A Tiny Coprocessor for Elliptic Curve Cryptography over the 256-bit NIST Prime Field

Jeroen Bosmans, Sujoy Sinha Roy, Kimmo Järvinen, and Ingrid Verbauwhede
KU Leuven ESAT/COSIC and iMinds
Kasteelpark Arenberg 10 bus 2452, B-3001 Leuven-Heverlee, Belgium
Firstname.Lastname@esat.kuleuven.be

Abstract—**Elliptic curve cryptography (ECC) over prime fields offers a wide range of portability since the underlying arithmetic operations that are performed over integers can be supported by general purpose computing devices. This portability helps in designing various ECC based public key protocols. However implementation of a fast enough ECC in tiny electronic devices such as RFID tags, sensor nodes, smart cards etc., is a very challenging design problem since such devices are very limited in terms of resources. In this paper we design the first lightweight ECC architecture over the NIST recommended 256-bit prime field, corresponding to a medium security level of 128-bits. The ECC architecture works as a coprocessor of a 16-bit microcontroller in a memory-mapped configuration. The architecture uses an area of only 5,933 GE on a 130 nm CMOS technology, and needs roughly 6 million clock cycles to calculate a scalar multiplication.**

I. Introduction

Elliptic curve cryptography (ECC) [8], [10] is a form of public-key cryptography that offers significantly shorter keys and lower computational requirements for achieving similar security levels than competing cryptosystems (e.g., RSA). For this reason, ECC is widely used when implementing public-key cryptography in applications that are short in area, power, energy, memory, etc. Such applications include, e.g., RFID tags, sensor network nodes, and smart cards.

Significant research efforts have been made for developing efficient and secure lightweight implementations of ECC (see, e.g., [1], [3], [4], [6], [7], [13], [14], [16], [17], [20], [21], [22]). A vast majority of the research has focused on elliptic curves defined over binary fields because they utilize carry-free arithmetic which results in significantly more efficient hardware implementations. However, most software implementations utilize elliptic curves defined over prime fields because they can directly take use of integer arithmetic supported by the processors and lead to more efficient software. For this reason, prime fields are nowadays more commonly used in practice. Hence, it is essential to have lightweight hardware for prime fields in order to provide seamless integration of lightweight cryptography to existing software implementations. This has particular importance, e.g., in applications related to the Internet of Things. Most lightweight implementations also focus on approx. 160-bit elliptic curves offering roughly 80 bits of security. While there are no known attacks on these curves at the moment, it has been recommended to shift to higher security levels (e.g. 128 bits) [11].

Only few lightweight implementations of ECC over prime fields are available in the public literature. The following surveys the most relevant ones. Özturk *et al.* [13] introduced modulus scaling techniques applicable for ECC over a prime field in the development of their processor. Their design is working over a 166-bit prime field. Gaubatz *et al.* [4] studied the feasibility of public key protocols in sensor networks, but they focused on very low security levels of only about 50 bits. The applicability of ECC in RFID-identification was examined by Wolkerstorfer [22], who reported a processor working both over a binary field and over a 192-bit prime field. Sinha Roy *et al.* [17] developed a tiny ECC coprocessor working over a 160-bit prime field which covers the computation of a scalar multiplication. In [7], Kern *et al.* presented a processor that offers authentication with elliptic curves over the same field but with an additional support for the SHA-1 hash function. Pessl and Hutter [14] presented a processor that offers similar functionality (160-bit ECC and the Keccak hash function) with a significant reduction in area and computation time. Three implementations over a 192-bit field were reported by Wenger [21], Hutter *et al.* [6], and Fürbass *et al.* [3]. They are slower and have larger areas because of the larger prime.

In this paper, we present a tiny coprocessor for ECC over the standardized NIST P-256 curve [12], which is an elliptic curve defined over a 256-bit prime field that offers roughly 128-bit security level. This offers seamless integration to various applications consisting of both hardware and software components as well as to systems built for the 128-bit security level (e.g., that use AES-128 encryption). The coprocessor is designed primarily to be used as a memory-mapped coprocessor for a microcontroller so that they share the same memory (see, e.g., the drop-in concept of [20]). This design decision was made because memory is known to be a significant expense in the case of lightweight ECC [20]. Our results show that ECC over prime fields can be implemented with very small area (less than 6000 gate equivalents (GE)) even with the relatively high 128-bit security level. To the best of our knowledge, our coprocessor is the first lightweight implementation of ECC over prime fields that reaches these security levels.

Section II gives a brief overview of ECC and introduces the algorithms used in the coprocessor. Section III describes the architecture of the coprocessor. Section IV presents the results on 130 nm CMOS and compares them with the related work. Section V ends the paper with conclusions.

II. Mathematical Background

We focus on elliptic curves defined by the equation:

$$y^2 = x^3 - 3x + b \tag{1}$$

where b is a constant in a finite field $\mathbb{F}_p$. We focus on the curve NIST P-256 defined in [12], where p is a 256-bit prime specifically selected so that modular reductions are easy to compute. Points (x, y) that fulfill (1) form an additive Abelian group together with a special point $\mathcal{O}$ which acts as the zero element of the group. The group operation $P_1 + P_2$ is defined for all points of the group and it returns a third point which is also in the group. The group operation is called point addition when $P_1 \neq \pm P_2$ and point doubling when $P_1 = P_2$. The fundamental operation in every ECC protocol is scalar multiplication $Q = kP$ where a point of the group is multiplied by a scalar k, i.e., P is added to itself k times. The security of ECC is based on the difficulty of computing the elliptic curve discrete logarithm problem which is the inverse of scalar multiplication: given Q and P find k. This is believed to be computationally impossible to solve if the parameters are chosen correctly (e.g., as defined in [12]).

Cryptographic protocols based on ECC can be visualized as a hierarchical pyramid (shown in Fig. 1). On top of the pyramid, there are protocols, e.g. for key exchange, authentication, digital signatures, etc. The highest level that we consider in this paper is scalar multiplication and it decomposes into point additions and point doublings. They are in turn computed with series of field operations. In the following, we provide brief descriptions of the algorithms that are used in the coprocessor.

A. Field Operations

The processor implements four primitive field operations: modular addition/subtraction/multiplication, and field inversion. The algorithms assume that we have a W-bit datapath and that the elements are split into $t = \lceil \log_2(p)/W \rceil$ words. Such algorithms are called multiprecision algorithms and we discuss them below. The algorithms are generic but, in our coprocessor, we have $\log_2(p) = 256$, $W = 16$, and $t = 16$.

1) Addition/Subtraction: Addition and subtraction are very similar and we only give the algorithm for the addition in Alg. 1. The algorithm is an adapted version of the algorithm from [9]. The first loop of the algorithm implements the multiprecision addition. The carry bit is denoted by ϵ. If $c \geq p$, the prime p should be subtracted once to get the result in the finite field. This subtraction is implemented by the second loop. We compute both possible results ($a + b$ and $a + b - p$) and, afterwards, decide which of them is correct based on the 1-bit values ϵ_1 and ϵ_2. Hence, Alg. 1 executes in constant time where as the algorithm in [9] subtracts p only if necessary. Our approach also does not need a big comparator for $c \geq p$.

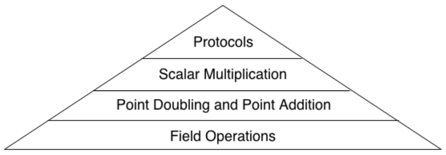

Fig. 1. Hierarchy of operations in ECC systems

Algorithm 1 Addition in $\mathbb{F}_p$

```
Input: $a[0], \ldots, a[t-1]$, $b[0], \ldots, b[t-1]$, $p[0], \ldots, p[t-1]$
Output: $c[0], \ldots, c[t-1]$ such that $c = a + b \mod p$
  $(\epsilon, c[0]) \leftarrow a[0] + b[0]$
  for $i$ from $1$ to $t-1$ do
      $(\epsilon, c[i]) \leftarrow a[i] + b[i] + \epsilon$
  $\epsilon_1 \leftarrow \epsilon$
  $(\epsilon, c'[0]) \leftarrow c[0] - p[0]$
  for $i$ from $1$ to $t-1$ do
      $(\epsilon, c'[i]) \leftarrow c[i] - p[i] - \epsilon$
  $\epsilon_2 \leftarrow \epsilon$
  if $\epsilon_1 = 1$ or $\epsilon_2 = 0$ then
      $c \leftarrow c'$
  return $c$
```

2) Multiplication: Multiplication in $\mathbb{F}_p$ is a critical operation in scalar multiplications. It is performed in two steps: first, an integer multiplication and, second, a modular reduction. We perform the integer multiplication by using the the famous product-scanning algorithm. We directly use the version available in [5] and, hence, we omit more detailed description here. The result of the multiplication will be twice the size of the operands (at most 512 bits).

Algorithm 2 Modular reduction for the NIST P-256 prime p

```
Input: $c[0] \ldots c[2 \cdot t - 1]$
Output: $d[0] \ldots d[t-1]$ such that $d = c \bmod p$
  $s_1 \leftarrow (c[15], \ldots, c[0])$
  $s_2 \leftarrow (c[31], \ldots, c[22], 0, \ldots, 0)$
  $s_3 \leftarrow (0, 0, c[31], \ldots, c[24], 0, \ldots, 0)$
  $s_4 \leftarrow (c[31], \ldots, c[28], 0, \ldots, 0, c[21], \ldots, c[16])$
  $s_5 \leftarrow (c[17], c[16], c[27], c[26], c[31], \ldots, c[26], c[23], \ldots, c[18])$
  $s_6 \leftarrow (c[21], c[20], c[17], c[16], 0, \ldots, 0, c[27], \ldots, c[22])$
  $s_7 \leftarrow (c[23], c[22], c[19], c[18], 0, \ldots, 0, c[31], \ldots, c[24])$
  $s_8 \leftarrow (c[25], c[24], 0, 0, c[21], \ldots, c[16], c[31], \ldots, c[26])$
  $s_9 \leftarrow (c[27], c[26], 0, 0, c[23], \ldots, c[18], 0, 0, c[31], \ldots, c[28])$
  $\epsilon \leftarrow 0$
  for $i$ from $0$ to $t-1$ do
      $(\epsilon, t_1[i]) \leftarrow s_1[i] + s_2[i] + s_2[i] + s_3[i] + s_3[i] + s_4[i] + s_5[i] + \epsilon$
  for $i$ from $\epsilon$ to $1$ do
      $(\epsilon_1, t_1) \leftarrow t_1 - p$
  $(\epsilon_1, t'_1) \leftarrow t_1 - p$
  if $\epsilon_1 = 0$ then
      $t_1 \leftarrow t'_1$
  $\epsilon \leftarrow 0$
  for $i$ from $0$ to $t-1$ do
      $(\epsilon, t_2[i]) \leftarrow s_6[i] + s_7[i] + s_8[i] + s_9[i] + \epsilon$
  for i from $\epsilon$ to $1$ do
      $(\epsilon_1, t_2) \leftarrow t_2 - p$
  $(\epsilon_1, t'_2) \leftarrow t_2 - p$
  if $\epsilon_1 = 0$ then
      $t_2 \leftarrow t'_2$
  $d \leftarrow (t_1 - t_2) \bmod p$
  return $d$
```

In general, the modular reduction of the result is a very costly operation. However, the prime p used by the NIST P-256 curve is a generalized Mersenne prime [18] which allows efficient reductions involving only additions and subtractions. We base our reduction algorithm on the ones available in [5],

[12] and it is given in Alg. 2. It also includes steps like $(\epsilon, c) \leftarrow a - b$, where a, b, c are 256-bit numbers.

3) Inversion: Divisions in $\mathbb{F}_p$ are performed by inverting the divisor and multiplying the result by the dividend. We base our inversion algorithm on Fermat's Little Theorem which states that $a^{p-1} = 1$ holds for all a in $\mathbb{F}_p$. Hence, $a^{-1} = a^{p-2}$ and we can compute the inverse with an exponentiation to a fixed exponent $p - 2$. Alg. 3 shows our inversion algorithm tailored for the NIST P-256 prime, which is based on the square-and-multiply exponentiation.

Algorithm 3 Inversion for the NIST P-256 prime p

```
Input: $a[0], \dots, a[t-1]$
Output: $b[0], \dots, b[t-1]$ such that $b = a^{-1}$
  $x \leftarrow a$; $b \leftarrow a$
  for $i$ from 0 to 255 do
    $x \leftarrow x \cdot x$
    if $2 \le i \le 95$ or $i = 192$ or $224 \le i \le 255$ then
      $b \leftarrow b \cdot x$
  return $b$
```

B. Point Operations

The next level of the hierarchy of Fig. 1 is the point operations. We present our algorithms for point addition and point doubling below. These algorithms use Jacobian coordinates in order to avoid expensive inversions by representing a point with three coordinates (X, Y, Z) (see, e.g., [5]). An affine point (x, y) is mapped to Jacobian coordinates by setting $(x, y, 1)$. The mapping back to affine coordinates is performed by computing $(X/Z^2, Y/Z^3)$ which involves an inversion.

1) Point Doubling: Point doubling computes the point $2 \cdot P$ so that the input and output points are in Jacobian coordinates. We base our point doubling algorithm on the algorithm provided in [5]. The algorithm is adapted so that it uses only the primitive field operations discussed in Section II-A. We also optimized the algorithm so that the number of intermediate variables is reduced. This is particularly important for lightweight implementations because it leads to smaller RAM requirements. The point doubling algorithm is given in Alg. 4.

2) Point Addition: For point addition, we use the mixed coordinate point addition algorithm from [5], where the first point is in Jacobian coordinates and the second point is in affine coordinates. The result is given in Jacobian coordinates. Also this algorithm is modified so that it uses only the primitive field operations and fewer temporary variables. The resulting algorithm is given as Alg. 5. The point doubling in Alg. 5 is computed by using Alg. 4.

3) Scalar Multiplication: In this paper, we use the basic double-and-add algorithm for computing scalar multiplications (see, e.g., [5]). The algorithm is given Alg. 6. It computes a point doubling for each k_i of the scalar and a point addition if $k_i = 1$. The algorithm ends with a conversion back to the affine coordinates. We emphasize that this algorithm can be changed by replacing the control logic. For instance, one can use Montgomery's ladder with regular pattern of operations for added security against side-channel attacks or windowing algorithms with precomputations for increased performance.

Algorithm 4 Point Doubling

```
Input: $P = (X_1, Y_1, Z_1)$ in Jacobian coordinates
Output: $2 \cdot P = (X_3, Y_3, Z_3)$ in Jacobian coordinates
  if $P = \mathcal{O}$ then return $\mathcal{O}$
  1. $T_1 \leftarrow Z_1 \cdot Z_1$        11. $T_3 \leftarrow T_4 + T_4$
  2. $T_2 \leftarrow X_1 - T_1$        12. $T_3 \leftarrow T_3 \cdot X_1$
  3. $T_1 \leftarrow X_1 + T_1$        13. $T_4 \leftarrow T_4 \cdot T_4$
  4. $T_2 \leftarrow T_2 \cdot T_1$        14. $T_4 \leftarrow T_4 + T_4$
  5. $T_3 \leftarrow T_2 + T_2$        15. $X_3 \leftarrow T_2 \cdot T_2$
  6. $T_2 \leftarrow T_2 + T_3$        16. $T_1 \leftarrow T_3 + T_3$
  7. $T_4 \leftarrow Y_1 + Y_1$        17. $X_3 \leftarrow X_3 - T_1$
  8. $Z_3 \leftarrow T_4 \cdot Z_1$        18. $T_1 \leftarrow T_3 - X_3$
  9. $T_4 \leftarrow Y_1 \cdot Y_1$        19. $T_1 \leftarrow T_1 \cdot T_2$
  10. $T_4 \leftarrow T_4 + T_4$       20. $Y_3 \leftarrow T_1 - T_4$
  return $(X_3, Y_3, Z_3)$
```

Algorithm 5 Point Addition

```
Input: $Q = (X_1, Y_1, Z_1)$ in Jacobian coordinates, $P = (x_2, y_2)$ in affine
    coordinates.
Output: $P + Q = (X_3, Y_3, Z_3)$ in Jacobian coordinates
  if $Q = \mathcal{O}$ then return $(X_1, Y_1, Z_1)$
  if $P = \mathcal{O}$ then return $(x_2, y_2, 1)$
  1. $T_1 \leftarrow Z_1 \cdot Z_1$        4. $T_2 \leftarrow T_2 \cdot y_2$
  2. $T_2 \leftarrow T_1 \cdot Z_1$        5. $T_1 \leftarrow T_1 + X_1$
  3. $T_1 \leftarrow T_1 \cdot x_2$        6. $T_2 \leftarrow T_2 - Y_1$
  if $T_1 = 0$ then
    if $T_2 = 0$ then return $2 \cdot (x_2, y_2, 1)$
    else return $\mathcal{O}$
  7. $Z_3 \leftarrow Z_1 + T_1$        13. $X_3 \leftarrow X_3 - T_1$
  8. $T_3 \leftarrow T_1 \cdot T_1$        14. $X_3 \leftarrow X_3 - T_4$
  9. $T_4 \leftarrow T_3 \cdot T_1$        15. $T_3 \leftarrow T_3 - X_3$
  10. $T_3 \leftarrow T_3 \cdot X_1$       16. $T_3 \leftarrow T_3 \cdot T_2$
  11. $T_1 \leftarrow T_3 + T_3$       17. $T_4 \leftarrow T_4 \cdot Y_1$
  12. $X_3 \leftarrow T_2 \cdot T_2$       18. $Y_3 \leftarrow T_3 - T_4$
  return $(X_3, Y_3, Z_3)$
```

III. Architecture

In this section we describe the hardware architecture of our lightweight prime field ECC coprocessor. The ECC coprocessor is memory mapped [15] with a 16-bit microcontroller following the drop-in concept of [16], [20]. In this configuration the RAM of the microcontroller is used to store the field elements during an ECC operation. We choose 16-bit microcontrollers such as TI MSP430F241x or MSP430F261x [19]. These families of microcontrollers are ideal for resource constrained applications since they are low-power, have at least 4KB of RAM, and can run at 16 MHz clock frequency. The microcontroller loads the input data into specific addresses of the RAM and then instructs the ECC coprocessor to compute a scalar multiplication. During this ECC operation, the shared RAM is controlled by the ECC coprocessor. The top level interface between the microcontroller, ECC coprocessor and the shared RAM is shown in Fig. 2. The ECC coprocessor, which we design in this work, is composed of four main blocks: the ALU, the instruction decoder, the RAM address unit and the controller. The internal architecture of the ECC coprocessor is shown in Fig. 3.

The Shared RAM stores field elements in 16-bit words. For the double-and-add scalar multiplication algorithm we need to store 16 field elements, each consuming 16 words. Thus in total 512 bytes of RAM is used during an ECC operation. To facilitate the access of data from the shared RAM, we segment

Algorithm 6 Scalar Multiplication

```
Input: Scalar $k = (k_{t-1}, k_{t-2}, ..., k_1, k_0)_2$, point $P$ on the elliptic curve
    in affine coordinates
Output: $Q = k \cdot P$ in affine coordinates
  $Q \leftarrow \mathcal{O}$
  for $i$ from $t-1$ downto $0$ do
      $Q \leftarrow 2 \cdot Q$
      if $k_i = 1$ then
          $Q \leftarrow Q + P$
  $Q \leftarrow$convert_Jacobian_affine($Q$)
  return $Q$
```

the RAM into 16 macro-slots. The address of a macro-slot is referred to as the *virtual address*. A macro-slot consists of 16 words, and a particular word from a macro-slot is accessed using the address-pointer *offset address*. The *Address Unit* of the ECC coprocessor concatenates the *virtual address* and the *offset* signals to an 8-bit *physical address* signal.

A. The arithmetic and Logic Unit (ALU)

The internal architecture of the ALU is shown in Fig. 4. It is the central part of the ECC coprocessor and is responsible for the prime field arithmetic operations. The architecture of ALU is designed to utilize a minimum amount of area. This area optimization is achieved by performing resource sharing as much as possible and by using a 16-bit data path. This particular width of the data path also nicely fits with the shared RAM which also has 16-bit words.

The centre of the ALU is the 16×16 integer multiplier which is used in almost every clock cycle. The input words to the multiplier are kept in two registers S_1 and S_2. These two registers are connected to the data-output of the shared RAM. The 32-bit result of the multiplier is split into two chunks of 16 bits. These two chunks go to an adder and an adder-subtracter circuit, where they can be added (or subtracted) to the previous values stored in the accumulator registers R_0 and R_1. The outputs from the adder and the adder-subtracter circuits are stored in the two accumulator registers.

B. The Controller

The controller block generates the control signals for the decoder and the address unit. This block is composed of a hierarchy of finite state machines (FSMs). The decoder receives commands from the controller block and then decodes them

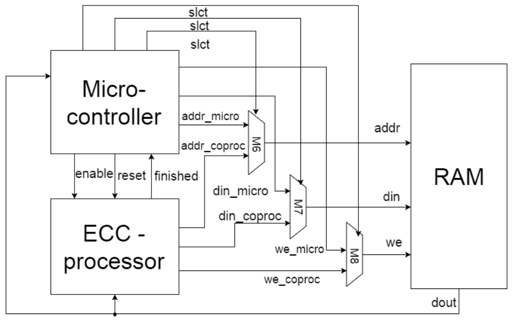

Fig. 2. Top level architecture

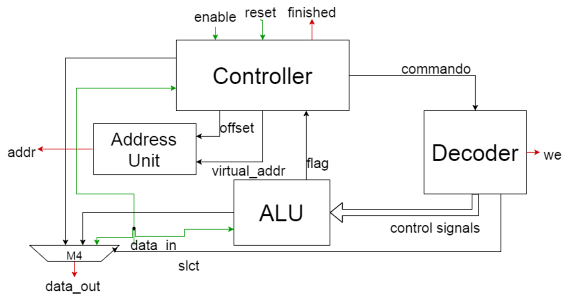

Fig. 3. ECC coprocessor architecture

into control signals for the ALU. These control signals (the green arrows in Fig. 4) control the inputs to the multiplexers and to write or clear the internal registers of the ALU. The hierarchy of FSMs in the controller block is divided into two levels: the low-level controller (L1 controller) and the high-level controller (L2 controller).

The L1 controller is in charge of the primitive field operations. It receives instructions from the L2 controller and generates control signals for the data path. During a field operation the L1 controller sends a *busy* signal to the L2 controller. The L2 controller is initiated by the microcontroller to perform the point operations. Below we describe how the FSMs in the controllers perform various tasks.

- *Field addition* implements Alg. 1. It adds two field elements by sequentially accumulating the words. The ith word of the first element is fetched from the RAM and then stored in the register S_2. Next, the word is multiplied by one (using the multiplexer M_1) and added with zero (using the multiplexer M_2) using the adder-subtracter circuit and finally stored in the accumulation register R_0. Now the ith word of the second field element is fetched from the RAM, then stored in S_2, again multiplied with 1, and brought to the input of the adder-subtracter circuit to be added with the content of R_0, which contains the ith word of the first element. The sum is stored in R_0 and the

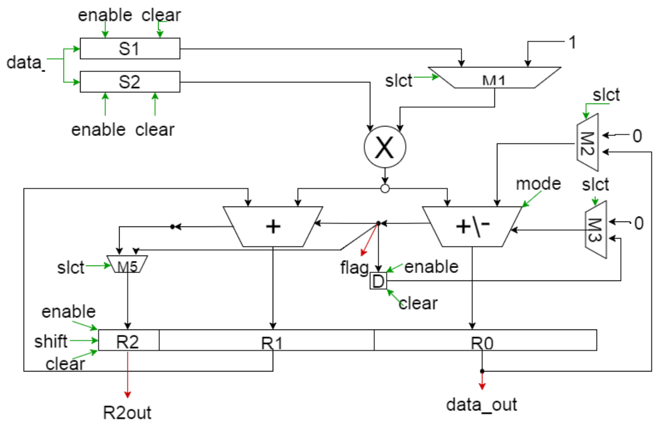

Fig. 4. The ALU block

carry is stored in the carry register D. The value stored in R_0 is written back in RAM in the next cycle. The carry register is taken into account during the addition of the $(i+1)$th words. After a multi-precision addition of two field elements, a multi-precision subtraction of the prime modulus from the result is performed. The result of modular field addition is determined based on the sign of this subtraction operation.

- *Field subtraction* performs subtraction of two field elements similarly to the field addition.
- *Field multiplication* first computes an integer multiplication of the two field elements and then performs a modular reduction by using Alg. 2. For the integer multiplication, words of the operands are serially multiplied and then accumulated using the product-scanning multiplication algorithm [5]. The result of the integer multiplication is stored in the RAM and is later processed during the modular reduction step. The modular reduction operation performs multi-precision integer additions and subtractions. This field multiplication FSM is also responsible for performing a field squaring operation. To save area, we have not kept a separate FSM for performing field squaring. Moreover an optimized field squaring over a prime field is only slightly faster than a field multiplication. The steps of the modular reduction are hardcoded in the FSM routine.
- *Field inversion* implements Alg. 3. It uses the field multiplication FSM for squarings and multiplications.

The L2 controller executes the point operations such as point addition and point doubling in a sequence. The steps of these operations (shown in Algs. 4 and 5) are stored as instructions in a ROM. During a scalar multiplication, a counter is used as a pointer to the instruction sequence.

IV. Results and comparison with related work

We described the ECC coprocessor using VHDL and verified its correct behavioral and post-route functionality using the Xilinx iSim simulation tool. We obtained the ASIC results after synthesizing the final code using the 'regular compile' function of the Synopsys Design Compiler D-2010.03-SP4 for UMC 0.13 μm CMOS with a voltage of 1.2 V, using the 'Faraday FSC0L low-leakage standard cell libraries'. Our ECC coprocessor architecture consumes a total area of only 5,933 GE. This gate count includes everything shown in Fig. 3 but excludes the area of the shared RAM which belongs to the microcontroller. The ECC coprocessor uses the RAM only during an ECC scalar multiplication. If the area of the RAM is taken into account (e.g. standalone ECC processor), then it would cost additional 6,000 GE following the memory-compiler generated area reports in [20]. The ECC coprocessor computes a scalar multiplication in nearly 6.2 million cycles and thus spends nearly 386 milliseconds at 16 MHz clock frequency. This computation time is reasonably sufficient for many applications that run on resource constrained platforms.

In Table I we compare the area of our ECC coprocessor architecture with other reported architectures over prime fields.

TABLE I
Comparison with reported lightweight prime field ECC implementations in terms of area

Ref.	Field	Func.	Area (GE)	Techn. (nm)
[4], 2005	$\mathbb{F}_{p_{100}}$	ECMV	18 720	130
[13], 2004	$\mathbb{F}_{p_{166}}$	ECSM	30 333	130
[7], 2010	$\mathbb{F}_{p_{160}}$	ECDSA, SHA-1	18 247	350
[17], 2013	$\mathbb{F}_{p_{160}}$	ECSM	26 000	32
[14], 2014	$\mathbb{F}_{p_{160}}$	ECDSA, Keccak	12 448	130
[22], 2005	$\mathbb{F}_{p_{192}}$	ECDSA	23 000	350
[3], 2007	$\mathbb{F}_{p_{192}}$	ECDSA, SHA-1	23 656	350
[6], 2010	$\mathbb{F}_{p_{192}}$	ECDSA, SHA-1	19 115	350
[21], 2011	$\mathbb{F}_{p_{192}}$	ECDSA,SHA-1	14 644	130
This work, 2015	$\mathbb{F}_{p_{256}}$	ECSM	5 933 + 256×16 RAM	130

*The 256 × 16-bit RAM is estimated to have an area of 5794 GE [20].

Since our architecture is the first one in the category of lightweight implementations for 256-bit prime fields, it is relatively hard to do fair comparison with the lightweight ECC architectures in Table I. Moreover several of these architectures report area costs of ECC based protocols. In the functionality column of the table, ECSM stands for Elliptic Curve Scalar Multiplication, ECDSA for Elliptic Curve Digital Signature Algorithm, ECMV is an elliptic curve key transport protocol, and SHA-1 and Keccak are two cryptographic hashing functions. Nevertheless, from the area reports in the table we see that even for a higher security prime field we can design an ECC architecture in a very small area footprint when we use a 16-bit data path and use the ECC architecture as a coprocessor of a microcontroller.

In Table II we compare the timing results of our ECC architecture with other lightweight prime field architectures. Our architecture requires roughly six million cycles which is larger by an order in comparison to the other reported cycle counts. This larger cycle originates from the greater security provided by our architecture. Where the other processors guarantee security of 50-90 bits, our processor offers security of 128 bits. The computational complexity of an elliptic curve scalar multiplication grows faster than a quadratic order of the bit security. A much fairer comparison is done when we consider a recent work by Sinha Roy, Järvinen and Verbauwhede [16], where a similar 16-bit memory-mapped ECC coprocessor is designed over a 283-bit NIST recommended binary field. This architecture consumes roughly 4,300 GE (excluding the shared RAM) and spends roughly 1.6 million cycles to perform a scalar multiplication. Our prime field architecture is slightly larger and obviously slower due to the more complicated prime field arithmetic which involves carry propagation. This extra cost of prime field arithmetic is compensated by its support for a wide range of platforms.

The benefit of our lightweight hardware architecture becomes more visible when compared with software implementations of ECC with similar security levels on 16-bit architectures. The most recent software implementation by Düll *et al.* in [2] performs a scalar multiplication on a specially optimized 256-bit prime curve Curve25519 in nearly 9.1 million cycles

TABLE II
COMPARISON WITH REPORTED LIGHTWEIGHT PRIME FIELD ECC IMPLEMENTATIONS IN TERMS OF COMPUTATION TIME

Ref.	Field	# cycles	Freq. (kHz)	Timing (ms)
[4], 2005	$\mathbb{F}_{p_{100}}$	205 250	500	410
[13], 2004	$\mathbb{F}_{p_{166}}$	638 000	20 000	32
[7], 2010	$\mathbb{F}_{p_{160}}$	511 864	1 000	512
[17], 2013	$\mathbb{F}_{p_{160}}$	250 000	1 000	250
[14], 2014	$\mathbb{F}_{p_{160}}$	139 930	1 000	140
[22], 2005	$\mathbb{F}_{p_{192}}$	458 950	68 500	6.7
[3],2007	$\mathbb{F}_{p_{192}}$	500 000	83 333	6
[6], 2010	$\mathbb{F}_{p_{192}}$	859 188	6 780	127
[21], 2011	$\mathbb{F}_{p_{192}}$	394 000	1 695	232
This work, 2015	$\mathbb{F}_{p_{256}}$	6 180 856	16 000	386

at 16 MHz clock frequency (which is also the frequency we use). This curve allows more efficient arithmetic than the NIST P-256 curve we use in our architecture. We preferred to use the NIST P-256 curve since it is used in several standards. Nevertheless our architecture is almost 1.5 times faster than the software implementation of the faster curve.

V. CONCLUSIONS AND FUTURE WORK

In this paper we showed that a lightweight ECC coprocessor architecture over a 256-bit NIST recommended prime field, providing 128 bit security is feasible. We even achieved the smallest area footprint in comparison to other reported prime field ECC architectures. Such a low area footprint was achieved by restricting the width of the data path to 16 bits and by instantiating the architecture as a coprocessor of a 16-bit microcontroller. The designed ECC coprocessor performs a scalar multiplication operation in 386 ms which is fast enough for most resource constrained applications.

Our target was to check the feasibility of a lightweight ECC processor over a prime field that is large enough to provide 128-bit security. In addition to an efficient and lightweight implementation, security against side channel attacks is also very important for an ECC architecture. In the presented architecture we did not implement countermeasures against side channel attacks. For example, the double-and-add scalar multiplication algorithm is not secure against side channel attacks since the bits of scalar can be derived by observing the instantaneous power consumption of the processor. We can solve this problem by executing a balanced scalar multiplication algorithm. Security against differential power attacks can be obtained by introducing randomness in the scalar multiplication. In the future we will integrate these countermeasures by modifying the microcode of the ECC architecture that is responsible for executing the scalar multiplication.

ACKNOWLEDGMENTS

S. Sinha Roy was supported by the Erasmus Mundus PhD Scholarship and K. Järvinen was funded by FWO Pegasus Marie Curie Fellowship. This work was supported by the Research Council KU Leuven: TENSE (GOA/11/007), by iMinds, by the Flemish Government, FWO G.0550.12N, G.00130.13N and FWO G.0876.14N, and by the Hercules Foundation AKUL/11/19.

REFERENCES

[1] L. Batina, J. Guajardo, T. Kerins, N. Mentens, and P. Tuyls. Public-key cryptography for RFID-tags. In *Proc. Int. Workshop on Pervasive Computing and Communication Security (PerSec 2007)*, pages 217–222. IEEE Computer Society Press, 2007.

[2] M. Düll, B. Haase, G. Hinterwälder, M. Hutter, C. Paar, A. H. Snchez, and P. Schwabe. High-speed curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. Cryptology ePrint Archive, Report 2015/343, 2015.

[3] F. Furbass and J. Wolkerstorfer. ECC processor with low die size for RFID applications. In *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS 2007)*, pages 1835–1838, May 2007.

[4] G. Gaubatz, J.-P. Kaps, E. Öztürk, and B. Sunar. State of the art in ultra-low power public key cryptography for wireless sensor networks. In *Proc. IEEE Int. Workshop on Pervasive Computing and Communication Security (PerSec 2005)*, pages 146–150, 2005.

[5] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2003.

[6] M. Hutter, M. Feldhofer, and T. Plos. An ECDSA processor for RFID authentication. In *Proc. Workshop on RFID Security (RFIDSec 2010)*, LNCS 6370, pages 189–202. Springer, 2010.

[7] T. Kern and M. Feldhofer. Low-resource ECDSA implementation for passive RFID tags. In *Proc. 17th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS'10)*, pages 1236–1239. IEEE, 2010.

[8] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[9] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1st edition, 1996.

[10] V. S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology (CRYPTO '85)*, LNCS 218, pages 417–426. Springer, 1986.

[11] National Institute of Standards and Technology (NIST). Recommendation for key management — part 1: General (revision 3). NIST Special Publication 800-57, 2007.

[12] National Institute of Standards and Technology (NIST). Digital signature standard (DSS). Federal Information Processing Standards Publication FIPS PUB 186-4, 2013.

[13] E. Öztürk and B. Sunar. Low-power elliptic curve cryptography using scaled modular arithmetic. In *Cryptographic Hardware in Embedded Systems (CHES 2004)*, LNCS 3156, pages 92–106. Springer, 2004.

[14] P. Pessl and M. Hutter. Curved tags — a low-resource ECDSA implementation tailored for RFID. In *Proc. Workshop on RFID Security (RFIDsec 2014)*, LNCS 8651, pages 156–172. Springer, 2014.

[15] P. R. Schaumont. *A Practical Introduction to Hardware/Software Codesign*. Springer, 1st edition, 2010.

[16] S. Sinha Roy, K. Järvinen, and I. Verbauwhede. Lightweight coprocessor for Koblitz curves: 283-bit ECC including scalar conversion with only 4300 gates. In *Cryptographic Hardware and Embedded Systems (CHES 2015)*, LNCS 9293, pages 102–122. Springer, 2015.

[17] S. Sinha Roy, B. Yang, V. Rozic, N. Mentens, J. Fan, and I. Verbauwhede. Designing tiny ECC processor. Presentation at the 17th Workshop on Elliptic Curve Cryptography, 2013. url: https://www.cosic.esat.kuleuven.be/ecc2013/files/sujoy.pdf (accessed Oct. 22, 2015).

[18] J. A. Solinas. Generalized Mersenne numbers. Technical Report CORR 1999/39, University of Waterloo, Combinatorics and Optimization, 1999.

[19] Texas Instruments. MSP430F261x and MSP430F241x, Jun. 2007, Rev. Nov. 2012. url: http://www.ti.com/lit/ds/symlink/msp430f2618.pdf (accessed Oct. 22, 2015).

[20] E. Wenger. Hardware architectures for MSP430-based wireless sensor nodes performing elliptic curve cryptography. In *Proc. Int. Conf. on Applied Cryptography and Network Security (ACNS 2013)*, LNCS 7954, pages 290–306. Springer, 2013.

[21] E. Wenger, M. Feldhofer, and N. Felber. Low-resource hardware design of an elliptic curve processor for contactless devices. In *Proc. Int. Workshop on Information Security Applications (WISA 2010)*, LNCS 6513, pages 92–106. Springer, 2010.

[22] J. Wolkerstorfer. Scaling ECC hardware to a minimum. In *Proc. Austrochip 2005 Mikroelektronik Tagung*, pages 207–214, 2005.