An ECDSA Processor for RFID Authentication

Michael Hutter, Martin Feldhofer, and Thomas Plos

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

Abstract. In the last few years, a lot of research has been made to bring asymmetric cryptography on low-cost RFID tags. Many of the proposed implementations include elliptic-curve based coprocessors to provide entity-authentication services through for example identification schemes. This paper presents first results of an 192-bit Elliptic Curve Digital Signature Algorithm (ECDSA) processor that allows both entity and also message authentication by digitally signing challenges from a reader. The proposed architecture enhances the state-of-the-art in designing a low-resource ECDSA-enabled RFID hardware implementation. A tiny microcontroller is integrated to provide protocol scalability and re-use of common algorithms. The proposed processor signs a message within 859 188 clock cycles (127 ms at 6.78 MHz) and has a total chip size of 19 115 gate equivalents.

Keywords: Radio-Frequency Identification, VLSI Design, Elliptic Curves, ECDSA, Authentication, Digital Signatures.

1 Introduction

Radio-Frequency Identification (RFID) is a wireless communication technology that has gained a lot of importance in the last decade. Especially passively powered RFID tags are of major interest because they do not need a dedicated power supply. They simply draw their energy from an electromagnetic field of a reader. Furthermore, passive tags are produced in a large scale (over 3 billion tags were shipped worldwide in 2009) and can label products on the market with low costs. The hardware design of cryptographic algorithms for such RFID tags has to meet therefore low-resource requirements in terms of power and area.

One of the major challenges in the design of security-related RFID implementations is the integration of asymmetric cryptography into passive RFID tags. Asymmetric cryptography is considered to need more computational effort than symmetric cryptography but has the main advantage that no pairs of secret keys have to be maintained by tags and readers. Tags can be shipped along with a secretly kept private key whereas readers can use the corresponding public key to verify the authenticity of RFID tags. The integration of asymmetric cryptography into passive RFID tags seems therefore inevitable to allow tag authentication in open-loop systems. Indeed, the integration of public-key cryptography can effectively help to thwart the trade in counterfeiting goods of many products in the industry.

S.B. Ors Yalcin (Ed.): RFIDSec 2010, LNCS 6370, pp. 189–202, 2010.
The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-642-16822-2_16

There exist several implementation proposals for symmetric as well as asymmetric cryptography on passive RFID tags. Feldhofer et al. [7] presented a low-resource hardware implementation of the Advanced Encryption Standard (AES). They integrated AES in a challenge-response protocol to allow tag and reader authentication. Their 128-bit AES implementation needs 3 595 GE of area and is able to encrypt a challenge within 1 016 clock cycles. Many asymmetric primitives, in contrast, are based on elliptic curves due to the advantage of the smaller key sizes compared to other existing primitives like RSA. However, most of the elliptic-curve based implementations provide entity-authentication services through identification schemes and do not allow a transferable proof of the authenticity of an RFID tag. In particular, identification schemes do neither offer non-repudiation nor data-integrity services that would proof the origin of tag data. A reader that challenged a tag cannot be assured of the authenticity of the data received since identification schemes do not provide message authentication capabilities. Message authentication through digital signatures, in contrast, allows a proof of origin even at a later instant of time and thus enables many solutions for new RFID applications.

In this article, we present first results of a low-resource ECDSA hardware-implementation for RFID that provides both entity and also message authentication services. The processor is able to digitally sign the challenge of a reader by applying ECDSA using a standardized NIST $\mathbb{F}_{p192}$ elliptic curve. The design improves the state-of-the-art in implementing ECDSA for RFID applications by offering a scalable architecture using a tiny microcontroller. A digital signature can be generated within 859 188 clock cycles (i.e. 127 ms at 6.78 MHz). The total size of the ECDSA processor is 19 115 gate equivalents.

The paper is structured as follows. Section 2 gives related work on ECC processors and discusses most recent implementations. In Section 3, the tag-authentication protocol using ECDSA is described. In Section 4, the ECDSA processor is presented and details of the implemented microcontroller are given in Section 5. Results are shown in Section 6 and the conclusions are drawn in Section 7.

2 State-of-the-Art ECC Implementations

There exist many publications that present ECC hardware implementations. Only a few of them focus on low-resource ASIC designs for passive RFID devices. S. Kumar and C. Paar [18] presented a hardware implementation of an elliptic-curve coprocessor for RFID over binary fields. Also L. Batina et al. [1,2] made a lot of research on ECC implementations over binary fields and analyzed also higher-layer authentication protocols based on the Schnorr and Okamoto scheme. J. Wolkerstorfer [27] and F. Fürbass et al. [8] described an ECC processor over the recommended NIST $\mathbb{F}_{p192}$ elliptic curve that targets ECDSA signature generation for RFID tags. They reported results for point multiplication but they neither include random number generation (RNG) nor the hashing of messages to complete the signing process. An ECC processor over $\mathbb{F}_{2^{163}}$ has been proposed

by Y. K. Lee et al. [19]. The processor includes a tiny microcontroller which is able to perform the Schnorr protocol for tag authentication. The same type of elliptic curve has been investigated by D. Hein et al. [11] who implemented an ECC coprocessor over $\mathbb{F}_{2^{163}}$ that is connected to an ISO 15693-compliant RFID front-end. Similar results have also been reported by H. Bock et al. [3]. They presented an ECC processor over $\mathbb{F}_{2^{163}}$ but included a Diffie-Hellman based authentication protocol. The chip is further equipped with an ISO 15693-compliant RFID front-end, random number generator, non-volatile memory, and provides countermeasures against implementation attacks.

