A Parameterizable Processor Architecture for Large Characteristic Pairing-Based Cryptography

Gary C.T. Chow
Department of Computing
Imperial College London, UK
cchow@doc.ic.ac.uk

Ken Eguro
Embedded and Reconfigurable Computing Group
Microsoft Research, Redmond, WA, USA
eguro@microsoft.com

Abstract. Cryptographic pairing (bilinear mapping) is a core algorithm for various cryptography protocols. It is computationally expensive and inefficiently computed with general purpose processors. Although there has been previous work looking into efficient hardware designs for pairing, most of these systems use small characteristic curves which are incompatible with practical software designs. In this paper, we propose a novel processor architecture for pairing-based cryptography applications using large characteristic curves. The architecture is parameterizable to fields with different bit-widths and different pairing algorithms. It takes advantage of some unique FPGA features such as huge aggregated memory bandwidth and massively parallel computation logic to achieve high performance and high energy efficiency. An example 512-bit pairing processor with this architecture can verify 9.6K pairings/second on a Xilinx Virtex-6 FPGA. It is 18.7x faster than a single threaded software version running on a 2.5 GHz Xeon E5420 CPU. The per-pairing energy consumption of the FPGA processor is estimated to be at least 6.0x better than its CPU counterpart. The proposed architecture is ideal for server-side applications requiring flexibility, performance and energy efficiency.

Keywords: Parameterizable processor, field programmable gate array (FPGA), power-aware cryptography, pairing-based cryptography, elliptic-curve cryptography (ECC), $\mathbb{F}_p$ arithmetic, large characteristic.

1 Introduction

Pairing-based cryptography protocols utilize bilinear mapping. It allows some important features such as identity-based cryptographic schemes and short signature schemes. These are not readily available using traditional cryptography techniques [1]. Furthermore, in recently years there has been a paradigm shift from traditional computing to "Cloud Computing". Instead of computing with local infrastructure, computations are provided as a service over the Internet ("The Cloud"). This generates huge demand for pairing-based cryptography computations to maintain system integrity. Despite improvements in clock frequency and the level of parallelism of CPUs, software implementations of pairing-based cryptography protocols remain insufficient, both in terms of performance and energy efficiency. This problem is exacerbated in server-side applications where pairing computation requirements scale

proportionally to the number of clients. The situation has driven research into hardware accelerators for cryptographic pairings.

Although efficient hardware pairing implementations were proposed in [2-7], they may not be suitable for use in practical systems for two reasons. First, they lack flexibility, targeting particular pairing algorithms using specific curves with pre-defined bit-widths (e.g. Modified Duursma-Lee Algorithm for Tate pairing over $\mathbb{F}_{2^{239}}$). Rigid implementations such as these make it difficult to explore the design space for use in different systems or to accommodate different modes of use. Second, they all target curves with small characteristics. Although, as will be discussed in more detail in Section 2.2, small characteristics curves are naturally very amenable for execution on dedicated hardware, they are more difficult to operate on using conventional processors compared to large characteristic curves. Furthermore, since somewhat less is known about pairing over small characteristic curves, it may be difficult to find elliptic curve / Galois field combinations for a reasonable pairing implementation.

This work differs from previous hardware designs in three major ways. First, we target large characteristic pairings that are comparatively well-understood and more compatible with client-side software. Second, instead of implementing a pairing kernel for a specific algorithm and curve, we take a higher-level approach and design a novel architecture that is parameterizable to different bit-widths and capable of implementing different pairing algorithms. Finally, the proposed architecture takes advantage of modern FPGA features to provide flexibility for system scaling.

To demonstrate the potential advantages of this design philosophy, we implemented a specialized pairing processor on a FPGA and achieved 18.7x greater throughput over a software implementation running on a conventional desktop processor. We also examine the per-pairing energy consumption of this FPGA implementation. The prototype reconfigurable processor is, evaluated very conservatively, 6.0x more energy efficient than its conventional software-based counterpart. To best of our knowledge, this is the first time power consumption has been considered in cryptographic pairing designs.

