Curved Tags - A Low-Resource ECDSA Implementation tailored for RFID

Peter Pessl and Michael Hutter

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
peter.pessl@gmail.com, Michael.Hutter@iaik.tugraz.at

Abstract. In recent years, a lot of effort was made to deploy asymmetric cryptography based on ECC to affordable RFID tags. However, many proposed hardware designs suffer from long execution times and many resource requirements. In this paper, we address this issue by presenting a low-resource implementation of a 160-bit ECDSA signature generation algorithm. As a novelty, we make use of the new KECCAK hashing algorithm. Moreover, we applied state of the art techniques such as co-Z ECC formulæ, a pipelined multiplication unit, RAM macros, and we evaluated fixed-base comb methods to improve the efficiency of ECDSA on passive tags. Furthermore, our design runs with constant runtime and provides basic resistance against common implementation attacks. It requires a total area of 12 448 GEs (including memory) and can generate a message digest within 140 kCycles, which is both smaller and considerably faster than comparable work. It has a power consumption of 42.42 µW at 1 MHz on a low-leakage 130 nm CMOS process technology. Our implementation competes with binary-field based ECC solutions not only in terms of area and power but also in speed.

Keywords: Low-Resource ASIC Design, ECC, Keccak, SHA-3.

1 Introduction

Implementing strong cryptography on Radio Frequency Identification (RFID) tags is a challenging but necessary task. Passively powered tags draw the required energy from the electromagnetic field of a reader; thus, much effort has to be put into low-power hardware designs to achieve high reading ranges. Moreover, tags must be cheap in order to keep production costs low, so the entire circuit has to be as small as possible.

Asymmetric cryptography is considered to be more resource consuming than symmetric cryptography. However, it has the big advantage that it makes key-distribution problems easier which is an important requirement especially in cases where RFID tags are going to be deployed in a large scale. Amongst many cryptographic services, the Elliptic Curve Digital Signature Algorithm (ECDSA) has been constituted over many years to provide most of the required needs for practical applications. It is a building block that can be used within several authentication protocols, e.g., embedded in identification schemes to allow entity authentication or in signature schemes to offer message authentication. The latter service is thereby an important feature to implement proof-of-origin applications—a feature that can help to prevent the exponential rise in counterfeiting activity.

There already exist many low-resource hardware implementations of ECDSA. However, many of these designs suffer from either being very large in size or they need horrible execution times. This makes the designs often not well suitable for various RFID applications where the costs are limited or fast tag response times are mandatory.

Our contribution. In order to tackle these problems, we present a highly optimized hardware implementation of ECDSA over the 160-bit prime-field curve `secp160r1`. Instead of integrating SHA-1 or SHA-2 into ECDSA, we are first to integrate the Keccak algorithm that is the winner of the NIST SHA-3 competition. We present a novel hardware architecture of Keccak by applying a *factor-4 interleaving* method that improves the performance compared to related work while consuming low resources. Concerning ECC scalar multiplication, we apply new techniques on various implementation levels, e.g., we use a 32-bit datapath width but integrate a 16-bit pipelined multiplication unit, we make use of an efficient RAM macro that is available for our targeted 130 nm CMOS process technology, we consider new explicit co-Z formulæ requiring 7 registers, and evaluate a fixed-base comb scalar multiplication technique on passive RFID tags. Especially the integration of comb methods show that a speed-up by a factor of 4 can be achieved with the costs of about 800 GEs for storing the required comb points. These results offer valuable insights to obtain good trade-offs between speed and production costs.

Our work requires an area of 12 448 GEs (or 63 700 μm^2 in 130 nm), has a mean power consumption of 42.42 μW per MHz, and generates a digital signature with less than 140 kCycles. These results make our design smaller and considerably faster than previously presented designs of comparable implementations. With the applied techniques our design is on a par with related work on low-resource binary-curve implementations. This is not surprising because we used a fixed-base comb technique but it is interesting for the RFID community and industry to achieve performances similar to binary-field based curve implementations while keeping the resource requirements low (especially for storing the necessary comb points).

Outlook. The paper is organized as follows. In Section 2, we first give a brief introduction to ECC and ECDSA. After that, we list the imposed requirements and most important design decisions. Our low-resource architecture is then presented in Section 3. Section 4 discusses the Keccak architecture and its integration into the design. Countermeasures aimed at securing the device against SCA are given in Section 5. Results are presented in Section 6.

2 Design Space Exploration

In this section, we first give a short introduction to ECC and ECDSA. Afterwards, we discuss the requirements and the needed properties of our design. Finally, we list all design choices for our targeted RFID-tag architecture.

Algorithm 1 ECDSA signature generation.

```
Require: Domain Parameters $T = \{p, a, b, G, n, h\}$, private key $d$, message $m$.
Ensure: Signature $(r, s)$.
1: Compute hash $e = \text{Hash}(m)$, truncate to bit length of $n$.
2: Select random $k \in [1, n-1]$.
3: Compute $(x, y) = kG$, $r = x \mod n$, if $r = 0$ goto 1.
4: Compute $s = k^{-1}(e + rd) \mod n$, if $s = 0$ goto 1.
5: return $(r, s)$.
```

ECC and the ECDSA. Elliptic curves are the base for many cryptographic primitives. An elliptic curve E defined over a finite-field $\mathbb{F}_p$ can be given in short Weierstrass form $y^2 = x^3 + ax + b$, with publicly known parameters $a, b \in \mathbb{F}_p$. Pairs of (x, y), with $x, y \in \mathbb{F}_p$, satisfying this equation are points $P \in E$. For a fixed point P with prime order n and a large integer $k \in [1, n-1]$, one can easily compute the point multiplication $Q = kP$. The inverse operation, i.e., finding k with given Q, P, is computationally hard and known as the Elliptic Curve Discrete Logarithm Problem (ECDLP). Point-scalar multiplications are carried out by means of the group operations point doubling and point addition, they require arithmetic in $\mathbb{F}_p$. Efficient curve addition and doubling formulæ, typically using some sort of projective coordinates, try to reduce the required field operations, especially the more expensive multiplications (M) and squarings (S).

ECDSA is the elliptic-curve counterpart of the Digital Signature Algorithm (DSA). Before signing, all parties must agree on domain parameters T, which specify a curve alongside a base point G with prime order n. For signature generation (Algorithm 1), the message m is hashed, then the base point G is multiplied with the nonce k. The resultant x coordinate and the variable s form the signature pair (r, s).

The Requirements. Goal of this work was to design an ASIC implementation of the ECDSA signature-generation algorithm. Our aim was to implement a low-cost coprocessor that features message authentication services and thus allows proof-of-origin authentication of RFID-tagged goods, for example, to tackle the problem of product counterfeiting. Signature verification is not supported and we did not include a random-number generator. To limit the tag requirements, we decided to base our implementation on the 160-bit standardized `secp160r1` [8,9] elliptic curve that features 80-bit security[a]. Starting from now the variables p, n refer to the primes defined in the standard, $\mathbb{F}_p$ and $\mathbb{F}_n$ denote the respective finite fields.

As opposed to many other hardware implementations that use the (older) SHA-1 or SHA-2 algorithms, we decided to evaluate the recently announced NIST SHA-3 competition winner, i.e., the KECCAK hashing algorithm [5]. A subset of

[a] Note that many standardization bodies removed 160-bit prime curves from their recommendation standards but a 80-bit security level is still sufficient for many low-cost RFID applications.

the algorithm will be incorporated into the Secure Hash Standard (SHS). Keccak is highly tunable, the security level can be controlled by setting the security parameter c (which stands for *capacity*). For a chosen capacity c, Keccak offers both a preimage and collision resistance of $c/2$ bits [4], so in order to match the security of the curve it was decided to set $c = 160$. The state size is also selectable, so we decided to use an 800-bit state instead of the 1 600-bit (full state) version to reduce the memory requirements. Thus, the exact used Keccak instance is called Keccak[r=640, c=160]. Note that in [28], it was shown that area-wise dedicated RAM-based implementations of 1 600 and 800-bit state versions require nearly the same amount of resources but the 800-bit version is about twice as fast as the full state version.

For designs meant to be deployed on (passive) RFID tags, low-power and low-area requirements are a prime goal (to obtain appropriate reading ranges and to be cheap in the production costs). So in the past, many designers that met these requirements payed the price of horrible execution times. To give an example, the work of Wenger et al. [32,33] needs more than 1 million clock cycles to perform ECDSA, which corresponds to about 10 seconds if the tag is clocked with 100 kHz. So in addition to the previous goals, we aim to drastically reduce the execution time of ECDSA in this paper while keeping the resource requirements as low as related work.

2.1 Basic Design Choices

In order to fulfill the requirements, we made the following design choices.

Storage type. Previous work, e.g., [16,17,18,21], often used some sort of synthesized dual-ported memory for storage of large data. The advantage of dual-port memory is that the core can access two words within one clock cycle. The major disadvantage, however, is that the size is almost twice as large as single-port memories. For example, the 2 048 bit dual-port register file RAM macro for UMC 130 nm CMOS from Faraday [11] requires $0.028\,mm^2$ while the single-port variant requires only $0.018\,mm^2$. We therefore decided to use a single-ported RAM macro as main storage element. Such specialized macros typically require much less resources than standard-cell based memory blocks.

Datapath and memory width. We decided to implement a 32-bit datapath and use a 32-bit memory width interface to achieve our targeted computation speeds. To keep the power consumption low, we integrated a 16×16-bit multiplier core that is used to perform a 32×32-bit multiplication within 4 clock cycles. Note that a 32×32-bit multiplier would speed up the computation but would drastically increase area and power requirements because it dominates the ECDSA datapath complexity. Also, the single-port memory is not able to deliver data fast enough to keep such a large multiplier busy.

ECSM with fixed-base comb methods. The elliptic-curve scalar multiplication is by far the most time consuming part of ECDSA, so it is essential to speed up this process. Comb methods, first proposed by Lim and Lee [24], allow a drastic speed improvement for schemes with a fixed-base point P. For a chosen width w, one needs to precompute all $[\alpha_{w-1}, \ldots, \alpha_1, \alpha_0]P = \alpha_{w-1}2^{(w-1)\ell}P + \cdots + \alpha_1 2^{\ell}P + \alpha_0 P$ and store these points in a (read-only) memory. Comb methods then rearrange the n-bit scalar k in a matrix with dimension $w \times \ell$, with $\ell = \lceil n/w \rceil$. The columns of this matrix are then processed from left to right, in a simple double-and-add fashion. Typical implementations, e.g., [7], have one major problem: if all column bits are equal to zero then the point addition step must be skipped. Such behavior is detectable by Simple Power Analysis (SPA) attacks and should therefore be avoided.

Hedabou et al. [15] presented a comb scheme resistant to SPA. The scalar k is first recoded using the *Zeroless Signed Digit* (ZSD) scheme. This recoding technique represents an odd integer with digits in $\{-1, 1\}$ and is based on the observation that $1 = 2^w - \sum_{i=0}^{m-1} 2^i, \forall w > 0$. Hence, all blocks of $000\ldots01$ (w bits) can be replaced by w signed digits $1\bar{1}\bar{1}\ldots\bar{1}\bar{1}$, with $\bar{1} = -1$ [14]. This representation can be obtained by simply shifting k to the right and reinterpreting all 0 bits as $\bar{1}$. The hardware costs of obtaining the ZSD representation is therefore almost zero. In fact, all-zero columns can obviously not appear when using a *zeroless* representation. Also, usage of a signed-digit representation allows to reduce the memory requirements from $2^w - 1$ [7] to 2^{w-1} precomputed points by utilizing the fact that point negations are easy to compute.

For this work, $w = 4$ was chosen. The reason for this is that it offers a good speed-up (factor 4) while keeping the hardware requirements low, i.e., 8 points need to be precomputed and stored in a ROM, which requires in total 2 560 bits (modern hardware synthesizers can implement these ROMs with less than 1 000 GEs).

EC addition formulæ. Execution of a comb method requires so-called point doubling-additions, i.e., point operations of the form $R = 2P + Q$. For this work, the doubling-addition formulæ by Longa and Miri [25] are used, they require 11 field multiplications (M) and 7 squarings (S), and can be executed with 7 field registers. They are based on Meloni's co-Z addition formulæ [26]. Note that there exist faster point doubling-addition formulæ based on co-Z notation, e.g., as shown in [14,19]. However, they rely on an update mechanism where the added point is updated with a new Z coordinate after each doubling-addition. For the case of comb methods, the added point is selected out of a set of precomputed points stored in ROM, so the faster co-Z algorithms are not efficiently applicable there.

Field inversions. Field inversions should be computed by means of a modular exponentiation, i.e., by applying Fermat's little theorem that states $a^p \equiv a \bmod p$, for all primes p. Then the inverse $a^{-1} \equiv a^{p-2} \bmod p$. Other algorithms, e.g., the binary or Montgomery inversion algorithm, are typically faster, but involve non-constant runtime loops or branch conditions and are therefore susceptible to side-channel attacks.

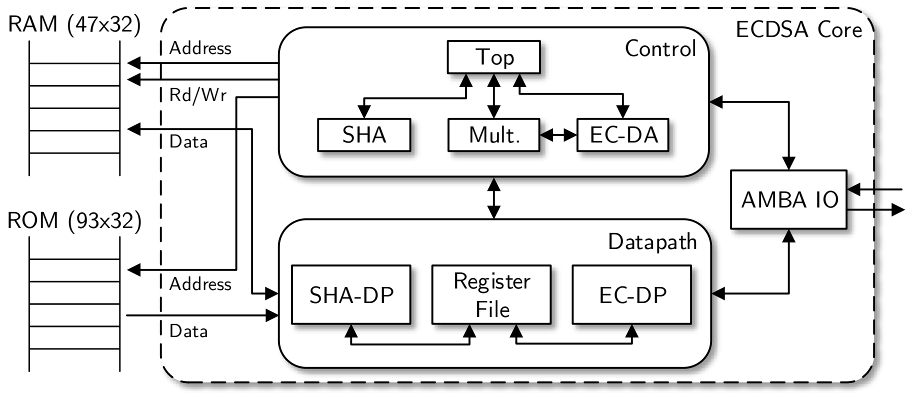


Fig. 1: Top-level view of the ECDSA core

3 A Low-Resource ECDSA Hardware Architecture

In this section, we present our low-resource ECDSA hardware architecture. We first give an overview and then discuss the datapath in more detail. Afterwards, we illustrate the implemented modular-reduction mechanisms and present optimized inversion algorithms.

3.1 Implementation Overview

Figure 1 shows the top-level structure of our design. The computation core contains a datapath, a controlling block, and an 8-bit AMBA APB interface. The Advanced Microcontroller Bus Architecture (AMBA) Advanced Peripheral Bus (APB), a very simple yet standardized interface, allows communication with the outside world, i.e., a bus master. The datapath (DP) can be split into separate modules, the SHA-DP is dedicated to the Keccak hashing algorithm and the EC-DP handles all other computations required for ECDSA, both parts share a common register file. The controlling block is comprised of multiple subcontrollers and a top-level controller, which is in charge of overseeing the signing process and steering the subcontrollers. The multiplication controller handles modular multiplication in both fields $\mathbb{F}_p$ and $\mathbb{F}_n$ by implementing two reduction techniques. The EC-DA controller houses the control logic required for performing elliptic-curve doubling-additions. Finally, the Keccak controller is dedicated to steering the hashing process. The controllers are implemented using a mixed approach, both finite-state machines and microcontroller-like programming are used.

The core is connected to a 47×32 bit (single-ported) RAM and a 93×32 bit ROM that stores all necessary constants (ECC comb points, prime modulus n, etc.).

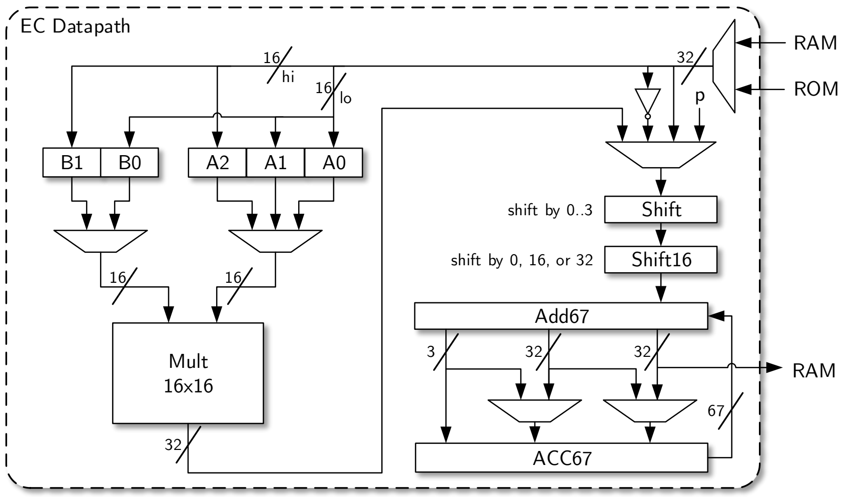


Fig. 2: The implemented ECDSA datapath

3.2 The Datapath

The datapath, shown in Figure 2, is comprised of a multiplication (left) and an accumulation path (right). A 67-bit accumulator register (ACC) alongside a 67-bit adder form an accumulation unit. Multiplexers allow shifting the current adder output by 32 bits to the right. The lower 32 bits of the adder output are fed to the RAM.

The accumulation operand can be selected from the inverted or non-inverted RAM output, the multiplication result or (a 32-bit part of) the constant modulus p. Thanks to the highly regular modulus p—only a single bit is zero, all others are one—it is possible to hardcode this constant without any noteworthy area gain. The selected operand is then routed through two configurable shifters, the first one can shift its input up to 3 bits to the left and thus produces a 35-bit output, while the second shifter can shift this result by either 0, 16, or 32 bits to the left.

The multiplication part of the datapath consists of two multiplication operand registers A, B, and a 16×16-bit integer multiplier producing a 32-bit output. The operand registers are made up of 16-bit chunks, A consists of 3 parts A0 to A2, while B consists of 2 parts B0 and B1. The operand registers are used to perform a pipelined multiplication, as explained in the next section.

3.3 Pipelined Multi-Precision Multiplication

To fully use the 32-bit single-ported memory interface, 32×32-bit multiplications are computed with the help of the dedicated 16-bit multiplier. A 32-bit multiplication takes four cycles and is done with a simple school book multiplication algorithm, as illustrated in Figure 3. Both 16-bit chunks of the first operand (A0 and A1) are multiplied with both chunks of the second operand (B0 and B1).

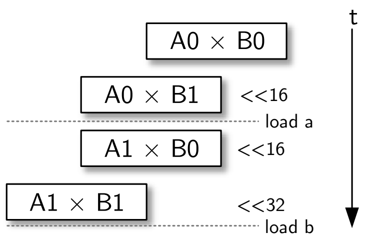


Fig. 3: Pipelined school-book multiplication scheme

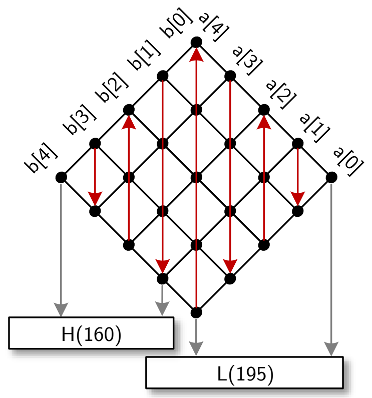


Fig. 4: Modular multiplication in $\mathbb{F}_p$

The 32-bit partial products are first shifted accordingly using the second shifter and then added to the accumulator, thus utilizing the multiply-and-accumulate (MAC) functionality of the datapath.

Prior to the start of the 32-bit multiplication, both operands need to be loaded into the designated operand registers. Dedicated operand load cycles would slow down the multiplication process, thus the operands for the next 32-bit multiplication are fetched during execution of the current one. As denoted by dotted lines in Figure 3, operand B is replaced at the end of the fourth cycle, operand A after the second cycle. However, the 16 high-order bits of A (A1) are still needed in the third and fourth multiplication cycle. For this reason there are three 16-bit operand registers for A. The 16 high-order bits of operand A are alternately stored in the operand registers A1 and A2.

A product-scanning approach is used to perform 160-bit multiplications by means of 32-bit ones, this also utilizes the MAC functionality. 25 32-bit multiplications need to be computed, so a plain (non-modular) multiplication takes $25 \times 4 = 100$ cycles. Squarings can be sped up by using the commutative property of the multiplication, i.e., $a[i]a[j] = a[j]a[i], \forall i \neq j$, and hence $a[i]a[j] + a[j]a[i] = 2(a[i]a[j])$. Only 15 partial products need to be computed, this takes 60 cycles. The doubling operation is carried out by the first shifter in the datapath, thus the hardware cost of this simple optimization is almost equal to zero.

The following sections discuss the integration of modular arithmetic into this multiplication scheme.

Arithmetic in $\mathbb{F}_p$. The prime $p = 2^{160} - 2^{31} - 1$ is a so-called pseudo-Mersenne prime that permits fast reduction. An integer $x > p$ can be reduced by splitting it in $x = h2^{160} + l$ and then computing $x \equiv l + h + (h << 31) \bmod p$, i.e., reduction is achieved using shifts by 31 digits and additions. Unfortunately, 31 is not a multiple of the word size 32, thus disallowing shifting by simple addressing.

Instead the datapath is modified to allow addition of a 32-bit word, i.e., a word of h, in two different locations of the accumulator concurrently.

The fast reduction algorithm is not only used in multiplication, but also for the basic operations addition, subtraction, and shifting. In the case of subtraction, a (fixed) multiple of the modulus is first added to the difference to ensure a positive intermediate result.

To gain higher speeds, the reduction is integrated into the multiplication process, this gives a time and storage space advantage. As seen in Figure 4, multiplication is performed in two phases. First only the upper columns 5 to 8 are processed, the 160-bit result H is stored in RAM. Then multiplication continues with the lower columns, the addition of H is interleaved with the multiplication process. The 195-bit sum $L + H + (H << 31)$ is again stored in RAM. Finally, another round of reduction is performed, this produces the final 160-bit output. In total, modular multiplication or squaring requires 123 or 83 cycles, respectively.

Arithmetic in $\mathbb{F}_n$. The prime group order n is not of any special form, thus a general reduction process needs to be implemented in addition. We therefore decided to implement the Montgomery multiplication scheme [27]. It requires division and reduction by a chosen integer R, with $R > n$. We used the *Integrated Product Scanning* approach by Koç et al. [22] in this work.

One of n's properties makes implementing modular multiplication using the Montgomery method difficult: its bit length of 161. The implementation is geared towards handling of 160-bit integers, everything above that adds an area and time overhead. Also, 161 is not a multiple of the word size 32.

However, the structure of the modulus n—the MSB is followed by many zero bits—allows optimizations. The probability of a random element in field $\mathbb{F}_n$ being greater or equal to 2^{160} is 2^{-79}. So we can assume that all operands and results of the Montgomery multiplications performed throughout signing are restricted to 160 bits. If, despite the diminishing odds, a multiplication yields a 161-bit result, the outcome can not be used as input for the following multiplications. Such an occurrence is detected and an error is returned at the end of the signing process. Restricting all values to 160 bits allows setting $R = 2^{160}$. This does not satisfy the requirement of $R > n$, however, an R greater than both multiplication operands a, b suffices. Montgomery's argumentation includes the estimation $ab < Rn$, with $a, b \in \mathbb{F}_n$ and $R > n$. This is obviously also true with $a, b < R < n$.

The output of the algorithm t is in range $t < 2n$, a conditional subtraction is required to retrieve the final output. Instead of avoiding this conditional subtraction, as first suggested by Walter [30,31], it is always executed and both t and $t - n$ are stored in RAM. The subsequent operation then accesses the correct result. The runtime of a single multiplication is 197 cycles.

3.4 Optimized Prime Field Inversion Algorithms

Field inversions are performed by means of a modular exponentiation, i.e., by using $a^{-1} \equiv a^{p-2} \bmod p$. Two inversions are performed during ECDSA, one after

ECC scalar multiplication (to back-transform into affine coordinates) and one during the signing process (to invert the ephemeral key k).

Z^{-2} mod p. The exponent in $Z^{-2} \equiv Z^{p-3}$ contains only three 0 bits, so an optimization of the used left-to-right square-and-multiply exponentiation algorithm is possible. By iteratively computing $Z^{2^{\ell}-1}$, with $0 < \ell \leq 7$, and storing some intermediate values, one can retrieve the inverse in 159 squarings (S) and only 11 multiplications (M). Our implementation needs 14 550 cycles.

k^{-1} mod n. Here a highly customized window algorithm is used. Instead of precomputing all possibilities of a fixed-size window, only selected powers are precomputed. A brute-force search for the ideal set of precomputed powers was conducted. In the initial precomputation phase the powers k^3, k^5, and k^9 are computed and stored in RAM, the actual multiplication is then performed in a simple square-and-multiply fashion. This inversion algorithm takes a total of 160S and 26M, or 30 252 cycles.

4 Integration of Keccak

The hashing modules are largely based on our previously presented Keccak implementation [28]. Some changes are made to adapt the design to the different environment. The RAM interface is widened to 32 bits and the used interleaving scheme is changed from factor-2 to a factor-4 to achieve a higher bus-width utilization.

Combined Processing and Interleaving. Keccak is defined for state sizes of $b = 25w$, with $w = 2^{\ell}$ and $0 \leq \ell \leq 6$. The Keccak-f permutation internally organizes this state as a 3-D matrix with dimension $5 \times 5 \times w$. This matrix can be split into 25-bit slices and w-bit lanes. The Keccak-f permutation is a round-based function, in each of its $12 + 2\ell$ rounds, five state mappings θ, ρ, π, χ and ι are executed. For a more thorough explanation of the components, we refer to the Keccak reference [5].

Most software implementations process the lane state-wise, i.e., lane after lane is fetched and processed. Slice-wise processing, first proposed by Jungk and Apfelbeck [20], poses as a hardware-friendly alternative. The described design features both slice and lane-wise processing. The round function is rescheduled and split into a slice-processing and a lane-processing phase.

The combined-processing approach requires that both slices and lanes can be accessed efficiently. This is challenging when using a RAM for state storage. To achieve a high bus-width utilization, an interleaved storage scheme is used. Four consecutive lanes are bit-wise interleaved, these interleaved words are then stored in RAM. In [28], only two lanes are interleaved. Due to the wider memory bus and the restriction to an 800-bit state, a factor-4 interleaving is more suitable here.

The Keccak Datapath. The Keccak datapath consists of four 32-bit registers, a slice unit, four rotation units required for the ρ transformation, and an

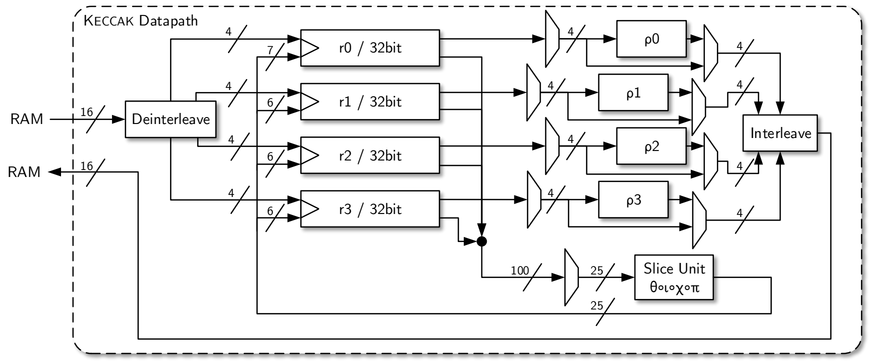


Fig. 5: KECCAK datapath

interleave and deinterleave unit. During the lane-processing phase each register stores one lane, during slice processing four slices are stored. For a more detailed explanation we refer the reader to [28].

KECCAK Integration. In order to save precious resources, the hashing modules are tightly integrated into the existing design.

One major shared part is obviously the RAM, the KECCAK modules use the same RAM used by other parts of the ECDSA algorithm. During hashing the 800-bit state is stored in the lower-order parts of the RAM.

It is also possible to merge the internal registers. The numbers of register bits included in the KECCAK datapath and ECDSA datapath match up nicely, which is not a coincidence. In fact, the ECDSA datapath was designed with the KECCAK memory requirements in mind. The registers are hence merged into a single shared register file, as previously seen in Figure 1.

5 Protections against Implementation Attacks

A chain is only as strong as its weakest link, so implementing a secure protocol like ECDSA without considering implementation attacks is grossly negligent (especially on easily accessible RFID tags). We implemented the following countermeasures to secure our design.

- Constant runtime and operation flow to provide protection against timing attacks and basic[b] Simple Power Analysis (SPA) attacks:
  - Constant runtime of modular reduction
  - Avoid negative results after subtractions by adding modulus p

[b] Constant runtime does not automatically make the implementation resistant against SPA attacks but often makes SPA attacks harder to perform because attackers are forced to exploit data-dependent leakage only instead of typically easier detectable operation-dependent leakage.

  - Constant runtime of conditional negation of stored y coordinate. The implemented comb method requires a conditional negation of the y coordinate of a stored point. Here first $2y \bmod p$ is computed, then the desired result is retrieved as $y = 2y - y \bmod p$ or $-y = y - 2y \bmod p$.
  - No all-zero columns in comb
  - Inversion using Fermat's little theorem
  - Always execute final subtraction in Montgomery multiplication
- We applied the Randomized Projective Coordinates (RPC) [10] countermeasure to thwart Differential Power Analysis (DPA) attacks. The (affine) coordinates of the first referenced comb point are transformed to a projective representation with a random Z coordinate. Then the processed values are different even if the same k is used. RPC also offers protection against template-based SPA attacks.
- As a final countermeasure, the computation of $s = k^{-1}(e + dr) \bmod n$ is reordered as $s = k^{-1}e + (k^{-1}d)r$. Then, the private key d is multiplied with the inverse of the random nonce, multiplying it with the known signature part r would open the gate for DPA attacks [17].
- ECDSA is not susceptible to refined power-analysis attacks (RPA) [13] and zero-value point attacks (ZPA) [1]. They require point multiplications with a fixed scalar and user-selectable base point (such as in ECDH schemes), neither is the case for ECDSA.

6 Results

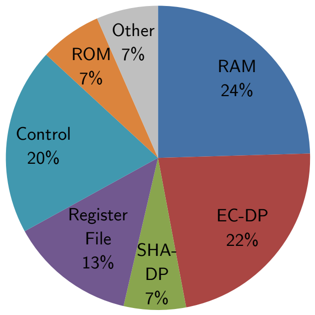
The design was implemented in VHDL and then synthesized using a mixed tool design flow. The Synopsys Design Compiler 2013.03 generated a netlist targeting the *FSC0L_D* standard-cell library from Faraday. The 0.13 µm *low-leakage* process by UMC is the base for this library. Single-ported RAM macros by Faraday are used as storage elements. All circuit-area results are given as reported by the synthesizer. Circuit area is given in *gate equivalents* (GE), 1 GE is equal to the size a two-input NAND gate (5.12 µm^2 in the chosen process). An analysis with the Encounter Power System v8.10 yielded the power consumption, the operating frequency was set to 1 MHz to allow a fair comparison.

Figure 6 gives an overview of the area usage of different parts of the design. With 3 055 GEs, the RAM block takes up roughly 1/4 of the total area of 12 448 GEs. The datapath, including the ECDSA datapath (Figure 2), the Keccak datapath (Figure 5), and the shared register file, amounts to 42 %. The EC-DP is dominated by the 16×16 multiplier with its 1 616 GEs. Interestingly, only 800 GEs are required for the ROM that is mainly used to store the precomputed elliptic curve points (2 976 bits in total). This is a very small price to pay when considering the massive speed-up provided by the fixed-base comb method (factor 4 times faster).

Signing a 160-bit message takes only 139 930 cycles. The point-scalar multiplication is by far the most time-consuming operation, it requires 87.5 kCycles. The first field inversion $Z^{-2} \bmod p$ uses 14.5 kCycles, due to the more expensive Montgomery multiplication scheme the second inversion $k^{-1} \bmod n$ is twice

Component	**Area**	
	[GEs]	**[μm^2]**
EC-DP	2 800	14 334
Multiplier	1 616	8 273
SHA-DP	823	4 211
Register File	1 662	8 509
Control	2 473	12 663
Other	816	4 178
ROM	820	4 198
Core Total	**9 393**	**48 093**
RAM	3 055	15 642
Total	**12 448**	**63 735**

(a) Area of chip components



(b) Area distribution

Fig. 6: Area of chip components

as expensive (30.3 kCycles). A single 640-bit message block can be hashed in 5.5 kCycles, which is an improvement over the 7.6 kCycles given in [28].

Comparison with Related Work. In Table 1, we compare our design with other low-resource implementations. In [21], Kern et al. presented an ECDSA design using the same `secp160r1` curve but with SHA-1 instead of KECCAK and a 350 nm CMOS technology. Compared to his implementation, we could improve the speed by a factor of about 3.6 (thanks to the use of the fixed-base comb method) and could reduce the area requirements by about 5 800 GEs (through the use of dedicated single-ported RAM macros). Recently, at the ECC workshop 2013, Roy et al. [29] reported a tiny ECC co-processor supporting both $\mathbb{F}_{p160}$ and $\mathbb{F}_{p192}$ arithmetic for a 32 nm IBM CMOS technology. Their design requires about 26 kGEs (in total) and 250 kCycles for a scalar multiplication over $\mathbb{F}_{p160}$. This is about twice as slow as our design.

ECDSA implementations over 192-bit prime fields have been, for example, reported by Wenger [32], Hutter et al. [17], and Fürbass et al. [12], They require more area, partly due to the larger prime field size, and are also significantly slower (in particular, our implementation is about 10 times faster than the work of Wenger [32]). Interestingly, our implementation can compete with many binary-field ECC processors. E.g., our ECDSA implementation is nearly as large as the work of Hein et al. [16] and Lee et al. [23] who reported an ECC implementation over $\mathbb{F}_{2^{163}}$. Compared to the results of Lee (with chosen multiplier digit-size $d = 1$), our implementation is about 2 times faster, includes KECCAK, and allows computation of digital signatures.

Power consumption is extremely difficult to compare over different process technologies, so values are only given for designs using a 130 nm process. The 192-bit prime curve processor by Wenger—it uses the same 130 nm process technology used here—requires 39.54 µW/MHz and 55 µJ of energy. In comparison, the power

Table 1: Comparison of prime and binary field ECC implementations

	Techn. [nm]	**Area [GEs]**	**Time [Cycles]**	**Power**[a] **[µW/MHz]**	**Field**	**Features**[b]
This work[c]	**130**	**12 448**	**139 930**	**42.42**	$\mathbb{F}_{p160}$	**ECDSA, Keccak**
Roy13 [29]	32	26 000	250 000	-	$\mathbb{F}_{p160}$	ECSM, dual field
Kern10 [21]	350	18 247	511 864	-	$\mathbb{F}_{p160}$	ECDSA, SHA-1
Wenger11 [32,33][cd]	130	14 644	1 394 000	39.54	$\mathbb{F}_{p192}$	ECDSA, SHA-1
Hutter10 [17]	350	19 115	859 188	-	$\mathbb{F}_{p192}$	ECDSA, SHA-1
Fürbass07 [12]	350	23 656	500 000	-	$\mathbb{F}_{p192}$	ECSM
Lee08 [23] (d=1)	130	12 506	302 457	32.42	$\mathbb{F}_{2^{163}}$	ECSM, Schnorr
Hein08 [16]	180	11 904	296 000	-	$\mathbb{F}_{2^{163}}$	ECSM
Bock08 [6] (d=4)	220	12 876	80 000	-	$\mathbb{F}_{2^{163}}$	ECSM, DH

[a] Power values of designs using other process technologies are omitted
[b] ECSM: Point-scalar multiplication only
[c] Uses a RAM macro for storage.
[d] In [32], an area of 11.7 kGEs was achieved based on a 180 nm process. In [33], the same design requires 14.6 kGEs in a 130 nm process.

consumption of our design is slightly higher (42.42 µW/MHz), but due to the lower cycle count the energy consumption is considerably smaller (6 µJ).

6.1 Discussion

The following points invite to further discussion.

Horizontal SCA. Horizontal SCA (e.g., [3]) was not considered in this paper, susceptibility of comb methods to such attacks needs to be evaluated. Especially the multiplication with the precomputed x and y coordinates might leak information. This operation could be secured by randomizing the sequence of partial product computation, as suggested by Bauer et al. [2].

Extension to larger curves. For applications with higher security demands the design could be extended to larger, e.g., 192 or 224-bit, curves. The computation core could be mostly reused, only small adaptations are required. However, the memory requirements (RAM and ROM) would rise, in the case of a 192-bit curve by 20 % or roughly 1 kGEs. Also, computation time would increase to around 200 kCycles.

Acknowledgements. The work has been supported by the Austrian Science Fund (FWF) under the grant number TRP251-N23 (Realizing a Secure Internet of Things - ReSIT), the FFG research program SeCoS (project number 836628), and the European Cooperation in Science and Technology (COST) Action IC1204 (Trustworthy Manufacturing and Utilization of Secure Devices - TRUDEVICE).

References

1. T. Akishita and T. Takagi. Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In C. Boyd and W. Mao, editors, *Information Security*, volume 2851 of *Lecture Notes in Computer Science*, pages 218–233. Springer Berlin Heidelberg, 2003.
2. A. Bauer, E. Jaulmes, E. Prouff, and J. Wild. Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations. In E. Dawson, editor, *Topics in Cryptology CT-RSA 2013*, volume 7779 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 2013.
3. A. Bauer, E. Jaulmes, E. Prouff, and J. Wild. Horizontal Collision Correlation Attack on Elliptic Curves. In *Selected Areas in Cryptography*, 2013.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Cryptographic sponge functions. Submission to NIST (Round 3), 2011.
5. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak reference. Submission to NIST (Round 3), 2011.
6. H. Bock, M. Braun, M. Dichtl, E. Hess, J. Heyszl, W. Kargl, H. Koroschetz, B. Meyer, and H. Seuschek. A Milestone Towards RFID Products Offering Asymmetric Authentication Based on Elliptic Curve Cryptography, July 2008.
7. M. K. Brown, D. R. Hankerson, J. C. L. Hernández, and A. J. Menezes. Software Implementation of the NIST Elliptic Curves Over Prime Fields. In D. Naccache, editor, *CT-RSA 2001, San Francisco, CA, USA, April 8-12*, volume 2020 of *LNCS*, pages 250–265. Springer, 2001.
8. Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0. Available online at `http://www.secg.org/`, September 2000.
9. Certicom Research. Standards for Efficient Cryptography (SECG), SEC 1: Elliptic Curve Cryptography, Version 1.0. Available online at `http://www.secg.org/`, September 2000.
10. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *CHES, Worcester, MA, USA, August 12-13*, volume 1717 of *LNCS*, pages 292–302. Springer, 1999.
11. Faraday Technology Corporation. Faraday FSA0A_C 0.13 μm ASIC Standard Cell Library, 2014. Details available online at `http://www.faraday-tech.com`.
12. F. Fürbass and J. Wolkerstorfer. ECC Processor with Low Die Size for RFID Applications. In *Proceedings of IEEE International Symposium on Circuits and Systems*. IEEE, IEEE, May 2007.
13. L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In Y. Desmedt, editor, *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, volume 2567 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2003.
14. R. R. Goundar, M. Joye, A. Miyaji, M. Rivain, and A. Venelli. Scalar multiplication on Weierstraß elliptic curves from co-Z arithmetic. *Journal of cryptographic engineering*, 1(2):161–176, 2011.
15. M. Hedabou, P. Pinel, and L. Bénéteau. Countermeasures for Preventing Comb Method Against SCA Attacks. In *First International Conference on Information Security Practice and Experience*, ISPEC'05, pages 85–96. Springer-Verlag, 2005.
16. D. Hein. Elliptic Curve Cryptography ASIC for Radio Frequency Authentication. Master thesis, Technical University of Graz, April 2008.

17. M. Hutter, M. Feldhofer, and T. Plos. An ECDSA Processor for RFID Authentication. In S. B. O. Yalcin, editor, *RFIDsec 2010, 6th Workshop, Istanbul, Turkey, June 7-9*, volume 6370 of *LNCS*, pages 189–202. Springer, 2010.
18. M. Hutter, M. Feldhofer, and J. Wolkerstorfer. A Cryptographic Processor for Low-Resource Devices: Canning ECDSA and AES like Sardines. In C. A. Ardagna and J. Zhou, editors, *Information Security Theory and Practices – WISTP, Heraklion, Crete, Greece, June 1-3*, volume 6633 of *LNCS*, pages 144–159. Springer, 2011.
19. M. Hutter, M. Joye, and Y. Sierra. Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In A. Nitaj and D. Pointcheval, editors, *AFRICACRYPT 2011, Dakar, Senegal, July 5-7*, volume 6737 of *LNCS*, pages 170–187. Springer, 2011.
20. B. Jungk. Area-Efficient FPGA Implementations of the SHA-3 Finalists. In *Reconfigurable Computing and FPGAs–ReConFig 2011, International Conference, November 30-December 2, Cancun, Mexico, 2011*, pages 235–241, 2011.
21. T. Kern and M. Feldhofer. Low-Resource ECDSA Implementation for Passive RFID Tags. In *ICECS, December 12-15, Athens, Greece*, pages 1236–1239. IEEE, 2010.
22. Ç. K. Koç, T. Acar, and B. S. Kaliski Jr. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, 16(3):26–33, June 1996.
23. Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede. Elliptic-Curve-Based Security Processor for RFID. *IEEE Transactions on Computers*, 57(11):1514–1527, November 2008.
24. C. H. Lim and P. J. Lee. More Flexible Exponentiation with Precomputation. In Y. Desmedt, editor, *Advances in Cryptology - CRYPTO '94*, volume 839 of *LNCS*, pages 95–107, 1994.
25. P. Longa and A. Miri. New Composite Operations and Precomputation Scheme for Elliptic Curve Cryptosystems over Prime Fields. In *in Public Key Cryptography (PKC08), LNCS*, pages 229–247. Springer, 2008.
26. N. Meloni. New Point Addition Formulae for ECC Applications. In C. Carlet and B. Sunar, editors, *Arithmetic of Finite Fields*, volume 4547 of *LNCS*, pages 189–201. Springer Berlin Heidelberg, 2007.
27. P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44:519–521, 1985.
28. P. Pessl and M. Hutter. Pushing the Limits of SHA-3 Hardware Implementations to Fit on RFID. In G. Bertoni and J.-S. Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *LNCS*, pages 126–141. Springer Berlin Heidelberg, 2013.
29. S. S. Roy, B. Yang, V. Rozic, N. Mentens, J. Fan, and I. Verbauwhede. Designing Tiny ECC Processor. Available online at `https://www.cosic.esat.kuleuven.be/ecc2013/files/sujoy.pdf`, 2013. Presentation at the 17th Workshop on Elliptic Curve Cryptography (ECC 2013).
30. C. D. Walter. Montgomery Exponentiation needs no Final Subtractions. *Electronics Letters*, 35:1831–1832, 1999.
31. C. D. Walter. Montgomery's Multiplication Technique: How to Make it Smaller and Faster. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES99, First International Workshop, Worcester, MA, USA, August 12-13.*, volume 1717, pages 80–93. Springer, 1999.
32. E. Wenger, M. Feldhofer, and N. Felber. Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices. In Y. Chung and M. Yung, editors, *WISA*, volume 6513, pages 92–106. Springer, 2010.

33. E. Wenger and M. Hutter. Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations. In P. Laud, editor, *NordSec 2011, Tallinn, Estonia, October 26-28*, volume 7161 of *LNCS*, pages 256–271. Springer Berlin Heidelberg, 2011.

A Explicit Doubling-Addition Formulæ

In Algorithm 2, the explicit doubling-addition formulæ are given. They are based on Longa and Miris formulæ and require 11M + 7S as well as 7 field registers ($X_1, Y_1, Z_1, R_{1 \ldots 4}$). Availability of *in-place* multiplication is assumed, i.e., the result can overwrite an input operand.

Algorithm 2 Doubling-addition using 11M + 7S and 7 field registers, computes $2P + Q$, with $P = (X_1, Y_1, Z_1)$ and a precomputed comb point $Q = (X_{ROM}, Y_{ROM})$.

```
Require: $X_1$, $Y_1$, $Z_1$, $X_{ROM}$, $Y_{ROM}$
Ensure: $X_1$, $Y_1$, $Z_1$
   1. $R_1 \leftarrow Y_{ROM}$
   2. $R_2 \leftarrow 2Y_{ROM}$
   3. $R_1 \leftarrow R_{1|2} - R_{2|1}$[a]
   4. $R_2 \leftarrow Z^2$
   5. $R_3 \leftarrow Z \times R_2$
   6. $R_1 \leftarrow R_1 \times R_3 - Y$
   7. $R_3 \leftarrow X_{ROM} \times R_2 - X$
   8. $R_4 \leftarrow R_3^2$
   9. $Z \leftarrow Z + R_3$
  10. $R_3 \leftarrow R_4 \times R_3$
  11. $Z \leftarrow Z^2 - R_2 - R_4$
  12. $R_2 \leftarrow X \times R_4$
  13. $Y \leftarrow Y \times R_3$
  14. $R_4 \leftarrow R_1^2$
  15. $X \leftarrow 4R_4 - 4R_3 - 8R_2 - 4R_2$
  16. $R_3 \leftarrow R_1 + X$
  17. $R_3 \leftarrow R_3^2$
  18. $R_1 \leftarrow X^2$
  19. $Y \leftarrow 8Y$
  20. $R_3 \leftarrow R_4 + R_1 - R_3 - 2Y$
  21. $R_2 \leftarrow 4R_2$
  22. $R_2 \leftarrow R_2 \times R_1$
  23. $Z \leftarrow Z \times X$
  24. $R_1 \leftarrow X \times R_1$
  25. $X \leftarrow R_3^2 - R_1 - 2R_2$
  26. $R_1 \leftarrow Y \times R_1$
  27. $R_2 \leftarrow R_2 - X$
  28. $Y \leftarrow R_3 \times R_2 - R_1$
  return $(X_1, X_2, Z)$
```

[a] Dependent on sign of operand Q (cf. Section 5) either R_1 or R_2 is subtracted.