3 Tag Authentication using ECDSA

In the following, we give a short introduction to elliptic-curve cryptography (ECC). Afterwards, we will describe the tag authentication protocol using ECDSA.

3.1 Elliptic Curve Cryptography

Elliptic curves are algebraic structures that constitute a basic class of cryptographic primitives which rely on a mathematical hard problem. The elliptic curve discrete logarithm problem (ECDLP) is based on the intractability of deriving a large scalar after its multiplication with a given point on an elliptic curve. An elliptic curve E over a finite field $\mathbb{F}_p$ with characteristic $p > 3$ can be defined by the short Weierstrass equation $y^2 = x^3 + ax + b$, where $a, b \in \mathbb{F}_p$ are publicly-known curve parameters satisfying $4a^3 + 27b^2 \not\equiv 0 \pmod{\text{p}}$ and $x, y \in \mathbb{F}_p$ is a point on the elliptic curve. Let P be a fixed point on the curve $E(\mathbb{F}_p)$ with prime order n and k a large integer scalar in $[1, n-1]$, then it is easy to compute the scalar multiplication $Q = kP$ but hard to find k by knowing only Q and P.

In practice, the scalar multiplication can be computed by iteratively applying group operations, i.e. addition and doubling of curve points. These operations use finite-field operations such as addition, subtraction, multiplication, squaring, and inversion. There exist several formulas for addition and doubling operations that try to reduce the number of finite-field operations. Especially formulas that represent elliptic curves in projective coordinates (the affine coordinates x and y are represented by the coordinates X, Y, and Z where $x = X/Z$ and $y = Y/Z$) are often used because the costly inversion operation can be omitted during scalar multiplication. Next to projective coordinate representation there exist several proposals to improve the performance and security of the scalar multiplication. One example is the Montgomery powering ladder [16] method that can be used with x-coordinate only group formulas (thus needing only intermediate registers for the projective X and Z coordinates) and additionally provides security against Simple Power Analysis (SPA).

3.2 The ECDSA Authentication Protocol

The following section describes tag authentication using ECDSA in a challenge-response protocol which is defined in the ISO 9798-3 [13] standard. Before starting the authentication process, the reader challenges the tag to get the public-key

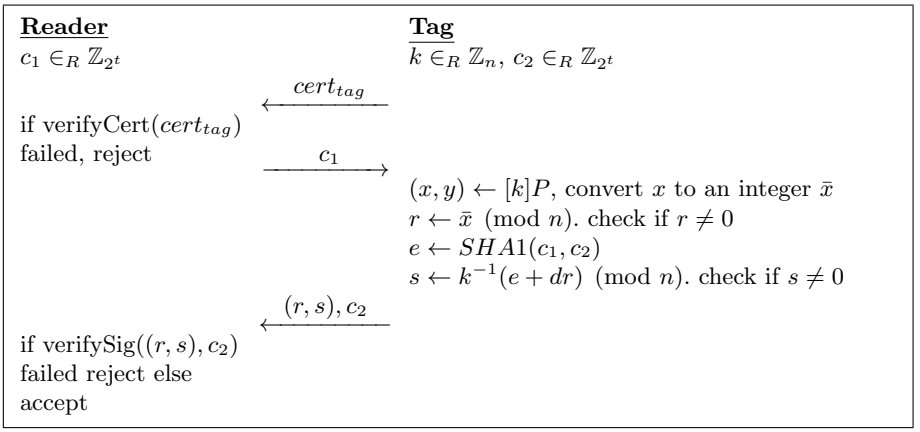


Fig. 1. Tag authentication protocol using ECDSA

certificate $cert_{tag}$. After validation of the certificate, the reader chooses a random number c_1 and sends it to the tag. After that, the tag digitally signs the challenge c_1 of the reader together with another random number c_2 using ECDSA. First, it performs a scalar multiplication using the ephemeral key k and a fixed point P on the elliptic curve. The result r is then multiplied with the private key d. Second, the hashed message e is added and the result is multiplied with the inverted ephemeral key k^{-1}. Finally, the intermediates r and s represent the digital signature which is sent together with the random value c_2 to the reader. The reader can then verify the signature and accept the tag authentication if succeeded.