2 Background

2.1 Overview of Cryptographic Pairing

We only give a brief overview of pairing and refer readers to [8] for a more comprehensive introduction.

A cryptography pairing is a bilinear mapping from two groups to another group. The following equation gives a Tate pairing as an example. Following the notation of [8]:

$$e_r(P, Q): E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k})[r] \rightarrow \mathbb{F}^*_{q^k}$$

Although there are different pairing algorithms, such as the Tate pairing and the Eta pairing, they can all be divided into two computational stages: the Miller loop (or a

modified Miller loop) and the final exponentiation. A Tate pairing algorithm is presented in Algorithm 1 to illustrate the two stages. Step 4 and 5 represent the Miller loop and final exponentiation, respectively.

Algorithm 1: Tate pairing

Input: $P, Q \in E(F_q)[r]$

Output: $e_r(P, Q)$

1) let the binary representation of r be $r = (r_t, \ldots r_1, r_0)_2$
2) select a point $R \in E(F_q)[r] \backslash \{\infty, P, -Q, P - Q\}$
3) set $f \leftarrow 1, T \leftarrow P$
4) for *i* from *t* down to 0 do:
   a) let *l* be the tangent line through *T*, and let *v* be the vertical line through *2T*.
   b) $T \leftarrow 2T$
   c) $f \leftarrow f^2 \cdot \frac{l(Q+R)}{v(Q+R)} \cdot \frac{v(R)}{l(R)}$
   d) If $n_i = 1$ then
      i. let l be the line through *T* and *P*, let v be the vertical line through *T+P*.
      ii. $T \leftarrow T + P$
      iii. $f \leftarrow f \cdot \frac{l(Q+R)}{v(Q+R)} \cdot \frac{v(R)}{l(R)}$
5) Return($f^{(q^k-1)/r}$)

2.2 Classification of Cryptographic Pairing

We divide cryptographic pairings into two groups: "hardware-friendly" and "software-friendly". Hardware-friendly pairings have Galois fields with small characteristics (i.e. binary or ternary fields) and their security strength comes from the large power in their extension fields (i.e. small *p* and large *m* in $\mathbb{F}_{p^m}$). They are preferred in dedicated hardware designs because operations in these fields have short carry-chains and the computation of different coefficients in the polynomial can be independently evaluated (thus, it is easy to accelerate them through parallel computation). Conversely, software-friendly pairings use Galois fields with large prime numbers and small powers in their extension field (i.e. large *p* and small *m* in $\mathbb{F}_{p^m}$). Operations in these fields have long carry-chains and straightforward parallelization of the arithmetic operations is generally not possible.

Although hardware-friendly curves favor dedicated hardware computation, they may not suitable for traditional desktop processors. This is because traditional CPUs are optimized towards large native word widths (e.g. 32-bit or 64-bit). Thus, computing over hardware-friendly curves may require wasting computational power calculating small pieces of data (e.g. 1-3 bits) with wide arithmetic units. As many more coefficients are required with hardware-friendly curves to achieve the same security level as one over a software-friendly curve, it requires disproportionately more time to compute using a conventional processor. Furthermore, in the "Cloud Computing" paradigm, client-side computers are usually inexpensive, making dedicated hardware for cryptographic pairing infeasible. Employing hardware-friendly curves in such a paradigm would significantly penalize clients' performance and should be avoided.

Another reason for using software-friendly curves is the fact that cryptographic pairings are relatively restrictive algorithms and there is simply much more research on curve / field combinations for software-friendly pairings. For example, a family of combinations (Barreto-Naehrig curves) has been discovered [9]. Such a family of curves is important because it allows a potential system developer to adaptively select different curves based upon criteria such as requisite security strength or the availability of computational resources.

2.3 Introduction to FPGA

Field programmable gate arrays (FPGAs) are high performance hardware devices with the flexibility of software. They are constructed from programmable logic and memory elements embedded in a programmable interconnect network. This programmability allows them to be specialized to particular computations. This specialization leads to resource efficiency, which can be used for parallel computation to achieve high throughput.

