field GF(2^m). The coprocessor supports all basic operations needed for the ECDSA. These operations are GF(2^m) addition, 4-bit digit-serial multiplication in GF(2^m) and calculation of the multiplicative inverse in GF(p) and GF(2^m). Particularly, the fast GF(2^m) inversion makes it possible to use affine instead of projective coordinates for elliptic-curve point operations. This results in a simplified control on the software level and smaller storage effort.

References

1. A. Menezes, P. Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
2. D. Hankerson, A. Menezes, S. Vanstone, *Guide to Elliptic Curve Cryptography*, ISBN 0-387-95273-X, Springer Verlag, 2004.
3. J. Großschädl, G. Kamendje, *Instruction Set Extension for Fast Elliptic Curve Cryptography over Binary Finite Fields GF(2^m)*, Application-Specific Systems, Architectures, and Processors—ASAP 2003, pp. 455–468, IEEE Computer Society Press,, 2003.

4. J. Goodman, A. P. Chandrakasan, *An Energy-efficient Reconfigurable Public-Key Cryptography Processor*, IEEE Journal of Solid-State Circuits, pp. 1808–1820, November 2001.
5. R. Schroeppel, Ch. Beaver, R. Gonzales, R. Miller, T. Draelos, *A Low-Power Design for an Elliptic Curve Digital Signature Chip*, Cryptographic Hardware and Embedded Systems—CHES 2002, LNCS 2523, pp. 366–380, Springer Verlag, Berlin, 2003.
6. S. Okada, N. Torii, K. Itoh, M. Takenaka, *A High-performance Reconfigurable Elliptic Curve Processor for GF(2^m)*, Cryptographic Hardware and Embedded Systems—CHES 2000, LNCS 1965, pp. 25–40, Springer Verlag, Berlin, 2000.
7. G. Orlando, Ch. Paar, *A High-performance Reconfigurable Elliptic Curve Processor for GF(2^m)*, Cryptographic Hardware and Embedded Systems—CHES 2000, LNCS 1965, pp. 41–56, Springer Verlag, Berlin, 2000.
8. N. Gura, S. Chang Shantz, H. Eberle, D. Finchelstein, S. Gupta, V. Gupta, D. Stebila, *An End-to-End Systems Approach to Elliptic Curve Cryptography*, Cryptographic Hardware and Embedded Systems—CHES 2002, LNCS 2523, pp. 349–365, Springer Verlag, Berlin, 2003.
9. M. Ernst, M. Jung, F. Madlener, S. Huss, R. Blümel, *A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over GF(2^m)*, Cryptographic Hardware and Embedded Systems—CHES 2002, LNCS 2523, pp. 381–399, Springer Verlag, Berlin, 2003.
10. E. Savaş, A. Tenca, Ç. Koç, *A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m)*, Cryptographic Hardware and Embedded Systems—CHES 2000, LNCS 1965, pp. 277–292, Springer Verlag, Berlin, 2000.
11. J. Großschädl, *A Bitserial Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m)*, Cryptographic Hardware and Embedded Systems—CHES 2001, LNCS 2162, pp. 206–223, Springer Verlag, 2001.
12. J. Wolkerstorfer, *Dual-Field Arithmetic Unit for GF(p) and GF(2^m)*, Cryptographic Hardware and Embedded Systems—CHES 2002, LNCS 2523, pp. 500–514, Springer Verlag, Berlin, 2003.
13. A. Gutub, A. Tenca, E. Savaş, Ç. Koç, *Scalable and Unified Hardware to Compute Montgomery Inverse in GF(p) and GF(2^m)*, Cryptographic Hardware and Embedded Systems—CHES 2002, LNCS 2523, pp. 484–499, Springer Verlag, Berlin, 2003.
14. J. Lopez and R. Dahab, *Fast Multiplication on Elliptic Curves over GF (2^m) without Precomputation*, Cryptographic Hardware and Embedded Systems—CHES 1999, LNCS 1717, pp. 316–327, Springer Verlag, 1999.
15. Victor S. Miller, *Use of Elliptic Curves in Cryptography*, LNCS 218, Springer Verlag, Berlin, 1985.
16. Neal Koblitz, *Elliptic Curve Cryptosystems*, Mathematics of Computation, Volume 48, 1987.
17. Éric Brier and Marc Joye, *Weierstraß Elliptic Curves and Side-Channel Attacks*, LNCS 2274, Springer Verlag, Berlin, 2001.
18. IEEE P1363, *Standard Specifications for Public-Key Cryptography*, IEEE standard, 2000.
19. Infineon Technologies, *Security and Chip Card ICs, SLE 66CX322P*, Product Information, 2002.
20. Elwyn R. Berlekamp, *Algebraic Coding Theory*, Aegean Park Press, revised 1984 edition, 1984.

21. Don B. Johnson, Alfred J. Menezes, and Scott Vanstone, *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, International Journal of Information Security, Volume 1, 2001.