4 System Architecture

The main components of the ECDSA processor are a microcontroller, a memory unit, and an arithmetic unit to perform elliptic-curve and SHA-1 operations. The reason why we have implemented a microcontroller instead of a dedicated finite state machine lies in the fact that we aimed a processor for RFID tags that is flexible and scalable in terms of different protocols and algorithm modifications. The processor should be rather modular to be re-used in different projects and already approved in practice. In fact, a microcontroller allows the writing of micro-code programs that can be easily modified and re-compiled if desired. In order to meet the performance requirements of ECC, we implemented several instruction-set extensions (ISE) that allow fast public-key arithmetic operations such as modular multiplication and inversion. In Figure 2, the architecture of an RFID tag including the proposed ECDSA processor is shown.

The ECDSA processor can be connected to an analog and digital front-end that transform analog signals into digital data. The analog front-end provides a circuit for voltage regulation, modulation, demodulation, clock extraction, communication signal pre-processing, as well as two antenna connections. The dig-

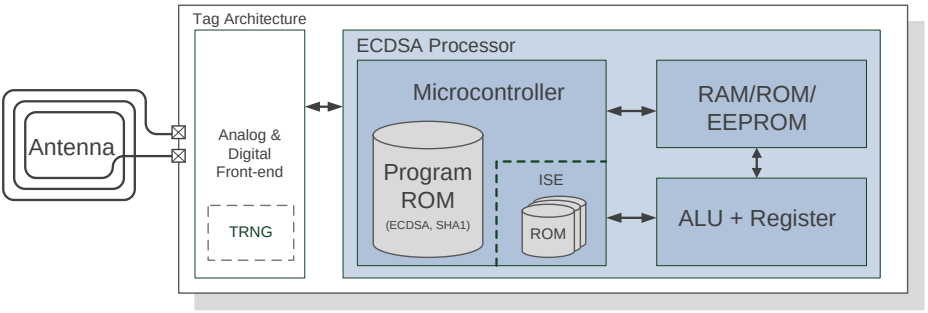


Fig. 2. Architecture of an RFID tag including the ECDSA processor

ital front-end implements the RFID protocol such as ISO 15693, ISO 14443, or ISO 18092. According to the ISO 7816-4 [14] standard, the ECDSA processor supports the INTERNAL AUTHENTICATE command that allows tag authentication using the protocol described in Section 3. A 16-byte challenge is sent from the reader that is signed by the tag. Next to that, the processor allows a direct access to the non-volatile memory to load and transmit the tag certificate, for instance. Random numbers are generated according to the FIPS 186-2 [23] standard. A random number generated from a TRNG has to be hashed using the SHA-1 algorithm and the result is used as a seed (XKEY seed) to produce any random number needed during ECDSA signature generation.

4.1 Memory Unit

The memory unit consists of a RAM, a ROM, and an EEPROM that can be accessed via a 16-bit dual-ported memory interface. One port (port A) is used to access the EEPROM, the other port (port B) is used to access the ROM table. The 128×16-bit RAM macro block can be accessed by both ports allowing the reading of data of two different addresses within one clock cycle. The first 192 bits, i.e. 12×16 bits, are reserved for the XKEY seed, 160 bits are used to store the hashed challenge, and 192 bits are needed for storing the ephemeral key. The remaining part of the RAM (7×192 bits) is used for ECDSA signing.

4.2 The 16-bit Datapath

The ECDSA datapath is shown in Figure 3 and consists of a 16×16-bit multiplier, two 40-bit adders, logic operations, several multiplexers, and one 40-bit accumulator register. According to the RAM organization, all operations are performed in words of 16 bits. The following 16-bit operations are supported: addition, subtraction, multiplication, NOT, AND, OR, and XOR. Furthermore, it allows to perform a 192-bit multiplication in a Multiply Accumulate (MAC) approach. The datapath supports only word-size operations, 192-bit finite-field operations are performed within the ISE architecture. Modular reduction has also been realized as an ISE.

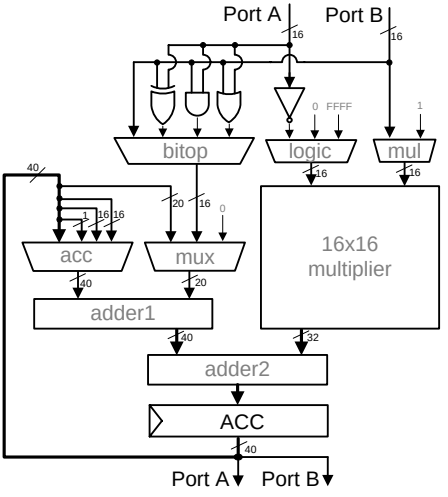


Fig. 3. The ECDSA datapath

5 The 8-bit Microcontroller

Controlling the ECDSA processor with a microcontroller provides much more flexibility than using a fixed state machine. We are using an 8-bit microcontroller with a Harvard architecture, *i.e.* program memory and data memory are separated. Such a design has the advantage that the program memory can have a different bit width than the data memory. The microcontroller is a Reduced Instruction Set Computer (RISC) supporting 32 instructions that have a width of 16 bits. The instructions are mainly divided into four groups: logical operations like XOR and OR, arithmetic operations like addition (ADD) and subtraction (SUB), control-flow operations like GOTO and CALL, and ISE operations.