Energy efficiency is another advantage of using FPGAs over general purpose processors. As the circuits can be specialized for particular computations, energy overhead is reduced. This is especially important for server-side applications where power consumption and cooling are critical issues.

3 Parameterizable Finite Field Arithmetic Unit

Our pairing-based processor is divided into two parts. The finite field arithmetic unit (ALU) carrying out computation in $\mathbb{F}_q$, and a control unit specialized for pairing algorithms. In this section we introduce the finite field arithmetic unit parameterizable to $\mathbb{F}_q$ arithmetic of different bit-widths.

3.1 Multiple-Precision Arithmetic

Although finite field arithmetic of both the base field ($\mathbb{F}_q$) and the extension field ($\mathbb{F}_{q^k}$) is required in pairing, the extension field arithmetic can be decomposed and only base field arithmetic has to be supported. In software-friendly pairing, the prime number q is usually large in order to provide sufficient security strength (i.e. 256-bit). Straightforward integer arithmetic circuits fail to achieve good performance with such large numbers due to the long carry chains. A common solution to the problem is multiple-precision arithmetic. Instead of computing a finite field arithmetic operation in a single clock cycle, we break it down into multiple limbs and process them sequentially. Throughout the paper, we use N to represent the total number of bits of the finite field, W to represent the width of each limb and $k = \lceil N/W \rceil$ be the number of limbs in each word. We represent each limb of a variable A with coefficients a_i using the following relationship:

$$A = a_{k-1}2^{(k-1)W} + a_{k-2}2^{(k-2)W} + \cdots + a_0$$

Fig. 1 shows the system architecture of the parameterizable finite field arithmetic unit. Three integer datapaths (a multi-purpose integer unit, a multiplication unit and a unit to assist with division) are constructed with *W*-bit integer arithmetic circuits. They are each controlled by a parameterizable controller and they share access to a segmented *N*-bit finite field register file. A master instruction unit controls sequences of the *W*-bit integer arithmetic to carry out modular arithmetic. The design of the ALU is mostly independent to the bit-width of underlying base field. Only minor changes are required in the datapath controllers to customize the ALU for different bit-widths (i.e. different number of iterations (*k*)). This approach eliminates most ALU modifications and maintains the architecture's consistency for fields with different bit-widths.

Although the performance of an individual ALU is limited because of limb serialization, the aggregated performance of all ALUs on the same FPGA is maximized by operating at high clock frequency. We achieve this by using a relatively small *W* and keeping carry chains short.

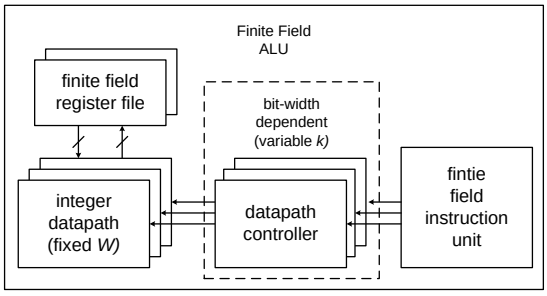

Fig. 1. Architecture of finite field arithmetic unit

3.2 Multi-Purpose Integer Datapath

Fig. 2 shows the simplified block diagram of the multi-purpose integer datapath in our ALU. The datapath can perform any computation required in modular addition, subtraction, negation, copying and inversion. It has two inputs connected to register files A and B, and an input connected to a ROM storing the modulus prime number. An inverter and a shifter are also included in the datapath to perform 2's complement subtraction and division by two. Table 1 shows all the possible operations of the multi-purpose datapath.

As an example, modular addition is computed in two passes using the multi-purpose datapath. During the first pass we test if the sum of two inputs is within the base field by configuring the datapath as (*regA* + *regB* – *prime*). Depending upon the result of the first pass, we can reconfigure the datapath as (*regA* + *regB*). Modular subtraction, negation and copying are computed with similar techniques. The algorithms used can be found in [10, ch. 14]. Modular inversion is computed using the binary extended Euclidean algorithm [10]. All computations required in the algorithm can be performed by the multi-purpose datapath with no additional circuitry.

