HW/SW Co-Design of Identity-Based Encryption using a Custom Instruction Set

Leonardo Amaral[#1], Guido Araujo[#2], Julio López[#3]

[#] *Institute of Computing, University of Campinas,*

Cidade Universitária Zeferino Vaz P. O. Box 6176 Campinas-SP, Brazil

[1] lsamaral.ios@gmail.com

[2] guido@ic.unicamp.br

[3] jlopez@ic.unicamp.br

Abstract—**Identity-Based Cryptography has been gradually accepted as a more effective way of implementing asymmetric cryptography. The calculation of cryptographically-suitable pairings is crucial for the performance of pairing based protocols. In this paper we present a comparative study of hardware implementation techniques for computing the η_T pairing over the finite field $\mathbb{F}_{3^{97}}$. Our hardware-software implementation use Altera Nios II processor as platform. Using code profiling we identify critical field operations which concentrate most of the execution time; then these operations were implemented as specialized FPGA instructions/modules and added to the processor. The specialized processor was synthesized and the application was tailored to the new hardware. Experimental results show that a considerable speedup can be achieved when compared to the baseline software only approach. Moreover, we show that such HW/SW co-design approach is competitive with other solutions.**

I. Introduction

Public key cryptography, as introduced by Whitfield Diffie and Martin Hellman in 1976 [1], has been largely used in computer security applications since its inception. Although effective, this scheme presents some restrictions like the need of a complex digital certificate infrastructure administration to assure the identity of the entities at both ends of the communication channel.

In 2001, Boneh and Franklin [2], introduced the use of pairings for Identity-Based Encryption. In this new asymmetric cryptography scheme, the user public key is some information that uniquely identifies the user, for example, the driver's license number or electronic address (e-mail). This type of public-key cryptography is known as Identity-Based Cryptography, and was initially proposed by Shamir in 1994 [3] (using a signature scheme).

Pairing computation is the most computational intensive primitive in Identity-Based Cryptography; therefore efficient implementation of pairing algorithms are vital to the overall system performance of pairing based protocols. More details about the pairing over elliptic curves can be found in [4][5].

In this paper we choose as candidate the η_T pairing (proposed by Barreto *et al* in [4]) and the algorithms proposed by Beuchat *et al* in [5]. The pairing computation can be decomposed into a series of smaller field arithmetic operations in $\mathbb{F}_{3^m}$, including: addition, subtraction, multiplication and cubing. In order to evaluate the computation effort of each operation, execution time profiling of a baseline software implementation was performed, and used to guide a careful FPGA hardware specialization of some critical operations. A generic HW/SW solution was also designed. The experimental results reveal that the proposed approaches can produce a considerable speedup, when compared to the software only solution, without the design effort/cost and inflexibility of a hardware only approach.

II. Related Work

Some techniques have been proposed to deal with Identity-Based Cryptography, most of which follows a purely hardware a [5][6][7] or purely software [8],[9],[10] approach. Most purely hardware approaches adopt the whole pairing implementation in hardware, and are usually designed as expensive hardware security proprietary modules or co-processors.

A purely hardware approach achieves the best performance. However, due to their high design/production costs and lack of flexibility their adoption in real world scenarios is more difficult. If a specialized system needs to compute thousands of pairings in a second, then this approach becomes necessary. Moreover, it offers an increment in security, since cryptanalysis attacks directly to the hardware is a complex and expensive process.

The purely software approach (which does not use any a custom instruction or specialized hardware) is slower when compared to the purely hardware. However, it offers a very low cost solution combined with higher flexibility.

In this work we explore a third alternative, based on a mix of hardware/software modules, in which software pairing counts with the help of specialized hardware modules for compute critical parts of the pairing. Such approach is subdivided into two variations. In the first one, referred from now on as specialized hardware/software, the hardware possesses highly specialized instructions or modules for small parts resolution of the pairing. In the second (in this work we will refer to it as generic hardware/software approach), the hardware possesses some generic instructions that can be used in the critical parts acceleration of the pairing. As shown later the first approach presents a slightly superior performance to second, while the second present a much larger flexibility than the first.

In this paper we designed pairing solutions using generic hardware/software and specialized hardware/software approaches. We measured the HW/SW solutions speedups with respect to a software only solution, and compared its cost and flexibility with respect to hardware only architectures.

III. The Platform

The design of a combined HW/SW solution to pairing requires a flexible FPGA platform which could allow an easy mapping of program modules into specialized hardware modules. The platform used herein was an Altera Nios II [11] development kit based on a Stratix II FPGA [12]. Nios II is an Altera 32-bit processor which enables the tailoring of new instructions into its instruction set, by means of a very flexible design environment. It has also an Avalon data bus which allows the insertion of new hardware modules directly to the processor bus.

Such platform is supported by a specialized Altera Quartus II toolchain [13] and *GCC* compiler, capable of generating optimized NIOS II code starting from *C/C++* applications. The VHDL was used in the modeling of specialized hardware modules. The *SOPC Builder* tool, which is integral part of Altera Quartus II, was used to enable the insertion of new hardware directly into the processor. That tool generates automatically *C* libraries that contain the signatures of the calls used to access the specialized hardware which is added to the processor, simplifying thus the access to the instructions/modules.

IV. Cryptography features of the System

In this paper, we chose η_T as the pairing algorithm and the scheme proposed by Beuchat *et al* in [5]. We have selected the adaptation of the cube-root-free reversed-loop algorithm, proposed in [5] as the basis to our pairing implementation. This algorithm presents some characteristics that can be observed in Table I.

The selected pairing algorithm was implemented using *C/C++* and, its execution was profiled using the *Oprofile* [14] tool in a *Pentium 4* processor. The goal of this experiment was to identify the sub-group of functions which was responsible for the largest share of the pairing execution. With this distribution, it is possible to pin point the best functions to be accelerated in hardware, when using a combined hardware/software approach.

Analyzing the results of the *Oprofile* execution one can identify that the two largest contributors to the pairing running time are the polynomials multiplication function over $\mathbb{F}_3{}^m$ (~96%) and the cubing function over $\mathbb{F}_3{}^m$ (~2%). Those functions sum together approximately 98% of the execution time required to compute the pairing over elliptic curves. In order to improve pairing performance, such operations were implemented in hardware.

V. Specialized Hardware/Software Approach

As mentioned in the previous sections, the field operations of multiplication and cubing in $\mathbb{F}_3{}^{97}$ were selected to be implemented in hardware. This was done using VHDL as a hardware description language. For such task two different approaches were defined according to the features of each function.

The polynomial cubing over $\mathbb{F}_3{}^{97}$ was implemented as a new instruction into the NIOS II processor instructions set. This architecture decision was taken due to the fact of the cubing function be a relatively small combinational logic, for which the overall latency could be accommodated within the execution stage of the processor pipeline, without impacting its clock cycle.

On the other hand, polynomial multiplication in $\mathbb{F}_3{}^{97}$ was implemented as a module attached to the processor data bus, given the size and latency of the chosen algorithm.

TABLE I
CRYPTOGRAPHY CHARACTERISTICS

Characteristic	Used Value
Underlying Field	$\mathbb{F}_3{}^m$, where m is coprime to 6 and m is 97
Curve	E: $y^2 = x^3 - x + 1$, b = 1
Irreducible polynomial of degree m	$f(x) = x^{97} + x^{12} + 2$
Number of rational points of Curve	190880563234078270754244862876156026926706 48963
Embedding degree	k = 6
ℓ	272686518905826101077496079813497618717146 2721
Distortion map	$\psi : E(\mathbb{F}_3{}^{97})[\ell] \rightarrow E(\mathbb{F}_3{}^{6*97})[\ell] \setminus E(\mathbb{F}_3{}^{97})[\ell]$ $(x, y) \mapsto (\rho - x, y\sigma)$ with $\rho \in \mathbb{F}_3{}^{3*97}$ and $\sigma \in \mathbb{F}_3{}^{2*97}$ satisfying $\rho^3 = \rho + 1$ and $\sigma^2 = -1$
Tower Field	$\mathbb{F}_3{}^{6*97} = \mathbb{F}_3{}^{97} [\rho, \sigma] \cong \mathbb{F}_3{}^{97}$ [X, Y] / (X^3 – X – b, Y^2 + 1)
Final Exponentiation	$M = (3^{3*97}) \cdot (3^{97} + 1) \cdot (3^{97} + 1 - \mu 3^{(97+1)/2})$
Parameters	λ and $v = -1$

A. *Cubing in $\mathbb{F}_3{}^{97}$*

As described in [5] cubing operation over $\mathbb{F}_3{}^{97}$ consists of computing the expression below and to reduce it by an irreducible polynomial $f(x)$ of degree m.

$$\mathrm{d}(x)^3 \bmod f(x) = \sum_{i=0}^{m-1} \mathrm{d}_i x^{3i} \bmod f(x)$$

The hardware implementation of this expression can be obtained by applying loop-unrolling in the given expression. The computation of each result element is achieved by summing up the polynomial coefficients over F_3, what can be done in parallel.

As said before, this operation was implemented as a custom instruction in the NIOS II processor. The new instruction receives two 32-bit operands, and an 8-bit (n) parameter which is passed to the call instruction to select which hardware sub functions will be used. The new instruction returns a 32-bit result, which corresponds to a fraction of the resulting polynomial. In our system each element of $\mathbb{F}_3$ is represented as two bits in hardware. In other words, an element in $\mathbb{F}_3{}^{97}$ is represented by 194-bit. Since a custom instruction can receive only 64-bit as input and returns only 32-bits as output, it is necessary to divide the cubing operation into 3 phases:

- In the first phase the instruction is called 4 times, using which provide it with 64-bit of the input polynomial at each instruction call. In the last call only 2-bit are passed. The input polynomial is stored in an intern register of the instruction.
- In the second phase the cubing computation is accomplished. For that the instruction is called again and a code which indicates such task is passed in the input *n*.
- In the third and last phase the result of the operation is returned. As we can return only 32-bit to each call, it is necessary 7 calls to return the whole result.

Therefore, 12 calls are used to compute the cubing operation.

B. Multiplication in $\mathbb{F}_3^{97}$

In this paper we used a modification of the first most significant element multiplication algorithm over $\mathbb{F}_3^{97}$ [5]. Basically, the algorithm receives as inputs two polynomials over $\mathbb{F}_3^{97}$, compute their multiplication and returns a $\mathbb{F}_3^{97}$ polynomial as result.

Similarly as in the NIOS II custom instruction call, the call of a bus module also presents limitation. A module attached to the Avalon data bus receives as input an operand of 32-bit and returns 32-bit as result. It also needs an 8 bits (*n*) selection operand that is passed as a function selector during the module call. Due to those characteristics the multiplication operation is sub-divides in three phases:

- In the first phase the module is called 14 times to pass the two input polynomials through of the input operand of 32-bit. The two input polynomials are stored into two internal registers in the module.
- In the second phase the multiplication is executed. This phase happens during the last call of the previous phase, and this no call is needed.
- In the third and last phase, 7 calls to the module are performed, so as to enable the return of the result.

Therefore, in total 21 calls to the multiplication module are done to accomplish the operation.

VI. Generic Hardware/Software Approach

Two instructions were selected to be implemented as new custom instructions into the processor. They were the multiplication and sum of two polynomials in $\mathbb{F}_3^{16}$. By using these two instructions, it is possible to compute the whole multiplication operation in $\mathbb{F}_3^{m}$.

A. Polynomial Multiplication in $\mathbb{F}_3^{16}$

This instruction receives as input two polynomials in $\mathbb{F}_3^{16}$ (represented by a word of 32-bit) and returns a polynomial in $\mathbb{F}_3^{31}$ (represented by two words of 32-bit). As we can return only 32-bit at each instruction call, it is necessary two calls to obtain the 64-bit result (the first call returns the least significant 32-bits, followed by the most significant 32-bit, after the second call).

B. Polynomial Sum in $\mathbb{F}_3^{32}$

This instruction receives as input two polynomials in $\mathbb{F}_3^{16}$ (represented by a word of 32-bit) and returns a polynomial in $\mathbb{F}_3^{16}$ (represented by a word of 32-bit). Such instruction return only 32-bit, requiring just one call to perform the sum operation.

C. Multiplication in $\mathbb{F}_3^{97}$

The multiplication using generic instructions can be divided into 3 phases. During the first phase, the partial products are computed in parallel. The second phase computes the sum of the partial products (also in parallel) and finally, the last phase performs the reduction operation by an irreducible polynomial of degree *m*.

In the first phase the two input polynomials *a(x)* and *b(x)* (both polynomials in $\mathbb{F}_3^{97}$) are subdivided into 7 polynomials in $\mathbb{F}_3^{16}$ (represented by a 32-bit word). Each element of the polynomial *b(x)* is multiplied by each element of *a(x)*, thus generating 49 partial products (each partial product is a polynomial in $\mathbb{F}_3^{31}$).

In the second phase the partial products generated in the first phase are summed. Finally, the resulting polynomial of the sum is reduced by an irreducible polynomial of degree 97.

VII. Results

Two metric were used to evaluate the developed approaches. The first metric is the number of ALUTs used for each entity implementation (Table II). Observe in that table that the implementation of the hardware/software approach needs a small amount of hardware when compared with a purely hardware approaches, as in [6] (needs of 14895 *slices*) or in [7] (needs of 55616 *slices*).

Another relevant metric was the time spent in the pairing, multiplication and cubing operations when comparing the different approaches, as these operations have been shown to represent most of the application execution time. These numbers are presented in Table III.

Observe that when the time needed to compute the multiplication operation is not considered the time of the pairing in the processor NIOS II falls ≈2%, when compared with the software approach without optimization running on a NIOS II processor. In other words, the pairing mapping done by *Oprofile* it coherent with the experimental result. Moreover notice that the time of the multiplication using the processor *Pentium 4* is ≈10 times faster than in the processor NIOS II, and that pairing also follows that same proportion. Hence, the speedups that we can achieve using hardware specialization would produce a proportionally speedup if the same technique would be used in a high-end processor.

TABLE II
Number of ALUTs for Hardware Entity

Entity	Total of ALUTs
Nios II Processor	2293
Multiplication Module	1746
Multiplication Instruction more Generic Sum Instruction	843
Cubing Instruction	261

TABLE III
EXECUTION TIMES (IN MILLISECONDS)

Approach	η_T Pairing	Multiplication	Cubing
Pentium 4 Processor at 2.4 GHz (without optimization)	7 x10^2	8,779 x10^{-1}	1,96 x10^{-3}
Nios II Processor at 50MHz (without optimization)	7,299 x10^3	8,656	4,1 x10^{-2}
Nios II Processor at 50MHz (using the multiplication module)	2,172 x10^2	5,199 x 10^{-2}	4,1 x10^{-2}
Nios II Processor at 50MHz (using the multiplication module more the cubing instruction)	2,147 x10^2	5,199 x 10^{-2}	3,60 x10^{-2}
Nios II Processor at 50MHz (using generic multiplication and sum Instructions)	3,024 x10^2	1,4899 x10^{-1}	4,1 x10^{-2}
Nios II Processor at 50MHz (not counting the time of the multiplication, it is the ideal acceleration)	1,484 x10^2	0	4,1 x10^{-2}

Table IV shows a comparison among our hardware multiplication module and several other implementations in $\mathbb{F}_{3^{97}}$. The pipelined multiplier presented in [15] is at the moment the fastest found in the literature, it requires only 11.467ns to compute a multiplication. It is important to notice that our implementation is not pipelined, and thus we expect a considerable speedup once this is done.

VIII. CONCLUSION

Our results show that the hardware/software approach leads to an improvement of about 3300% in pairing performance, thus being the best option when we needed a more performance while maintaining low cost. However, the generic hardware/software approach is little behind, getting an improvement of ≈2400% and using a circuit ≈51% smaller than the specialized hardware/software approach. Thus we can conclude that both approaches produce a considerable performance gain and that the generic hardware/software approach is the best candidate for implementation on low cost devices, given it possesses great flexibility at lower cost.

REFERENCES

[1] W. Diffie, M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, n. 6, pp. 644-654, Nov. 1976.

[2] D. Boneh, M. Franklin, "Identity-Based Encryption from the Weil pairing," *SIAM Journal on Computing*, vol. 32, n. 3, pp. 586-615, 2003.

[3] A. Shamir, "Identity-Based cryptosystems and signature schemes,", in *Proc. of the Crypto'84 on Advances in Cryptology*, 1985, p. 47-53.

[4] P. S. L. M. Barreto, S. D. Galbraith, C. Ó' Héigeartaigh, M. Scott, "Efficient pairing computation on supersingular Abelian varieties," *Designs, Codes and Cryptography*, vol. 42, n. 3, pp. 239–271, Mar. 2007.

[5] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, T. Takagi. "Algorithms and arithmetic operators for computing the η_T pairing in characteristic three," *IEEE Transactions on Computers*, vol. 57, n. 11, pp. 1454-1468, Nov. 2008.

[6] J.-L. Beuchat, M. Shirase, T. Takagi, E. Okamoto, "An algorithm for the η_T pairing calculation in characteristic three and its hardware implementation," in *Proc. of the 18th IEEE Symposium on Computer Arithmetic*, 2007, p. 97–104.

[7] T. Kerins, W. P. Marnane, E. M. Popovici, P. S. L. M. Barreto, "Efficient hardware for the Tate pairing calculation in characteristic three," in *Proc. of the Cryptographic Hardware and Embedded Systems – CHES'05*, 2005, p. 412–426.

[8] O. Ahmadi, D. Hankerson, A. Menezes "Software implementation of arithmetic in GF(3^m)," in *Proc. of the 1st international Workshop on Arithmetic of Finite Fields*, vol. 4547, pp. 85-102, Jun. 2007.

[9] J.-L. Beuchat, E. López-Trejo. L. Martínez-Ramos, S. Mitsunari, F. Rodríguez-Henríquez, "Multi-core implementation of the Tate pairing over supersingular elliptic curves," in *Cryptology ePrint Archive*, 2009, Report 2009/276.

[10] Y. Kawahara, T. Takagi, E. Okamoto, "Efficient implementation of Tate pairing on a mobile phone using Java," in *Proc. of the CIS 2006*, NAI 4456, 2007, p. 396–405.

[11] Altera Nios II processor website. Available: http://www.altera.com/products/ip/processors/nios2/ni2-index.html, 2009

[12] Altera Stratix II FPGA website. Available: http://www.altera.com/products/devices/stratix-fpgas/stratix-ii/stratix-ii/st2-index.jsp, 2009.

[13] Altera Quartus II Subscription edition software website. Available: http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html, 2009.

[14] Oprofile website. Available: http://oprofile.sourceforge.net/about/, 2009.

[15] N. Cortez-Duarte, F. Rodríiguez-Henríquez, J.-L. Beuchat, E. Okamoto, "A pipelined Karatsuba-Ofman multiplier over GF(3^{97})," in *Cryptology ePrint Archive*, 2008, Report 2008/127.

[16] R. Ronan, C. Murphy, T. Kerins, C. Ó' Héigeartaigh, P. S. L. Barreto, "A flexible processor for the characteristic 3 η_T pairing," *International Journal of High Performance Systems Architecture*, vol. 1, n. 2, pp. 79-88, 2007.

[17] P. Grabher, D. Page, "Hardware acceleration of the Tate pairing in characteristic three," In *Proc. of the Cryptographic Hardware and Embedded Systems – CHES'05,* 2005, p. 398–411.

[18] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, "Arithmetic operators for pairing-based cryptography," In *Proc. of the 9th international workshop on Cryptographic Hardware and Embedded Systems – CHES'07*, 2007, p. 239-255.

TABLE IV
HARDWARE COST OF SEVERAL $\mathbb{F}_{3^{97}}$ MULTIPLIERS SPECIFICALLY DESIGNED FOR PAIRINGS

Multipliers	Platform	Cycles	Clock Period (ns)	Latency (ns)	Frequency (MHz)	Area (Slices/Aluts)
Ronan *et al* [16]	Virtex II Pro	7	16.23	113.6	61.6	3737
This work	Stratix II	32	5.915	186.28	169	1746
Grabher *et al* [17]	Virtex II Pro	28	6.67	186.6	150	946
Beuchat *et al* [18]	Cyclone II	33	6.711	221.46	149	700
Kerings *et al* [7]	Virtex II Pro	25	34.129	853.22	29.3	1821