Main components of the microcontroller are the ROM, the register file, the program counter, the instruction decode unit, and the ALU. The ROM contains the program memory and has a size of 600×16 bits. The register file is the data memory of the microcontroller and consists of 16 registers with a width of 8 bits each. Instructions are executed within a two-stage pipeline that consists of a fetch and a decode/execute step. In the first stage, the instruction that is addressed by the 11-bit program counter is loaded from the ROM into the instruction decode unit. In the second stage, the instruction is decoded by the instruction decode unit and executed by the ALU, followed by updating the program counter. The program counter contains a call stack that allows up to three recursive subroutine calls. All instructions are executed within a single clock cycle, except the control-flow operations and the ISE operations. Control-flow operations require two clock cycles. The number of clock cycles required for an ISE operation is not fixed and depends on the operation that is executed.

There are two types of registers in the register file of the microcontroller: special-purpose registers and general-purpose registers. The special-purpose reg-

isters involve an accumulator register for advanced data manipulation, a status register that gives information about the status of the ALU (e.g. carry bit after addition), and input/output (I/O) registers. The latter are used for accessing external devices like the memory unit or the ECDSA arithmetic unit via memory-mapped I/O. The I/O registers are also used for reacting on external events via busy waiting, since interrupts are not supported by the microcontroller. General-purpose registers are used for arbitrary data manipulations and temporarily storing data.

For implementing the microcontroller program, we have developed a self-written instruction-set simulator and assembler. Both programs are written in JAVA and allow a fast and easy way of program development. The simulator supports a single-step mode and gives access to the internal state of the microcontroller. This makes debugging and testing of the program very convenient. Moreover, optimizing the instruction set and adjusting parameters of the microcontroller like data bit width or call-stack size can be done much faster than in a direct hardware simulation.

In the following, the implemented ISE-microcode sequences are explained in a more detail and the program for ECDSA is described.

5.1 Instruction Set Extensions

The processor supports 55 instruction-set extensions for several ECDSA operations. The ISEs have been implemented in eight distinct microcode ROM tables that are able to address up to 128 microcode patterns. The partition has its reason in the fact that each table can possess a different data bit width which actually reduces the area footprint of the processor. In the following, we first describe the modular arithmetics such as addition, subtraction, multiplication, and Mersenne-like NIST prime reduction. The NIST recommended prime over $\mathbb{F}_{192}$ is $p_{192} = 2^{192} - 2^{64} - 1$. Second, we describe Montgomery-based operations such as multiplication and inversion that are mainly used for the general prime-field operations during the signing process. Third, we will focus on the ISE implementation of the message-digest calculation using SHA-1.

Modular Arithmetics. Modular addition has been realized as an ISE operation and works as follows. First, the microcontroller sets the needed address parameters to perform an addition. Second, an ISE-INST_ADD sequence is invoked using a MICRO instruction. The addition sequence needs 19 microcode patterns to add the 192-bit values a and b. Third, the microcontroller jumps to the NIST_RED subroutine which is also used for modular multiplication. Within the subroutine, another MICRO instruction is called to reduce the result. For modular reduction, we applied the fast NIST reduction method that needs 12 microcode patterns to reduce the result. The reduction sequence takes the carry ε of the addition result c and adds it to the result at bit position 0 and 63 (using the NIST prime we used the fact that 2^{192} is congruent to $2^{64} + 1 \pmod{p}$). If the carry is zero, a zero value is added accordingly. However, the performed reduction step guarantees that the result is still smaller than 2^{192} but does not guarantee the case that $2^{192} - 2^{64} - 1 \leq c < 2^{192}$. To handle this

Algorithm 1 Modular addition

Require: Modulus p, and $a, b \in [0, p-1]$.
Ensure: $c = a + b \pmod{p}$, $\varepsilon \in [0, 1]$.
1: $(\varepsilon, C[0]) \leftarrow A[0] + B[0]$.
2: **for** i **from** 1 **to** 11 **do**
3: $\quad(\varepsilon, C[i]) \leftarrow A[i] + B[i] + \varepsilon$.
4: **end for**
5: $c \leftarrow c + (2^{64} + 1) * \varepsilon$. (NIST Red.)
6: **if** $(c \geq p)$ **then**
7: $\quad c \leftarrow c - p$.
8: **end if**
9: Return (c).