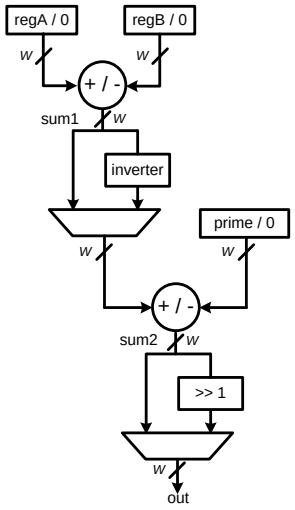

Fig. 2 Multi-purpose integer datapath

Table 1 Possible operation of multi-purpose datapath

node	Possible output
sum1	regA
	regB
	regA + regB
	regA – regB
sum2	sum1
	sum1 + prime
	sum1 – prime
	prime – sum1 (using inverter and 2's complement)
out	sum2
	sum2 / 2 (using left shifter)

3.3 Modular Multiplication

Modular multiplication is the most time consuming operation in the ALU. It requires an *N*-bit multiplication and a reduction process to reduce the 2*N*-bit output back to the prime field. We use the division by multiplication technique for the reduction [11]. Table 2 shows the five stages of our modular multiplication algorithm with the detailed computation steps and the corresponding datapath.

Due to its easy adaptation to FPGAs, the proof-of-concept system described here utilizes long multiplication. However, the same basic system architecture can be adapted if more advanced multiplication techniques are desired. The integer multiplication datapath is constructed with partial product generators (PPGs) and an adder tree. The PPGs generate partial products $P_i = b_i \times A$. The adder tree sums the partial products to the final product with the following equation:

$$R = P_0 + 2^W P_1 + \cdots + 2^{(k-1)W} P_{k-1}. \quad (1)$$

Fig. 3 shows the block diagram of the partial product generator. One of the input registers of the PPG is loaded with a particular limb of data B (b_i). Limbs of data A are fed to the other input beginning from the lower words, and the partial product is computed in (k+1) clock cycles. We use dedicated embedded multipliers within the FPGA to realize W-bit multiplication. This increases the clock frequency of the data-path and reduces the use of flexible programmable logic.

Table 2 Five stages in modular multiplication

Stage	Computation steps	Remarks	Datapath
1	$P_1 \leftarrow A \times B$	Compute product of inputs	Long multiplication
2	$T_1 \leftarrow m' \times P_1$	Multiple P_1 with the pre-computed division constant	Long multiplication
3a	$T_2 \leftarrow \left(P_1 - H(T_1)\right) \gg 1$	H(T_1) is the upper 2N-bit of T_1	Division
3b	$e \leftarrow (T_1 + T_2) \gg (N-1)$		
4	$P_2 \leftarrow e \times prime$	Compute second product	Long multiplication
5	$R \leftarrow P_1 - P_2$	Compute remainder, result of modular multiplication	Multi-purpose

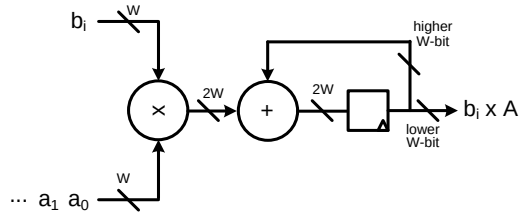

Fig. 3 Block diagram of partial product generator

We connect four partial product generators in the configuration shown in Fig. 4. The A inputs are connected through shift registers such that the partial products are aligned for summation. Fig. 5 shows the four-input adder tree connected to the PPGs' outputs. The multiplication takes k/4 passes to compute the product. The first four limb outputs in each pass are transferred back to the main register file as result. The remaining k-limb partial sum is stored in a temporary memory and added with the partial products of next pass. All the outputs are transferred to the register file in the last pass.

