A compact FPGA-based architecture for elliptic curve cryptography over prime fields

Jo Vliegen[*†], Nele Mentens[*†], Jan Genoe[†], An Braeken[‡], Serge Kubera[‡], Abdellah Touhafi[‡§] and Ingrid Verbauwhede[*]
[*]ESAT, SCD/COSIC, Katholieke Universiteit Leuven, 3001 Leuven, Belgium
[†]IWT, ACRO/ES&S, Katholieke Hogeschool Diepenbeek, 3590 Diepenbeek, Belgium
[‡]IWT, Erasmushogeschool Brussel, 1070 Anderlecht, Belgium
[§]Vrije Universiteit Brussel, 1050 Brussel, Belgium
Email: *firstname.lastname*@{esat.kuleuven.be}{iwt.khlim.be}{ehb.be}{vub.ac.be}

***Abstract*—This paper proposes an FPGA-based application-specific elliptic curve processor over a prime field. This research targets applications for which compactness is more important than speed. To obtain a small datapath, the FPGA's dedicated multipliers and carry-chain logic are used and no parallellism is introduced. A small control unit is obtained by following a microcode approach, in which the instructions are stored in the FPGA's Block RAM. The use of algorithms that prevent Simple Power Analysis (SPA) attacks creates an extra cost in latency. Nevertheless, the created processor is flexible in the sense that it can handle all finite field operations over 256-bit prime fields and all elliptic curves of a specified form. The comparison with other implementations on the same generation of FPGAs learns that our design occupies the smallest area.**

***Index Terms*—Application-specific processor; FPGA; elliptic curve cryptography; digital signature;**

I. Introduction

The application field of FPGAs has clearly outgrown the prototype-only use. More and more FPGA implementations are in an environment which used to be ASIC-only territory. When these applications, implemented on an FPGA, need secure data communication, cryptographic components become indispensable. Most standardized cryptographic algorithms were never proven to be theoretically unbreakable, which implies that their security depends upon the fact that they were never broken. Physical attacks, as introduced by Kocher in [8] and [7], are capable of reducing the security level of cryptographic algorithms. For these reasons, updates and upgrades of the implemented cryptographic algorithms can become necessary at any moment in time. In this rapidly changing environment, the reconfigurability of an FPGA is a very useful feature which is not available on an ASIC.

For the majority of products, the cryptographic functionality is not part of the core tasks. Hence it is desirable to use a minimum of the reconfigurable area for these functions. In applications that only occasionally communicate with the outside world, it is acceptable to have a relatively large communication-time overhead in favor of a relatively small area overhead for the implementation of cryptographic components.

One of the most important goals in a secure communication setup is data confidentiality. This can be achieved by using encryption/decryption primitives. Authentication, on the other hand is another important issue of secure communication, which is needed for both the entities and the data.

Many modern FPGAs already have efficient hardware components available to provide data confidentiality with a security level which is appropriate for long-term protection [3]. However, authentication and in particular entity authentication is not a standard feature on an FPGA. To provide authentication, digital signatures are widely used. Algorithms for generating and verifying digital signatures include RSA [14], the Digital Signature Algorithm (DSA) [12] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [12]. According to [5] and [3], ECC provides an equally strong security level with smaller key lengths, in comparison to RSA and DSA. This makes ECDSA a better choice when compact implementations are targeted.

This paper describes the architecture of the most crucial component in ECDSA, namely the one that implements elliptic curve operations and modular operations over a finite field $\mathrm{GF}(p)$. For this purpose, a flexible, yet compact, elliptic curve processor is developed for applications where speed is of minor importance. The processor is optimized for FPGA families that were introduced to the market from the year 2003 on, but that are still used in many new products. These FPGAs also contain some Hard-IPs (HIPs). The presented processor uses Block RAM and multiplier HIPs, which are available on the majority of FPGA devices.

The organization of the paper is as follows. In Sect. II we position the proposed processor among other implementations. Sect. III explains the algorithms which are implemented in our elliptic curve processor. Sect. IV gives a description of the processor architecture and implementation. The results of the implementation are analyzed in Sect. V. Finally, we draw a conclusion and give a glance on future work in Sect. VI.

II. Related Work

Elliptic curve computations consist of operations in the underlying finite field. The most widely used fields for Elliptic Curve Cryptography (ECC) are binary fields and prime fields, denoted by $\mathrm{GF}(2^n)$ and $\mathrm{GF}(p)$, respectively. Because the hardware implementation of binary field operations results in carry-free logic, these fields are the most optimal for use in hardware in terms of speed and area. On the other hand, ECDSA also contains modular operations over GF(p) next to the elliptic curve operations, which makes it interesting to choose for elliptic curves over prime fields in order to be able to share the datapath. Therefore, we focus on ECC over GF(p) with a field size of 256 bits. According to [3], this field size is sufficient to provide long-term protection. The processor is flexible in the sense that it can handle all elliptic curves of the form $y^2 = x^3 + ax + b$, with $a, b \in$ GF(p), and all modular operations over GF(p) for an arbitrary 256-bit prime p.

Most FPGA implementations of ECC over GF(p) are optimized for speed, while the proposed implementation targets compactness, as explained in Sect. I. The design of Örs *et al.* is based on a systolic array architecture for the modular multiplier [13], while Sakiyama *et al.* use a course-grained matrix of modular units to perform the modular multiplication [15]. These implementations do not use any mathematical HIPs, which makes them suitable for any FPGA or ASIC. The implementation of McIvor *et al.* [9] is optimized for the same family of FPGAs as our processor and it also employs the multiplier blocks on the FPGA. The highest speed is achieved by Güneysu and Paar in 2008 [4]. They target newer and more expensive FPGAs that contain DSP blocks. In Sect. V, the performance of the presented design is compared with other implementations on the same generation of FPGAs.

III. Algorithm

As mentioned in Sect. I, ECDSA is a standardized algorithm for generating and verifying digital signatures [12]. Fig. 1 shows the operation hierarchy for ECDSA signature operations.

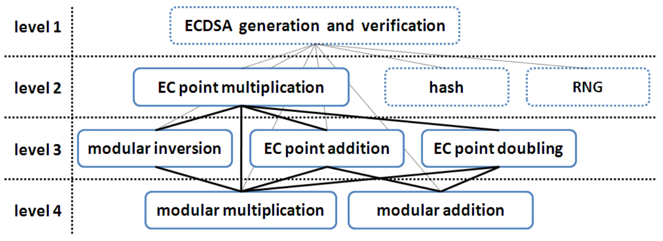

Fig. 1. Operation hierarchy for ECDSA signature generation and verification. The operations surrounded by full lines are executed by our processor. RNG stands for Random Number Generator.

The most critical computation in terms of area and speed, in ECDSA, is an Elliptic Curve Point Multiplication (ECPM). Since only the x-coordinate is used in ECDSA, the Montgomery ladder, shown in Alg. 1, can be used for the computation of an ECPM [11]. Another benefit of this algorithm is the fact that the branches for the keybit being both 0 and 1 are balanced.

Alg. 1 shows that an ECPM can be performed using two lower-level point operations, namely Elliptic Curve Point Addition (ECPA) and Elliptic Curve Point Doubling (ECPD). These two operations are implemented as reported in [6], by Izu *et al.*. They combine an ECPA and an ECPD in one set of formulas (ECPAPD). This requires, however, a conversion to

Algorithm 1 Montgomery ladder [11]

```
Require: point $P$, non-negative integer $k = (1k_{l-2}...k_1k_0)_2$
Ensure: $Q = x(kP)$
  $Q \leftarrow P$, $S \leftarrow 2P$
  for $i = l-2$ downto 0 do
    if $k_i = 1$ then
      $x(Q) \leftarrow x(Q+S)$, $x(S) \leftarrow x(2S)$
    else
      $x(S) \leftarrow x(Q+S)$, $x(Q) \leftarrow x(2Q)$
    end if
  end for
  Return $x(Q)$
```

standard projective coordinates. In general, this can be done by multiplying both coordinates x and y with Z. Conversion back to affine coordinates is done by multiplying the X and Y coordinate with the modular inverse of Z. This modular inverse is calculated by using "Fermat's little theorem", which states: $m^p \equiv m \mod p$ when $\gcd(m,p) = 1$. For all elements m in GF$(p)\backslash\{0\}$ the requirement $\gcd(m,p) = 1$ is satisfied. Using this theorem, an equation for the inverse of a field element can be derived: $m^{-1} \equiv m^{p-2} \mod p$. This shows that the inverse of an element in GF(p) can be computed by a modular exponentiation, which ultimately can be performed by using a series of modular multiplications. The operations in Fig. 1 on level 4 are Modular Multiplication (MM) and Modular Addition/Subtraction (MAS). MAS can be calculated by doing consecutive integer additions and/or subtractions. An efficient way to implement MM is by using Montgomery multiplication [10]. This replaces the trial division, that is necessary in modular multiplication, with a division by a power of two, which comes at no cost in hardware. To use Montgomery multiplication the operands and the product need to be transformed to and from the 'Montgomery representation'.

To summarize Sect. III, one can say that the coordinates for the EC calculations are first converted into Montgomery and standard projective representation. In this representation all necessary MMs and MASs for the ECPAs and ECPDs are calculated, after which the resulting point is converted back to affine coordinates and normal representation.

IV. Processor Architecture

As described in Sect. III, the execution of ECDSA ultimately results in calculating a 'list' of MMs and MASs. Therefore, we designed a processor that can handle these two operations in an efficient way. The general architecture of the implemented processor is shown in Fig. 2.

One could distinguish two memory blocks, the MM and MAS unit, the control unit, the communication unit and some extra registers.

The processor design comprises two memories: the instruction memory and the work memory. To implement these memories, a choice is made to use Block RAM. By combining these Block RAMs in different configurations, our memory

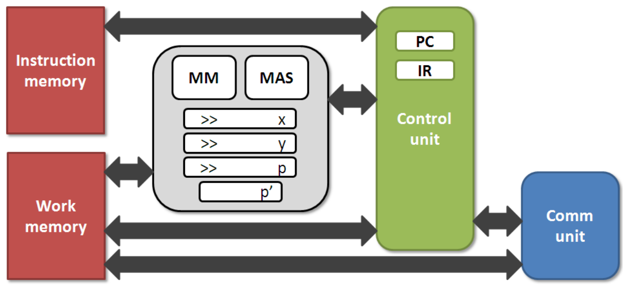

Fig. 2. General processor architecture.

needs can be satisfied. When performing an MM, using a Montgomery multiplication, a final subtraction is needed [10]. In [16], Walter presents a method that avoids this final subtraction. The cost of this solution is that the width of the intermediate values increases with one digit size w, which results in a work memory width of $256 + w$ bits. Therefore, the Block RAMs are combined in a $512 \times (256 + w)$-bit work memory, resulting in an address with a length of 9 bits. Since the instruction set of the processor exists out of 17 instructions, the opcode is 5 bits wide. This results in 32-bit wide instructions and hence a 512×32 bit instruction memory.

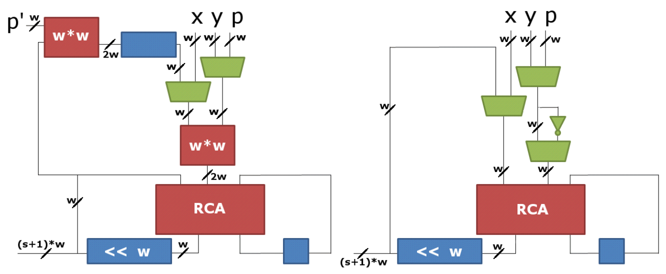

Fig. 3. Datapaths of the MM (left) and the MAS (right).

The MM component follows the algorithm for the implementation of a Montgomery multiplication on a w-bit datapath. This algorithm comes from the work of Koç *et al.* in [1]. In this work, the CIOS method has the smallest space requirements and needs the smallest number of additions and multiplications to calculate one single-precision multiplication. For these reasons, the MM follows this CIOS algorithm. This results in the datapath in Fig. 3 (left) on which the CIOS method, with the improvement of Walter [16], is executed.

The MAS component in Fig. 2 is implemented as shown in Fig. 3 (right). We choose the same digit size w as the modular multiplication, to be able to share the input registers for both components.