Listing 1.1. Program for addition

```
...
MovLF(ADDR1_REG, 0x4);
MICRO(INST_ADD, addr_par9, 19);
CALLR("NIST_RED");
...

LABEL("NIST_RED");
  MICRO(INST_RED1, addr_null, 4);
  MICRO(INST_RED2, addr_null, 8);
  BWS(STATUS, CU_NEXT_INSTR);
  BTC(STATUS, CU_CARRY);
  MICRO(INST_SUB, addr_par14, 19);
RET();
```

case, a logical AND operation is performed on the higher eight words, i.e. the most 128 significant bits of the result. For this, we separated the sequence into the INST_RED1 and INST_RED2 instruction. The INST_RED1 instruction reduces the four least significant words of the result, the remaining eight words are handled by the INST_RED2 instruction. In particular, during the INST_RED2 instruction, an AND operation is performed on all words (i.e. 128 bits) and the resulting bit is stored in the MSB of the accumulator. If the MSB is one, the obtained result is greater than the modulus p and an extra-reduction step has to be performed, otherwise it is zero. The microcontroller tests if the MSB of the accumulator is set by reading a dedicated memory mapped I/O register bit (CU_CARRY) which is directly connected to the ECDSA datapath. A subtraction operation is called afterwards that reduces the result modulo p. Note that this extra reduction is performed in extremely rare cases since the probability of occurrence is $P(p \leq c) = \sum_{i=1}^{2^{64}} \frac{1}{2^{192}} = \frac{1}{2^{128}}$. Neglecting the execution of that extra reduction step, modular addition can be performed in $19 + 12 + 1 = 32$ clock cycles. The modular addition algorithm is shown in Algorithm 1 and the ISE invocations are shown within a code snippet of the ECDSA program in Listing 1.1.

Modular subtraction is performed similar. First, the ISE subtraction sequence is invoked by the microcontroller. It needs 19 patterns and clock cycles, respectively. Second, the microcontroller checks if there exists a borrow or not and adds the modulus p if necessary.

Modular multiplication is basically more complex than modular addition and subtraction and needs special attention in the design to obtain adequate performance. We implemented the multiplication in a Multiply Accumulate (MAC) architecture where every 16-bit word of the operands are multiplied and accumulated to the datapath register. The approach of adding partial products is similar to the method proposed by J. Grossschädl [9]. The multi-precision multiplication is done in a product scanning form (often referred as Comba multiplication method). On the algorithmic level, there exist mainly two loops to perform the

multiplication. The inner loop performs a multiplication of two 16-bit words and the outer loop assigns the sum of the partial products to the accumulator register. In order to minimize the needed memory consumption (192*2=384 bits are naturally necessary to store the multiplication result), we reduced the result during multiplication within an interleaved reduction method. Thus, no additional register is needed to obtain the result. For reducing the higher part of the result, we again used the properties of the recommended NIST prime p by simply adding the modulus at the bit positions 63 and 0. This has to be done two times to reduce the entire 192-bit number. After that, the lower 192-bits of the multiplication are computed and added to the already reduced higher parts. Note that a final reduction is necessary afterwards to reduce the carry of the final addition. The entire modular multiplication including final reduction needs 204 clock cycles. Two ISE instructions (INST_MUL1 and INST_MUL2) have been implemented in two ROM tables. For the final reduction, we re-used the NIST_RED subroutine as it has been already used for modular addition.

Montgomery Inversion and Multiplication. In order to perform inversion of the ephemeral key k in ECDSA and also to convert the projective coordinates back into affine coordinates, we implemented the inversion algorithm proposed by P. Montgomery [22]. On the one hand, this has the advantage that not only the modular inversion but also the modular multiplication (for general primes) can be computed faster than with conventional methods. On the other hand, values have to be transformed into the so-called Montgomery representation, e.g. $x \mapsto \tilde{x} = xR \bmod p$, where $R > p$ represents the Montgomery constant. Due to that reason, we implemented the Montgomery inversion algorithm according to B. Kaliski [17]. It takes an input a and outputs the inverse of a in Montgomery representation, i.e. $a^{-1}R \pmod p$. Furthermore, we implemented the Montgomery modular multiplication operation according to G. Hachez and J. J. Quisquater [10]. It takes operands which are already transformed into the Montgomery domain. Thus, no Montgomery-domain transformations have to be performed online during the ECDSA-signature generation. For Montgomery inversion, we have implemented seven ISEs, for the Montgomery multiplication there exists five ISEs.

The SHA-1 Algorithm. Basically, the SHA-1 algorithm takes a 512-bit input-message block and performs several logic operations on 6 different state variables A,B,C,D,E, and F. After each round, the state is shifted to the right. In total, 80 rounds are performed to obtain the message digest of 160 bit. For the computation, we implemented 14 ISEs: two instructions are used to initialize the state by loading the ROM constants (h0..h4 and k0..k3) into RAM, nine ISEs are used within the 80 rounds (one ISE is executed only after loop index 16, and eight ISEs are individually executed before loop index 20, 40, 60, and 80), and three ISEs are used to produce the final hash value. Furthermore, we did not rotate the content of the state but simply shifted the addresses to reach the best performance. The loop index and branching conditions have been realized in the microcontroller. 3 639 clock cycles are needed to hash a 512-bit message.

5.2 The Program for ECDSA Signature Generation

In the following, we describe the ECDSA program for signature generation. The program needs about 600 lines of code and is stored in a dedicated program ROM that is accessed by the microcontroller. Note that the program can also contain the protocol execution according to the used RFID standard. This will only add costs in the program ROM but not in all other parts. The program can be separated into the following eight main parts.

1. **Power Up.** After power up of the tag, a generated TRNG seed is used for further random number generation.
2. **Random Number Generation (RNG).** After receiving a reader challenge, the tag first performs four SHA-1 computations to generate random numbers for the ephemeral key and the applied side-channel countermeasures. The RNG is done according to the FIPS 186-2 [23] standard.
3. **Randomized Projective Coordinates (RPC).** As a side-channel countermeasure, we randomized the projective coordinates of the base point P according to the proposal of S. Coron [5]. We multiplied the X coordinate of P with a random number λ and took λ as a Z coordinate.
4. **Double the Base Point.** Instead of doubling the base point P before scalar multiplication, we pre-computed it and stored the projective coordinates (X, Z) of $Dbl(P)$ in ROM needing 24×16 bits of memory. Note that the entire scalar multiplication is performed without projective Y coordinates as described in part 6.
5. **Common-Z Coordinates.** For better scalar-multiplication performance, we raised the projective coordinates of $P = (X_0, Z_0)$ and $Dbl(P) = (X_1, Z_1)$ to a common Z coordinate by performing $X_0 \leftarrow X_0 \cdot Z_1$, $X_1 \leftarrow X_1 \cdot Z_0$, and $Z \leftarrow Z_0 \cdot Z_1$. This has the advantage that only three coordinates have to be maintained in RAM during scalar multiplication which actually can be used to reduce the number of needed registers or to increase the computation performance [20,21].
6. **ECC Scalar Multiplication.** We applied the improved Montgomery ladder proposed by T. Izu, B. Möller, and T. Takagi [15]. First, the method provides security against SPA attacks by performing the same operations in every Montgomery-ladder iteration. Second, x-coordinate only formulas have been applied according to E. Brier and M. Joye [4]. This allows to perform all computations without y coordinates. Third, the doubling and addition operations are combined to one operation which helps to reduce the computation of intermediate values that are used in both group operations. Fourth, performing doubling and addition in common Z-coordinate representation allows fast formulas for our implementation needing 12 finite-field multiplications, 4 squarings (realized as multiplications), 9 additions, and 7 subtractions for one Montgomery-ladder iteration (including common Z-coordinate transformation). Three coordinates (X_0, X_1, and Z) and four intermediate values of 192 bits have to be stored in RAM. The resulting storage requirement for the point multiplication is therefore 7×192 bits. Fifth, the Montgomery ladder holds the base point as invariant throughout the

entire point multiplication. This fact can be used to provide fault-injection countermeasures by checking the invariant during and/or after scalar multiplication. The following curve-equation check incorporates the invariant to provide such a countermeasure.

7. **Check Curve Equation.** We check if the resulting point is still on the curve after scalar multiplication. We implemented the countermeasure according to N. Ebeid and R. Lambert [6] that checks the curve equation in projective coordinates without the need of inversions[1]. However, the countermeasure includes also the recovery of the projective Y coordinate which results in 22 multiplications, 12 additions, and 7 subtractions to perform the countermeasure in our implementation. The additional overhead for including the countermeasure is 2.36 % of total chip area and 0.82 % of execution time.
8. **Final Signing Process.** The last step in our ECDSA implementation is to perform the final signing process. For this, the projective X coordinate of the scalar multiplication is transformed into affine coordinates by a multiplication with the inverted Z coordinate. After that, all operations are performed modulo the general prime n. First, the ephemeral key k is inverted. Second, the value $\bar{x}$ is tested to be zero or greater than the modulus n (a subtraction of n is performed if necessary). Third, we calculated $s = k^{-1}e + (k^{-1}r)d$ instead of $s = k^{-1}(e + dr)$. This has its reason in the fact that the fixed private key d will be multiplied by a randomized intermediate value $k^{-1}r$ which avoids first-order Differential Power Analysis (DPA) attacks targeting the intermediate values of the private key multiplication [12]. The resulting digital signature is stored in RAM and consists of the tuple (r, s).

6 Synthesis Results

The proposed ECDSA processor has been synthesized using a 0.35 μm CMOS technology (c35b4 AMS) with Cadence RTL compiler. The synthesis result includes the entire processor including microcontroller, program ROM, ISE ROM tables, address and instruction decoding, RAM macro, ROM (ECC constants), datapath (ALU + register), and an 16-bit AMBA interface. The synthesis results are shown in Table 4.

The power consumption of the processor has been simulated using Synopsys NanoSim. For the 0.35 μm CMOS technology, the simulated total mean current is 387 μA at 3.3 volt and 847 kHz. The power consumption distribution of the circuit is shown in Figure 5. The highest power consumption is needed for the RAM macro followed by the datapath, clock tree, and ISE circuit. The program ROM, the ROM for ECC constants, and the microcontroller circuit need only around 3-4 % of the total power consumption.