The addition, subtraction and shifting operations needed in the reduction stage of the multiplication are computed with a specialized datapath, shown in Fig. 6. The (N-1)-bit shift operation is reduced to an M-bit shift where $M = N$-1 (mod W). It takes a single pass to compute output e using the datapath. Although M is depending on the number of bits in the base field, a unified design for variable bit-width shifting is used. Thus, it is easy to customize the datapath to different base fields.

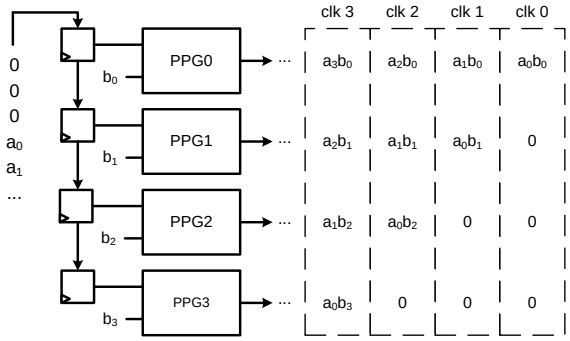

Fig. 4 Partial product generators configurations

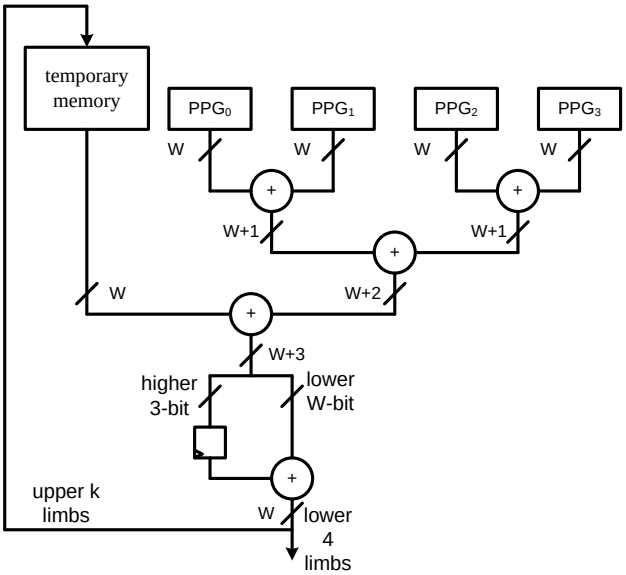

Fig. 5 Adder tree for summation

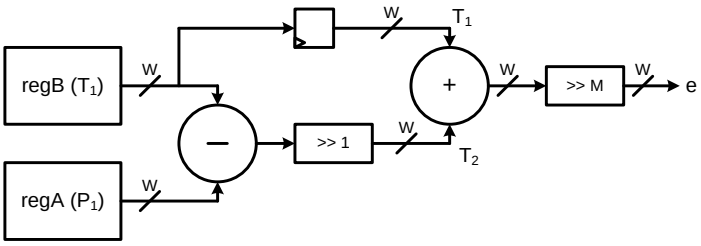

Fig. 6 Division datapath

4 Specialized Pairing Processor

While the parameterizable ALU provides flexibility for fields of different bit-widths, it needs to be incorporated into a programmable processor to accommodate different curves and pairing algorithms. In this section we illustrate how to realize such a processor efficiently by leveraging the reconfigurability of FPGAs.

There are two common approaches to constructing a pairing processor using a finite field ALU. The first approach is to use hardwired state machines. Although such an approach provides compact and efficient designs, it is difficult to customize the controller for different curves and algorithms to explore the design space. Another approach is to combine the ALU with a general purpose processor such as the Xilinx MicroBlaze [12]. This approach allows the modification of the curve and pairing algorithm through software. However, most general purpose processors offer functionality unnecessary for pairing functions. Thus, they impose unnecessary overhead and limit the performance of the whole system. We solve these problems by designing a specialized pairing processor and leveraging FPGAs' reconfigurability.