In the proposed processor, the usual sequence for instruction handling is followed. To provide the user with a set of programs rather than a single program, the instruction memory is devided into multiple parts. With an offset, given through the communication unit, the user can select which program shall be executed.

The processor is designed to aid secure communication. Next to mathematical attacks, physical attacks exploit imperfections in the implementation. An example of the latter type of attacks is getting information through observation of side channels, among which the most frequently used is power consumption. SPA attacks try to get information by examining the instantaneous power consumption of an implementation while it is running. The function for performing an ECPM provides a countermeasure for SPA on the algorithmic level by using the Montgomery ladder [11]. Also in the MAS component, an effort was made to protect the implementation against SPA attacks by making the number of integer additions and subtractions independent of the input values. Both these countermeasures against SPA come at the cost of slower execution speed. Countermeasures against Differential Power Analysis (DPA) attacks are not implemented yet. Nevertheless, algorithmic countermeasures, such as the ones introduced by Coron in [2], can easily be incorporated.

To communicate with the processor, a 256-bit shift register is implemented. This register can be read from and written to through the use of an external memory, *i.e.* a dual-ported Block RAM memory outside the ECC processor, with one port connected to the communication unit of the ECC processor and the other port to an embedded microprocessor.

Apart from the discussed components above, there are eight extra registers. Two registers are the PC and IR. These two registers have a width of respectively 9 bits and 32 bits. The mathematical components require two operand registers X and Y which they share, as mentioned above. These two shared $(256 + w)$-bit registers are implemented as shift registers, which shift over w positions. When calculating an ECPM, one has to evaluate every bit of the scalar. To implement this, one 256-bit shiftregister to store the scalar and one 8-bit counter to count the number of cycles, are foreseen.

V. Implementation Results

We described the proposed processor in VHDL for a Xilinx VirtexII-Pro XC2VP30 FPGA. Three implementations were made, with different digit length w: 16 bits, 32 bits and 64 bits. The results of these implementations are presented in Table II. We conclude that he maximal clock frequency decreases with an increasing digit size.

	$w = 16$	$w = 32$	$w = 64$
Number of occupied slices	1832	2085	2817
Number of RAMB16s	9	9	10
Number of MULT18X18s	2	7	26
Minimal clock period [ns]	9.242	14.669	39.200
Maximal clock frequency [MHz]	108.20	68.17	25.51

TABLE II
RESOURCES NEEDED AND OBTAINABLE CLOCK SPEED FOR A XC2VP30, WHERE w IS THE DIGIT LENGTH

The latency is measured for some operations and is equal to the number of clock cycles multiplied by the minimal clock period. The cycle counts and latencies of the most important operations are reported in Table III.

A comparison of the implementation of our processor to other implementations (mentioned in Sect. II), on the same

reference	area			max. freq.	ECPM latency	FPGA	remarks
	# slices	# MULTs	# BRAM	(MHz)	(ms)		
[9]	15755	256	0	39.5	3.84	Virtex-II Pro	no comm. unit
[15]	27597	0	0	40	17.7	Spartan-3S-5000	supports 2048-bit RSA, no comm. unit
[13]	5614 (*est.*)	0	0	91.31 (*est.*)	42.52 (*est.*)	Virtex-1000E	160-bit ECC area, no comm. unit
this work	1832	2	9	108.20	29.83	Virtex-II Pro 30	with comm. unit, $w = 16$, with SPA-cm
this work	**1694**	2	9	108.20		Virtex-II Pro 30	without comm. unit, $w = 16$, with SPA-cm
this work	2085	7	9	68.17	15.76	Virtex-II Pro 30	with comm. unit, $w = 32$, with SPA-cm
this work	1947	7	9	68.17		Virtex-II Pro 30	without comm. unit, $w = 32$, with SPA-cm

TABLE I
IMPLEMENTATION COMPARISON WITH THE STATE-OF-THE-ART

Operation	$w = 16$	$w = 32$	$w = 64$
CC_{MAS}	44	28	20
CC_{MM}	637	197	73
CC_{MI}	247253	79141	31757
CC_{ECPAPD}	11621	3853	1601
CC_{ECPM}	3227993	1074625	451733
Latency			
t_{MAS} [ns]	406.65	410.73	784.00
t_{MM} [µs]	5.89	2.89	2.86
t_{MI} [ms]	2.29	1.16	1.24
t_{ECPAPD} [µs]	107.40	56.52	62.76
t_{ECPM} [ms]	29.83	15.75	17.71

TABLE III
CYCLE COUNTS AND LATENCIES OF OPERATIONS, ON A XC2VP30, WHERE w IS THE DIGIT LENGTH

generation of FPGAs, results in Table I. From this table, it can be concluded that the designs of Örs *et al.* [13] and Sakiyama *et al.* [15] result in a comparable latency for one ECPM. Their occupied areas, however, are much larger than the ones of our designs. McIvor *et al.* [9] report a latency that is a little over 4 times faster than the one our implementation achieves for $w = 32$, but the number of slices occupied by their design is approximately 8 times higher. Their number of 18-bit multiplier blocks is more than 36 times higher, but they do not use Block RAM. Note that our designs include a 32-bit communication unit, while the other implementations do not. Therefore, Table I additionaly gives the area of our designs without communication unit. The presence of SPA-countermeasures (SPA-cm) is also mentioned in Table I.

VI. CONCLUSION AND FUTURE WORK

In this paper we proposed an elliptic curve processor, which can be used to perform the finite field operations in an elliptic curve digital signature unit. The processor is optimized for compactness on an FPGA. The implementation uses dedicated multipliers and Block RAMs on the FPGA. Compared to other designs, that are optimized for the same generation of FPGAs, we achieve the most compact solution. This makes our processor suitable for applications where a small area overhead for security is desirable.

Future work includes the evaluation of the performance after the introduction of DPA countermeasures. Furthermore an increase of the digit size w is not beneficial in this architecture. Other methods should be evaluated for $w \geq 64$.

ACKNOWLEDGMENT

Jo Vliegen is funded by the IWT-TETRA project: "Secure Techniques for Remote configuration of Embedded Systems" (80138).

REFERENCES

[1] Ç. K. Koç, T. Acar, and B. S. Kaliski. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.

[2] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç.Koç and C. Paar, editors, *Proceedings of the 1st international workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 1717 in Lecture Notes in Computer Science, pages 292–302. Springer-Verlag, 1999.

[3] D. Giry. Cryptographic key length recommendation, 2009.

[4] T. Güneysu and C. Paar. Ultra high performance ECC over NIST primes on commercial FPGAs. In E. Oswald and P. Rohatgi, editors, *Proceedings of the 10th international workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 5154 in Lecture Notes in Computer Science, pages 62–78. Springer-Verlag, 2008.

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

[6] T. Izu, B. Möller, and T. Takagi. Improved Elliptic Curve multiplication methods resistant against Side Channel Attacks. In *Progress in Cryptology (IndoCrypt)*, number 2551 in Lecture Notes in Computer Science, pages 296–313. Springer-Verlag, 2002.

[7] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. J. Wiener, editor, *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)*, number 1666 in Lecture Notes in Computer Science, pages 388–397. Springer-Verlag, 1999.

[8] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In M. J. Wiener, editor, *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)*, number 1109 in Lecture Notes in Computer Science, pages 104–113. Springer-Verlag, 1996.

[9] C. McIvor, M. McLoone, and J.V. McCanny. An FPGA Elliptic Curve Cryptographic accelerator over GF(p). *IEE Conference Publications*, 2004(CP506):589–594, 2004.

[10] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[11] P. L. Montgomery. Speeding the Pollard and Elliptic Curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[12] National Institute of Standards and Technology. FIPS 186-3: Digital Signature Standard, 2009.

[13] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle. Hardware implementation of an Elliptic Curve Processor over GF(p). In *In 14th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 433–433. IEEE Computer Society, 2003.

[14] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[15] K. Sakiyama, N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede. Reconfigurable modular arithmetic logic unit supporting high-performance RSA and ECC over GF(p). *International Journal of Electronics*, 94(5):501–514, 2007.

[16] C. D. Walter. Montgomery exponentiation needs no final subtraction. *Electronic letters*, 35(21):1831–1832, 1999.