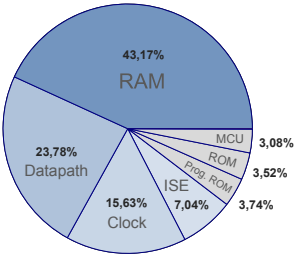
Table 1 gives a comparison with related work. ECC implementations over 192-bit prime fields have been reported by F. Fürbass et al. [8], J. Wolkerstorfer [27], and A. Satoh et al. [25]. E. Öztürk et al. [24] reported a $\mathbb{F}_p$ coprocessor over

[1] The curve-equation formula at page 3 should be $Z(Y^2 - bZ^2) = X(X^2 + aZ^2)$.

Fig. 4. Synthesis results

Component	GE
Microcontroller without program ROM	1 786
Program ROM (ECDSA, SHA1, RNG)	2 132
ISE control logic (ROM, decoder,...)	3 310
2048-bit RAM macro	8 727
ROM for ECC constants	789
Datapath (ALU+register)	2 371
Total Size	**19 115**

Fig. 5. Power consumption chart



the field $(2^{167} + 1)/3$. Their implementations differ in the supported features but need between 23 000 GE and 30 000 GE of chip area. ECC implementations over binary fields have been reported by D. Hein [11], H. Bock [3], Y. K. Lee [19], S. Kumar [18], L. Batina [2], and R. Schroeppel [26].

Table 1. Comparison with Related Work

	Area [GE]	Time [Cycles]	Field	Features
This Work	**19 115**	**859 188**	$\mathbb{F}_{p192}$	**ECDSA, SHA1, RNG**
Fürbass07 [8]	23 656	502 000	$\mathbb{F}_{p192}$	ECDSA(no SHA1,no RNG)
Wolkerstorfer05 [27]	23 800	677 000	$\mathbb{F}_{p192}$	ECC
Öztürk04 [24]	30 333	545 440	$\mathbb{F}_{(2^{167}+1)/3}$	ECC
Satoh03 [25]	29 655	4 165 000	$\mathbb{F}_{p192}$	ECC
Hein08 [11]	11 904	296 000	$\mathbb{F}_{2^{163}}$	ECC
Bock08 [3]	12 876	80 000	$\mathbb{F}_{2^{163}}$	ECC, DH, RNG
Lee08 [19]	12 506	302 457	$\mathbb{F}_{2^{163}}$	ECC, Schnorr
Kumar06 [18]	19 048	527 284	$\mathbb{F}_{2^{193}}$	ECC
Batina06 [2]	8 104	353 000	$\mathbb{F}_{2^{131}}$	ECC, without memory
Schroeppel02 [26]	191 000	93 000	$\mathbb{F}_{2^{178}}$	ECC, ElGamal, PRNG

7 Conclusions

In this article, we present results of a low-resource ECDSA processor suitable for RFID applications. The processor allows digitally signing of challenges of a reader and offers a large scale of important cryptographic services such as entity and message authentication, non-repudiation, and data integrity. Furthermore, it allows applications to perform an electronic proof of origin of RFID tags in the field. To meet the low-area requirements, we based our design on a tiny microcontroller that implements several instruction-set extensions for public-key cryptography. The total size of the processor is 19 115 GE and needs 859 188 clock cycles to digitally sign a message. The chip will be fabricated as a prototyping sample in summer 2010.

Acknowledgements. The authors would like to thank Johannes Wolkerstorfer and Marcel Medwed for their valuable inputs and discussions. This work has been supported by the Austrian Government through the research program FIT-IT Trust in IT Systems (Project CRYPTA, Project Number 820843), and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

References

1. Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., Verbauwhede, I.: Public-Key Cryptography for RFID-Tags. In: Workshop on RFID Security – RFIDSec, July 12-14, Graz, Austria. pp. 1–16 (2006)
2. Batina, L., Mentens, N., Sakiyama, K., Preneel, B., Verbauwhede, I.: Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks. In: Buttyán, L., Gligor, V., Westhoff, D. (eds.) Security and Privacy in Ad-Hoc and Sensor Networks – ESAS, Hamburg, Germany, September 20-21. vol. 4357, pp. 6–17. Springer, Heidelberg (2006)
3. 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. In: Dominikus, S. (ed.) Workshop on RFID Security – RFIDsec, Budapest, Hungary, July 9-11. pp. 1–14 (July 2008)
4. Brier, E., Joye, M.: Weierstraß Elliptic Curves and Side-Channel Attacks. In: Naccache, D., Paillier, P. (eds.) Public Key Cryptography – PKC, Paris, France, February 12-14. LNCS, vol. 2274, pp. 335–345. Springer, Heidelberg (2002)
5. Coron, J.S.: Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES, Worcester, MA, USA, August 12-13. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
6. Ebeid, N., Lambert, R.: Securing the Elliptic Curve Montgomery Ladder Against Fault Attacks. In: FDTC, Lausanne, Switzerland, September 6. pp. 46–50 (September 2009)
7. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong Authentication for RFID Systems using the AES Algorithm. In: Joye, M., Quisquater, J.J. (eds.) CHES, Cambridge, MA, USA, August 11-13. LNCS, vol. 3156, pp. 357–370. Springer, Heidelberg (August 2004)
8. Fürbass, F., Wolkerstorfer, J.: ECC Processor with Low Die Size for RFID Applications. In: International Symposium on Circuits and Systems – ISCAS, New Orleans, LA, USA, May 27-30. IEEE, IEEE (May 2007)
9. 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, Cambridge, MA, USA, August 11-13. LNCS, vol. 3156, pp. 133–147. Springer, Heidelberg (August 2004)
10. Hachez, G., Quisquater, J.J.: Montgomery Exponentiation with no Final Subtractions: Improved Results. In: Koç, Ç.K., Paar, C. (eds.) CHES, Worcester, MA, USA, August 17-18. vol. 1965, pp. 91–100. Springer, Heidelberg (August 2000)
11. Hein, D., Wolkerstorfer, J., Felber, N.: ECC is Ready for RFID – A Proof in Silicon. In: SAC, Sackville, Canada, August 14-15. pp. 401–413. LNCS (LNCS) (September 2008)
12. Hutter, M., Medwed, M., Hein, D., Wolkerstorfer, J.: Attacking ECDSA-Enabled RFID Devices. In: Abdalla, M., Pointcheval, D., Fouque, P.A., Vergnaud, D.

(eds.) ACNS, Paris-Rocquencourt, France, June 2-5. LNCS, vol. 5536, pp. 519–534. Springer, Heidelberg (May 2009)
13. International Organisation for Standardization (ISO): Information Technology - Security Techniques - Entity authentication mechanisms - Part 3: Entity authentication using a public key algorithm (1993)
14. International Organisation for Standardization (ISO): ISO/IEC 7816-4: Information technology - Identification cards - Integrated circuit(s) cards with contacts - Part 4: Interindustry commands for interchange. http://www.iso.org (1995)
15. Izu, T., Möller, B., Takagi, T.: Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks. In: Menezes, A., Sarkar, P. (eds.) INDOCRYPT, Hyderabad, India, December 16-18. LNCS, vol. 2551, pp. 296–313. Springer, Heidelberg (2002)
16. Joye, M., Yen, S.M.: The Montgomery Powering Ladder. In: Goos, G., Hartmanis, J., van Leeuwen, J. (eds.) CHES, Redwood Shores, CA, USA, August 13-15. LNCS, vol. 2523, pp. 291–302. Springer, Heidelberg (2003)
17. Kaliski, B.: The Montgomery Inverse and its Applications. IEEE Transactions on Computers 44(8), 1064–1065 (August 1995)
18. Kumar, S.S., Paar, C.: Are standards compliant Elliptic Curve Cryptosystems feasible on RFID? In: Workshop on RFID Security – RFIDSec, July 12-14, Graz, Austria (2006)
19. Lee, Y.K., Sakiyama, K., Batina, L., Verbauwhede, I.: Elliptic-Curve-Based Security Processor for RFID. IEEE Transactions on Computers 57(11), 1514–1527 (November 2008)
20. Lee, Y.K., Verbauwhede, I.: A Compact Architecture for Montgomery Elliptic Curve Scalar Multiplication Processor. In: Kim, S., Yung, M., Lee, H.W. (eds.) WISA, Jeju Island, Korea, August 27-29. LNCS, vol. 4867, pp. 115–127. Springer (2007)
21. Meloni, N.: Fast and Secure Elliptic Curve Scalar Multiplication Over Prime Fields Using Special Addition Chains. Cryptology ePrint Archive (http://eprint.iacr.org/2006/216.pdf), Report 2006/216 (2006)
22. Montgomery, P.L.: Modular Multiplication without Trial Division. Mathematics of Computation 44, 519–521 (1985)
23. National Institute of Standards and Technology (NIST): FIPS-186-2: Digital Signature Standard (DSS) (January 2000), http://www.itl.nist.gov/fipspubs/
24. Öztürk, E., Sunar, B., Savas, E.: Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In: Joye, M., Quisquater, J.J. (eds.) CHES, Cambridge, MA, USA, August 11-13. LNCS, vol. 3156, pp. 92–106. Springer, Heidelberg (August 2004)
25. Satoh, A., Takano, K.: A Scalable Dual-Field Elliptic Curve Cryptographic Processor. IEEE Transactions on Computers 52(4), 449–460 (2003)
26. Schroeppel, R., Beaver, C., Gonzales, R., Miller, R., Draelos, T.: A Low-Power Design for an Elliptic Curve Digital Signature Chip. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES, Redwood Shores, CA, USA, August 13-15. LNCS, vol. 2523, pp. 366–380. Springer, Heidelberg (2003)
27. Wolkerstorfer, J.: Is Elliptic-Curve Cryptography Suitable for Small Devices? In: Workshop on RFID and Lightweight Crypto – RFIDsec, July 13-15, 2005, Graz, Austria. pp. 78–91 (2005)