Despite variations between different curves and pairing algorithms, we note that the basic operation of all pairing is data independent. In the Miller loops of every pairing algorithm, branching only depends on a constant binary sequence related to the pairing order. Similar dependence can be observed in final exponentiation – the sequence is related to the constant exponent. We take advantage of such observations to design a processor specialized for pairing. We encode the branching sequences into a circular shift and use a special instruction (**J**ump and **S**hift **R**egister) to carry out these data independent branchings. Whenever a JSR instruction is encountered, the processor will examine the most significant bit of the shift register and decide the branching direction. The register is then shifted circularly by 1 bit. As the number of JSR instructions is fixed in every pairing algorithm, the shift register is restored to its initial value after each pairing is completed.

By using the JSR instruction, we minimize the non-finite field control instructions that would normally be necessary to implement loops. The set of control instructions in our processor is shown in Table 3. By simplifying the control aspect of the design, this system has considerably less overhead as compared with a general-purpose processor. The use of the JSR instruction also reduces the number of instructions in a pairing program. Although the binary sequence of the whole pairing could be very long, shift registers are abundant and inexpensive in FPGAs. For instance, a 1024-bit shift register can be implemented with 64 logic elements in a Xilinx FPGA (0.1% of even the smallest modern device). The size and initial value of the circular shift register is only dependent upon the curve and pairing algorithm used. It can be changed by reconfiguring the FPGA. Fig. 7 shows the block diagram of the pairing processor.

Table 3 Integer arithmetic and jump instruction set in the pairing processor

Instruction	Example	Functionality
JUMP	JUMP &addr	Unconditional jump
LOAD	LOAD cnt, val	Load specific loop counter with an immediate value
JDEC	JDEC cnt, &addr	Decrease the loop counter cnt by 1, branch if zero
JSR	JSR	Branch if the most significant bit of the circular shift register is 1, right shift circularly by 1 bit afterward

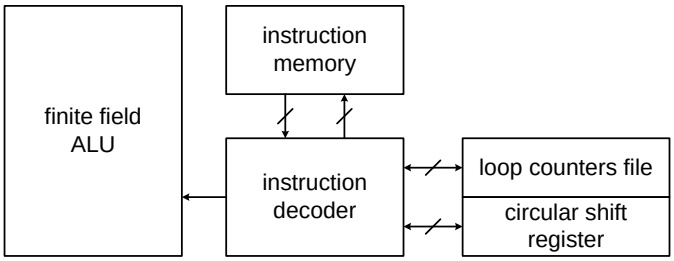

Fig. 7 Architecture of the specialized pairing processor

5 Results and Comparison

To evaluate the potential of the proposed processor architecture, we mapped the 512-bit "Type A" pairing from the well-known PBC library [13] to a Xilinx FPGA. We compared our system to the traditional software implementation running on a desktop processor. Both performance and energy efficiency were evaluated.

5.1 Software Benchmark

The "Type A" pairing is constructed on the curve $y^2 = x^3 + x$ over $\mathbb{F}_q$ for a 512-bit prime $q = 3 \bmod 4$. This pairing has an embedded degree $k = 2$, and thus the pairing result are in $\mathbb{F}_{q^2}$. PBC utilizes the GMP multiple precision library to provide large integer arithmetic [14]. The GMP library is heavily optimized to most commercial CPUs using inline assembly language. We believe this combination fairly represents one of the fastest possible software-based implementations of a given pairing algorithm. To provide a baseline for comparison, PBC was compiled with the default options on two different 64-bit Linux machines. Table 4 shows the parameters of these systems.

Table 4 Parameter of the software-based benchmarking systems

	CPU1	CPU2
Model	Intel Xeon E5420	AMD Phenom x4 9650
Max Clock Frequency	2.5 GHz	2.3 GHz
Technology Node	45 nm	65 nm
Integer Unit Bit-Width	64	64
Number of Cores	4	4
Power Consumption (Thermal Design Power)	80 W	95 W

5.2 FPGA results

We synthesized, placed, and routed this 512-bit pairing processor to two different Xilinx FPGAs. The specifications of these two devices are shown in Table 5. The Virtex-6 device was selected because it represents a state-of-the-art modern FPGA. However, for practical purposes we also needed to map the system to an older-generation Virtex-5 device. Although fully functional FPGA designs can be compiled for any device using the Xilinx development tools, actual execution requires the physical device itself – the Virtex-5 device was the platform available to us. This system contained 16 parallel pairing cores. Correctness of the system was verified with over 60 million Monte Carlo test vectors running over a period of two days.

	Virtex-5	Virtex-6
Model	xc5vlx110t	xc6vlx760
Technology Node	65 nm	40 nm
	Per core resource utilization	
Look Up Table (LUT)	1471	1850
Registers	2207	1947
DSP Slices (embedded multipliers)	4	4
Block Memory (18-Kbit)	4	4
Max Clock Frequency	208 MHz	399 MHz

Table 5 FPGA results of the benchmarking processor

5.3 Comparison and Analysis

Table 6 shows the performance and energy efficiency of the software and FPGA pairing systems. The Virtex-5 results come directly from the 16-core prototype design. The Virtex-6 resource requirements were estimated based upon the total resource count available on the device divided by the per core requirements. We estimated the Virtex-6 power consumption by using XPower, a tool provided by the vendor. We also estimate the power consumption of the CPUs by using their thermal design power (TDP) specifications. As CPU-based systems require peripherals such as the Northbridge chipset, hard drive and main memory, our comparison likely drastically underestimates their power consumption. Conversely, the FPGA designs are fully self-contained and do not require any peripherals. Thus, their estimations are more accurate.

Although we tried to make a fair comparison between the software and hardware systems using the same algorithmic optimizations, the software implementation has two advantages over the hardware processor. First, our processor uses long multiplication with $O(N^2)$ complexity. The GMP library uses multiplication algorithms with far lower complexity (e.g. Karatsuba). Second, our processor uses division by multiplication for reduction while the PBC library employs Montgomery reduction. Thus, the FPGA needs to perform more integer multiplications for the same operation. We would like to introduce Karatsuba and Montgomery multiplication in future versions to improve overall performance.

216 of the proposed processors running in the Virtex-6 FPGA are 18.7x faster than a single-threaded software implementation running on a 2.5 GHz Xeon. Although the speed up drops to 4.7x when comparing with a 4-thread version, the estimated energy efficiency (J/pairing) is 6.0x better in our FPGA design. As discussed earlier, the improvement in energy efficiency of the FPGA-based system is expected to be even higher in real-life scenarios.

Table 6 Performance and energy efficiency comparison of software and FPGA pairing system.

	Intel Xeon E5420	AMD Phenom x4 9650	Virtex-5 (xc5vlx110t)	Virtex-6 (xc6vlx760)
	Single core result			
Throughput per core (pairings / second)	511.8	463.6	23.1	44.3
Latency (*ms*)	1.954	2.157	43.3	22.6
Clock Frequency	2.5 GHz	2.3 GHz	208 MHz	399 MHz
	Multiple core aggregated result			
# of cores	**4**	4	16	**216**
System Throughput (pairings / second)	**2047.2**	1854.4	369	**9568.8**
Average Latency (*ms*)	**0.49**	0.54	2.7	**0.10**
System Power Consumption (W)	**80**	95	5.42	**62.2**
Energy / pairing (*mJ* / pairing)	**39.1**	51.1	14.6	**6.5**

6 Conclusion

In this paper we presented a novel processor architecture for large characteristic pairing-based cryptography. The processor is parameterizable to different curves/ pairings and optimized for modern FPGAs. Hence, it is suitable for cryptosystems requiring variable strength security. We demonstrated an example 512-bit processor that provides superior throughput while offering better energy efficiency compared with traditional software-based pairing systems. To best of our knowledge, this is the first time that energy efficiency has been considered in pairing-based cryptography. Our results suggest that the proposed pairing processor is suitable for server-side applications in a datacenter environment where energy efficiency and cooling are critical issues.

Reference

[1] Menezes , A.: An Introduction to Pairing-Based Cryptography. Unpublished manuscript, 2005. http://www.math.uwaterloo.ca/~ajmeneze/publications/pairings.pdf

[2] Shu, C., Kwon, S., Gaj, K.: FPGA Accelerated Tate Pairing based Cryptosystems over Binary Field. In Field Programmable Technology, pp. 173 –180 (2006).

[3] Kerins, T., Marnane, W., Popovici, E., Barreto, P.: Efficient Hardware for the Tate Pairing Calculation in Characteristic Three. In CHES, pp 412-426 (2005).

[4] Kerins, T., Popovici, E., Marnane, W.: Algorithms and Architectures for Use in FPGA Implementations of Identity Based Encryption Schemes. In FPL, pp. 74-83 (2004).

[5] Grabher, P., Page, D.: Hardware Acceleration of the Tate Pairing in Characteristic Three. In CHES, pp. 398-411 (2005).

[6] Beuchat, J., Brisebarre, N., Detrey, J., Okamoto, E.: Arithmetic Operators for Pairing-Based Cryptography. In CHES, pp. 239-255 (2007).

[7] Ronan, R., Eigeartaigh, C., Murphy, C., Terins, T., Barretto, P.: A Reconfigurable Processor for the Cryptographic ηT Pairing in Characteristic 3. In the International Conference of Information Technology, pp. 11-16 (2007).

[8] Dutta, R., Barua, R., Sarkar, P.: Pairing-Based Cryptographic Protocols: A Survey. In Cryptology ePrint Archive, Report 2004/064.

[9] Barreto, P., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In Selected Areas in Cryptography – SAC, pp. 319-331 (2005).

[10] Menezes, A.J., Oorschot P.C.V., Vanstone, S.A.: Handbook of Applied Cryptography CRC Press, 1996.

[11] Granlund,T., Montgomery, P. L.: Division by Invariant Integers using Multiplication. In ACM SIGPLAN, pp. 61-72 (1994).

[12] Xilinx inc, MicroBlaze Soft Processor Core. http://www.xilinx.com/tools/microblaze.htm.

[13] Lynn, B.: The Pairing-Based Cryptography Library. http://crypto.stanford.edu/pbc/

[14] Torbjörn Granlund et al.: GNU Multiple Precision arithmetic library 5.0.1. http://gmplib.org/ (2010).

[15] Barreto, P., Kim, H., Lynn, B., Scott, M.: Efficient Algorithms for Pairing-Based Crypto-Systems. In Crypto, LNCS, vol. 2442, pp. 354-368 (2002).

[16] Duursma, I., Lee, H.: Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$. In Asia Crypto, LNCS, vol 2894, pp. 111-123 (2003).

[17] Galbraith, S., Harrison, K., Soldera, D.: Implementing the Tate pairing. In ANTS, LNCS, vol. 2369, pp. 324-337 (2002).

[18] Granger, R., Page, D., Stam, M.: Hardware and Software Normal Basis Arithmetic for Pairing Based Cryptography in Characteristic Three. IEEE Transactions on Computers, 54, pp. 852-860 (2005).

[19] Chatterjee, S., Sarkar1, P., Barua, R.: Efficient Computation of Tate Pairing in Projective Coordinate over General Characteristic Fields. In Information Security and Cryptology – ICISC, pp. 168-181 (2004).

[20] Fan, X., Gong , G., Jao, D.: Efficient Pairing Computation on Genus 2 Curves in Projective Coordinates. In Selected Areas in Cryptography - SAC, LNCS, vol. 5381, pp. 18--34 (2009).

[21] Kammler, D., Zhang, D., Schwabe, P., Scharwaechter, H., Langenberg, M., Auras, D., Ascheid, G., Mathar, R.: Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves. In CHES, pp. 254-271 (2009).

[20] Hankerson, D., