2011 International Conference on Reconfigurable Computing and FPGAs

HARDWARE DESIGN OF A 256-BIT PRIME FIELD MULTIPLIER SUITABLE FOR COMPUTING BILINEAR PAIRINGS

Cuauhtémoc Chávez Corona
Departamento de Computación
CINVESTAV-IPN
cchavez@computacion.cs.cinvestav.mx

Edgar Ferrer Moreno
Turabo University, Gurabo, PR
edgferrer@suagm.edu

Francisco Rodríguez Henríquez
Departamento de Computación
CINVESTAV-IPN
francisco@cs.cinvestav.mx

***Abstract*—We present a hardware-oriented architecture able to compute a 256-bit prime finite field multiplication efficiently. Taking advantage of the Karatsuba algorithm, the proposed architecture splits a 256-bit integer multiplication into fourteen 64-bit sub-products plus a number of additions that are performed using parallel and pipelined arrangements. The resulting 512-bit partial product is reduced into a 256-bit integer using a polynomial variant of the Montgomery reduction algorithm. The multiplier architecture presented here can be directly adapted for computing bilinear pairings over Barreto-Naehrig curves. In order to improve the performance of our design, the architecture makes use of twelve *DSP48 slices*, which are high-performance built-in blocks available in the Xilinx Virtex-6 family of FPGA devices.**

***Keywords*-Modular multiplication, cryptography, bilinear pairing, finite field arithmetic, digital design, FPGAs.**

I. INTRODUCTION

Modular multiplication is a fundamental operation for modern cryptographic schemes. It is an essential operation for carrying out modular exponentiation in public key cryptosystems based on discrete logarithms and for the RSA encryption scheme [13]. Moreover, prime field multiplication is an important arithmetic operation required for computing elliptic curve scalar multiplications and bilinear pairings.

In recent years, bilinear pairings have become of great research interest in the cryptographic community. Cryptographic schemes based on bilinear pairings require pairing-friendly elliptic curves, and the Barreto-Naehrig (BN) curves, which were introduced in [2], have emerged as a popular choice. Bilinear pairings operating over BN curves can achieve approximately 128-bit security level when employing 256-bit primes for defining the underlying prime field where the associated arithmetic operations should be performed. BN curves are defined over a finite field generated by a prime p that can be written in a parameterized form as,

$$p = p(t) = 36t^4 + 36t^3 + 24t^2 + 6t + 1, \qquad (1)$$

where t is an arbitrary integer such that $p(t)$ is a prime.

We stress that all operands required in a pairing computation must be in the field F_p, and any integer value for t can be chosen as long as the condition that p is prime as well as other restrictions stated in [2] are met.

This paper describes an efficient architecture for computing prime field multiplication, the most time consuming operation of bilinear pairings defined over ordinary elliptic curves.

Due to its high flexibility and speed capabilities, the target device for our implementation is a Field Programmable Gate Array (FPGA) device. The enhanced DSP slices and memory blocks present in the latest models of FPGA devices are used to develop an efficient hardware architecture for the field multiplier [11], [16]. The architecture so obtained is then compared with existing state of the art solutions to gain a measure of the overall efficiency of our FPGA-based solution. The designs presented in this work were modeled using VHDL, and the ModelSim SE 6.3 and Xilinx ISE 10.1 tools were used for synthesis and place-and-route simulations.

The remainder of this paper is organized as follows. Section II provides a short overview of existing work concerning efficient arithmetic for bilinear pairings over BN curves. Our approach for modular multiplication is introduced in section III. Section IV describes the realization of an efficient 64-bit multiplier which is a crucial building block of our scheme. Main design details of the proposed architecture are given in Section V, while implementation results and performance comparison are given in Section VI. We close in Section VII with some concluding remarks.

II. RELATED WORKS

Several efforts for computing bilinear pairings efficiently have been reported both in hardware [5] and software [3] platforms. However, in the case of hardware accelerators these works have been mainly focused on elliptic curves over binary and ternary extension fields [3], [8], [17]. These curves have become popular due to carry-free computations and bit-wise operations that can be easily applied to the arithmetic over these two kinds of finite fields. With this hardware-friendly arithmetic, fast and space-saving designs can be achieved. But this advantage in terms of speed of computation is paid off by using a large number of bits in order to achieve an appropriate level of security.

978-0-7695-4551-6/11 $26.00 © 2011 IEEE
DOI 10.1109/ReConFig.2011.46

This is in contrast with prime fields associated to BN curves, where a comparatively more complex arithmetic must be computed but where same levels of security can be achieved with smaller field extensions.

The first hardware implementation of bilinear pairings using Barreto-Naehrig curves was reported as recently as 2009 [12]. However, the design proposed there happened to be slower than software-based implementations as the one reported in [4]. In the same year, a new hardware-oriented multiplier was presented in [9]. Although this multiplier did not improve the timings reported by software-based implementations, it considerably lowered the cost in terms of the time and space that were reported in the first hardware implementation of [12]. This novel multiplier was developed on ASIC (Application-Specific Integrated Circuit) technology, which tends to be faster than the reconfigurable technology, although the implementation process is more time-consuming and complex.

The main idea behind the aforementioned multiplier is to treat field elements as polynomials in the variable t of (1), and to use the polynomial version of the Montgomery multiplication algorithm. The prime p defining the field is obtained as in (1), and all the elements in the field which are less than p can be represented in polynomial form as,

$$a = a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0$$

It is noticed that whereas the bit-length of a is 256 bits, the bit-length of the coefficients a_3, a_2, a_1, a_0 is of 64 bits. On the other hand, since the maximum value it can reach is 36, a_4 has at most 7 bits. Due to this transformation, a 512-bit product can be obtained by carrying out several 64-bit multiplications which, in principle, are faster and easier to handle. The above approach can be realized as shown in Algorithm 1.

Algorithm 1 Prime field multiplication proposed in [9]

Require: $at^4 \bmod p = a(t) = \sum_{i=0}^{4} a_i t^i$,
$bt^4 \bmod p = b(t) = \sum_{i=0}^{4} b_i t^i$,
$p(t) = 36t^4 + 36t^3 + 24t^2 + 6t + 1$.
Ensure: $a(t)b(t)t^4 \bmod p$
1: $c(t)(= \sum_{i=0}^{4} c_i t^i) \leftarrow 0$
2: **for** $i = 0$ to 4 **do**
3: $c(t) \leftarrow c(t) + a(t)b_i$
4: $\mu \leftarrow c_0$ div t; $\gamma \leftarrow c_0 \bmod t$
5: $g(t) \leftarrow p(t)(-\gamma)$
6: $c(t) \leftarrow (c(t) + g(t))/t + \mu$
7: **end for**
8: **for** $i = 0$ to 3 **do**
9: $c_{i+1} \leftarrow c_{i+1} + (c_i$ div $t)$; $c_i \leftarrow c_i \bmod t$
10: **end for**
11: **return** $c(t)$

Algorithm 1 takes advantage of the special form of the prime p as given in (1). It is noticed that this prime is obtained from a polynomial with independent term 1 so that $p \bmod t = 1$, which implies, $p^{-1} \bmod t = 1$. Notice that Algorithm 1 includes division and reduction by t. However, as noted in [9], these potentially costly operations can be efficiently computed by performing a careful selection of the parameter t, such that $t = 2^n + s$ where n is approximately 64 and s should be selected as a small number with low Hamming weight. Under this scenario, the modulo and division can be performed by applying two times the following operations:

$$\mu \leftarrow c_i \; div \; 2^n; \gamma \leftarrow c_i \; mod \; 2^n - \mu s,$$

where modulo and division by 2^n are operations that can be carried out efficiently.

III. A Karatsuba-based Modular Multiplication

This section introduces the adopted approach for performing modular multiplication, which is based on the multiplier described in the above section but with some subtle modifications.

The general structure was directly taken from [9], although the algorithm was improved by reducing the number of required sub-products. This reduction of operations became possible by shifting the multiplication strategy to a 5-term Karatsuba-based multiplier [1]. There exists a 5-term version of the Karatsuba multiplication algorithm developed by Montgomery in [14], which requires only 13 products, but it uses a very high number of additions of the partial results and due to the lack of obvious patterns in the algorithm flow, the design appears to be not hardware-friendly. Because of these reasons, a slightly modified Karatsuba-based multiplier was used. Our scheme computes a 256-bit product by performing 14 sub-products and lesser additions than the multiplier in [14]. Furthermore, due to its regular structures it turns out that this algorithm is more hardware-friendly than the work presented in [14]. Taking advantage of the aforementioned strategies, we developed an original approach for modular multiplication as described in Algorithm 2.

Algorithm 2 performs a 256-bit modular multiplication by computing polynomial multiplication followed by Montgomery polynomial reduction. On the other hand, one can decrease the number of additions in the polynomial reduction if we use the appropriate parameter t. For instance, by choosing the parameter $t = 2^n - s$, with $n = 61$ and $s = -2^{15} - 1$, one can achieve better polynomial reduction results.

A. The Karatsuba Multiplier

The Karatsuba multiplier uses the *divide-and-conquer* approach for breaking down the main multiplication problem into several simpler problems. Thus, the problem of multiplying a 256-bit number (represented as a polynomial of degree 4) is split into several 64-bit products, to be

Algorithm 2 Our Algorithm for Modular Multiplication

Require: $at^4 \bmod p = a(t) = \sum_{i=0}^{4} a_i t^i$,
$bt^4 \bmod p = b(t) = \sum_{i=0}^{4} b_i t^i$,
$p(t) = 36t^4 + 36t^3 + 24t^2 + 6t + 1$, $t = 2^n + s$
Ensure: $a(t)b(t)t^4 \bmod p$
1: $c(t)(= \sum_{i=0}^{8} c_i t^i) \leftarrow 0$
2: $c(t) = 5 - term - Karatsuba(a(t), b(t))$
3: **for** $i = 0$ to 4 **do**
4: $\mu \leftarrow c_0$ div 2^n; $\gamma \leftarrow c_0 \bmod 2^n - \mu s$
5: $g(t) \leftarrow p(t)(-\gamma)$
6: $c(t) \leftarrow (c(t) + g(t))/t + \mu$
7: **end for**
8: **for** $i = 0$ to 3 **do**
9: $\mu \leftarrow c_i$ div 2^n; $\gamma \leftarrow c_i \bmod 2^n - \mu s$
10: $c_{i+1} \leftarrow c_{i+1} + \mu$; $c_i \leftarrow \gamma$
11: **end for**
12: **return** $\sum_{i=0}^{4} c_i t^i$

exact, 14 partial 64-bit products. If we had used the school-book method, 25 multiplications would have been needed. However, as it has been mentioned above, Karatsuba product savings imply the usage of many adders and registers to store partial products. A modified Karatsuba approach more amenable for hardware implementations is presented in Algorithm 3.

Notice that a relatively high number of additions are required in Algorithm 3. However, these additions show a particular pattern that can be exploited. Using a careful scheduling, a new product can be obtained every 15 clock cycles, unlike the algorithm presented in [9] which requires 23 clock cycles to obtain a new product. In order to compute each coefficient in the final result, at least a new product per clock cycle must be processed so that the multiplier processor is kept always busy.

IV. The 64-bit Multiplier

The most important component in the proposed implementation is a 64×64-bit multiplier. Since the target device is a Virtex 6 FPGA device, one can construct a 64-bit multiplier by using the *DSP48* slices as 18×18 multipliers, but in this case the higher bits in the first operand of the multiplier would be wasted. This issue would also complicate the extra manipulation of bits necessary for computations of partial results in the FPGA fabric. An alternative to face these inconveniences is presented in [18]. The underlying idea is to build large multipliers using asymmetric *DSP48* slices while the small products are arranged in a diamond-looking pattern.

Hence, a 64-bit product can be achieved by performing nine sub-products that can be rearranged as is shown in Figure 1. This structure obtains a 64-bit multiplication requiring only 4 additions, since the other additions are performed by simply concatenation of the partial products.

Algorithm 3 Five-Term Karatsuba Multiplier

Require: $a(t) = \sum_{i=0}^{4} a_i t^i$, $b(t) = \sum_{i=0}^{4} b_i t^i$
Ensure: $c(t) = a(t)b(t)$
1: $c(t)(= \sum_{i=0}^{8} c_i t^i) \leftarrow 0$;
2: $p_0 = a_0 b_0$;
3: $p_1 = a_1 b_1$;
4: $p_2 = (a_0 + a_1)(b_0 + b_1)$;
5: $p_3 = a_2 b_2$;
6: $p_4 = (a_0 + a_2)(b_0 + b_2)$;
7: $p_5 = a_3 b_3$;
8: $p_6 = (a_2 + a_3)(b_2 + b_3)$;
9: $p_7 = (a_1 + a_3)(b_1 + b_3)$;
10: $p_8 = (a_0 + a_1 + a_2 + a_3)(b_0 + b_1 + b_2 + b_3)$;
11: $p_9 = a_4 b_4$;
12: $p_{10} = (a_0 + a_4)(b_0 + b_4)$;
13: $p_{11} = (a_0 + a_1 + a_4)(b_0 + b_1 + b_4)$;
14: $p_{12} = (a_2 + a_4)(b_2 + b_4)$;
15: $p_{13} = (a_2 + a_3 + a_4)(b_2 + b_3 + b_4)$
16: $c_0 = p_0$
17: $c_1 = p_2 - p_1 - p_0$
18: $c_2 = p_4 + p_1 - p_0 - p_3$
19: $S1 = p_6 - p_5 - p_3$
20: $c_3 = p_8 - p_7 - p4 - c_1 - S1$
21: $c_4 = p_{10} - p_9 - p_0 + p_3 + p_5 - p_1 + p_7$
22: $c_5 = p_{11} - p_1 - p_{10} - c_1 + S1$
23: $c_6 = p_{12} - p_9 + p_5 - p_3$
24: $c_7 = p_{13} - p_5 - p_{12} - S1$
25: $c_8 = p_9$
26: **return** $c(t)$

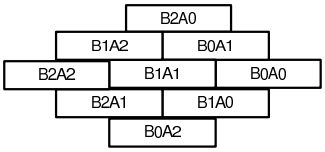

Figure 1. Rearrangement of products from the school-book method

This method is also effective for asymmetric multipliers. For instance, the asymmetric operands with 24 and 17 bits can be performed by the described method,[1] in this way, the operands can be divided as is depicted in Figure 2, and these operands can be rearranged according to the structures presented in Figure 3. Notice that the final 64-bit multiplication requires 12 products and 5 additions, the result is completed by concatenations over the operands.

[1] although *DSPslices* can be programmed as 25×18 multipliers, we took the design decision of reserving the most significant bit of each operand for defining the operand's sign.

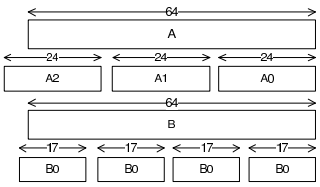

Figure 2. Asymmetric division of operands

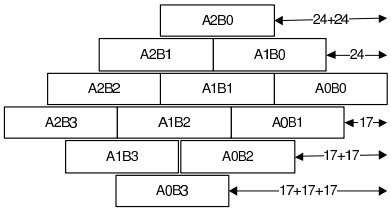

Figure 3. Arrangement of the operands in our design

V. Architecture

In the preceding Sections, the basic components of the architecture required to implement Algorithm 2 were described. The coefficients of the operands a and b are received as inputs and then transformed into a polynomial form of the Montgomery product [6]. As depicted in Figure 4, the architecture is divided into three distinct architectural sections: the polynomial products (5-term Karatsuba), polynomial reduction (the number of coefficients is reduced from 9 to 4), and the reduction of coefficients, which could produce an extra coefficient in some cases. After these three transformations the final result of the product is a polynomial with 5 coefficients.

A. Product of Polynomials

Figure 5 depicts the input component of the architecture which consists of two *DualPortRam* memory blocks for storing the input operands of the multiplier. There are adders and registers between the memories and the multiplier, this is for performing the set of additions prior to each product (lines 4-15 of Algorithm 3) and storing the results of additions to be reused later. In this way, the critical path gets reduced and consequently the operating frequency gets increased.

1) The Multiplier: The multiplier was implemented according to the method described in the previous Section. As Figure 6 shows, all the sub-products are accomplished in parallel exploiting the *DSP48* slices. Afterwards, the products are unified by means of 5 additions that use a pipelined architecture. In this fashion, the multiplier gives a new product every clock cycle with an associated latency of 4 clock cycles.

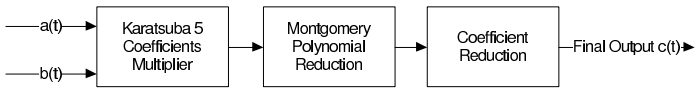

Figure 4. Block diagram for the proposed design

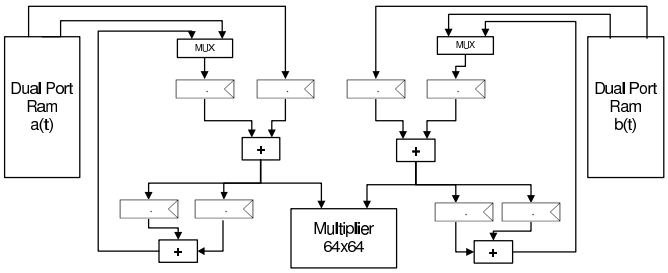

Figure 5. First phase of the architecture

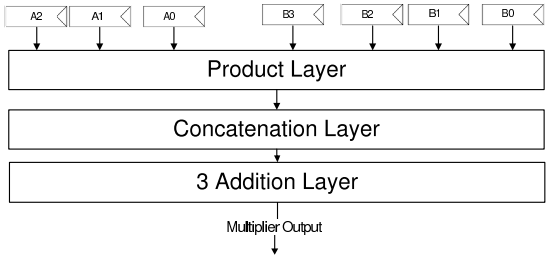

Figure 6. Implementation of the 64×64-bit multiplier

2) The Adders: This is a complicated and demanding part of the design presented in this work. This portion of the architecture processes every product produced, in such a way that when the multiplier delivers the 14 sub-products, an output with 9 coefficients corresponding to the resulting polynomial is obtained. In this way, the 64-bit multiplier block is kept busy at all times. One important component of this part of the architecture, is the one responsible of carrying out one addition and two subtractions that correspond to the basic operations inherent to the Karatsuba algorithm. Other relevant component contains two accumulators for supporting long additions, and finally, another essential component is the bank of registers used for storing temporary variables and operation results.

This phase of the architecture is especially demanding because it deals with 128-bit operands, whereas in the rest of the phases only 64 and 80-bit elements are manipulated. The overall result is depicted in Figure 7. Notice that this part contains the full critical path, which goes from the register **Rac2o** to **Ri1**, i.e. the critical path goes from a 4-input multiplexer to an adder, both of them handling 128-bit operands.

B. Polynomial Reduction

The architecture shown in Figure 8 describes the polynomial reduction phase. This architecture can reduce coefficients one by one without having the whole polynomial computed. It takes 3 clock cycles to reduce each coefficient, i.e., 15 clock cycles are needed to complete the entire reduction.

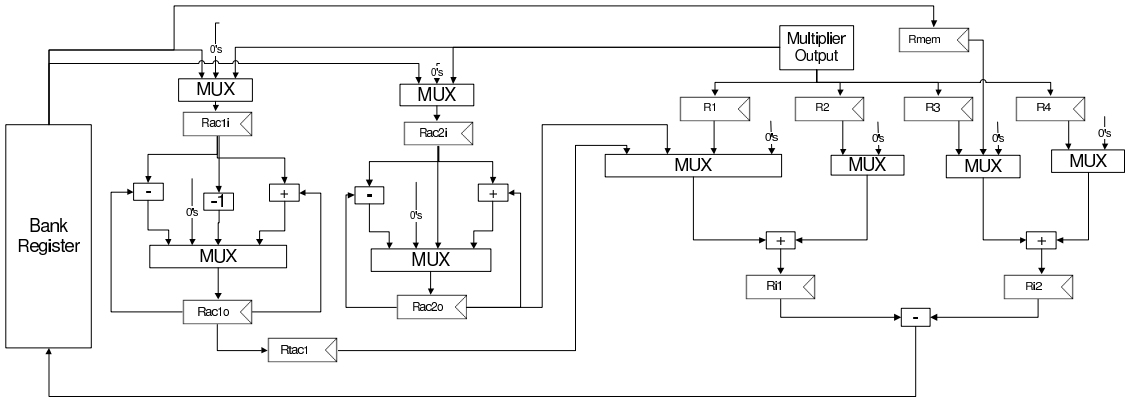

Figure 7. Additions in the Karatsuba Algorithm

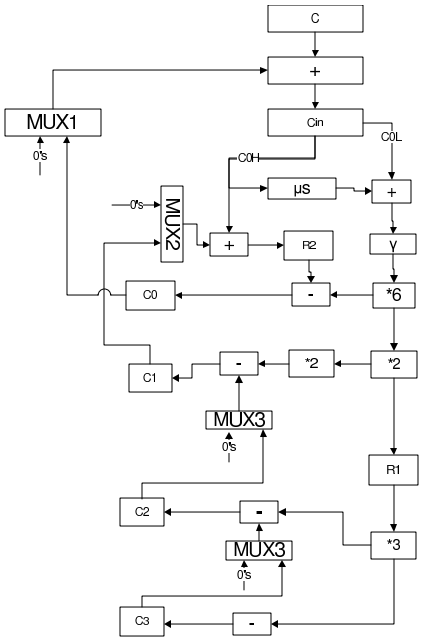

Figure 8. Polynomial Reduction

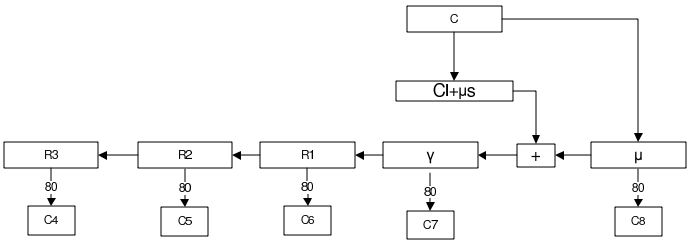

Figure 9. Coefficient Reduction

C. Coefficient Reduction

This final step has to be performed to avoid that the polynomial coefficients get greater than t, this assures that the resulting polynomial can be used in successive iterations of the multiplication algorithm. In order to save hardware resources, the coefficient reduction is implemented sequentially, since it is not necessary to make this operation faster because it works in parallel with the modules presented in the previous subsections. The design of the circuit for coefficient reduction is shown in Figure 9.

VI. Results

Table I shows that the proposed design provides one 256-bit field multiplication every 15 clock cycles, with an associated latency of 40 clock cycles. This performance is competitive with other similar works published in the literature. Recent efforts such as the ones presented in [10] and [19] report high performance 256-bit prime field multipliers suitable for pairing computations. Moreover, [19] takes a new direction, presenting a multiplier based on RNS arithmetic that requires more resources, but achieves higher frequencies. [2]

Table I
Characteristics of implementation in each phase

Component	Latency	Frequency (MHz)
Multiplier Input	3	500
64 x 64 multiplier	4	313
Additions	18	210
Montgomery Reduction	18	270
Coefficient Reduction	15	240
Ending	40	210

[2] An updated version of [19] was recently published in [7].

Table II
Comparisons with recent implementations

Criterion	[9]	This work	[10]	[19]
Frequency (Mhz)	204	223.7	210	250
Cycles per Product	23	15	5	15
Latency	0	40	25	8
Device	ASIC	Virtex6	Virtex6	Virtex6
Area	183 kGates	4815 Slices	4014	-
DSP48 Slices	-	12/48	46/288	36/2014

VII. Conclusions

We have presented an architecture that computes 256-bit prime field multiplications. Our implementation is faster than the original multiplier on which our approach is based [9]. However, our multiplier still uses more FPGA resources, especially in the modules that compute the additions associated to the Karatsuba multiplier, which besides handling very large operators is very demanding in terms of the amount of data to be processed. A worthy alternative to be exploited in the future is to use *DSP48* slices for performing the most expensive operations present in our architecture. Since the cutting-edge models of FPGAs contain abundant *DSP48* slices, we could use these logic elements to accelerate other operations besides the field multiplication. Other options include to adopt the techniques reported in [15], that support carry and carry less additions at the price of employing more hardware resources.

References

[1] A. V. Aho and J. E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1974.

[2] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography - SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.

[3] J.-L. Beuchat, J. Detrey, N. Estibals, E. Okamoto, and F. Rodríguez-Henríquez. Fast architectures for the ηt pairing over small-characteristic supersingular elliptic curves. *IEEE Transactions on Computers*, 99(PrePrints), 2010.

[4] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over barreto-naehrig curves. In M. Joye, A. Miyaji, and A. Otsuka, editors, *Pairing-Based Cryptography - Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2010.

[5] J.-L. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari, and F. Rodríguez-Henríquez. Multi-core implementation of the Tate pairing over supersingular elliptic curves. In *Proceedings of the 8th International Conference on Cryptology and Network Security*, CANS '09, pages 413–432, Berlin, Heidelberg, 2009. Springer-Verlag.

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

[7] R. C. C. Cheung, S. Duquesne, J. Fan, N. Guillermin, I. Verbauwhede, and G. X. Yao. Fpga implementation of pairings using residue number system and lazy reduction. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 421–441. Springer, 2011.

[8] N. Estibals. Compact hardware for computing the Tate pairing over 128-bit-security supersingular curves. In M. Joye, A. Miyaji, and A. Otsuka, editors, *Pairing-Based Cryptography - Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 397–416. Springer, 2010.

[9] J. Fan, F. Vercauteren, and I. Verbauwhede. Faster F_p-arithmetic for cryptographic pairings on Barreto-Naehrig curves. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 240–253. Springer-Verlag, 2009.

[10] J. Fan, F. Vercauteren, and I. Verbauwhede. Efficient hardware implementation of F_p-arithmetic for pairing-friendly curves. *Computers, IEEE Transactions on*, PP(99):1, 2011.

[11] T. Güneysu. Utilizing hard cores of modern FPGA devices for high-performance cryptography. *Journal of Cryptographic Engineering*, 1:37–55, 2011.

[12] D. Kammler, D. Zhang, P. Schwabe, H. Scharwächter, M. Langenberg, D. Auras, G. Ascheid, and R. Mathar. Designing an ASIP for cryptographic pairings over Barreto-Naehrig curves. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 254–271. Springer, 2009.

[13] A. J. Menezes, P. C. V. Oorschot, S. A. Vanstone, and R. L. Rivest. Handbook of applied cryptography, 1997.

[14] P. Montgomery. Five, six, and seven-term Karatsuba-like formulae. *Computers, IEEE Transactions on*, 54(3):362–369, 2005.

[15] H. D. Nguyen, B. Pasca, and T. Preußer, B. FPGA-Specific Arithmetic Optimizations of Short-Latency Adders. available at: http://hal-ens-lyon.archives-ouvertes.fr/ensl-00542389/PDF/short_latency_adders.pdf.

[16] F. Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pérez, and C. K. Koc. *Cryptographic Algorithms on Reconfigurable Hardware (Signals and Communication Technology)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[17] C. Shu, S. Kwon, and K. Gaj. Reconfigurable computing approach for Tate pairing cryptosystems over binary fields. *IEEE Trans. Computers*, 58(9):1221–1237, 2009.

[18] S. Srinath and K. Compton. Automatic generation of high-performance multipliers for FPGAs with asymmetric multiplier blocks. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '10, pages 51–58, New York, NY, USA, 2010. ACM.

[19] G. X. Yao, J. Fan, R. C. Cheung, and I. Verbauwhede. A high speed pairing coprocessor using RNS and lazy reduction. Cryptology ePrint Archive, Report 2011/258, 2011. http://eprint.iacr.org/.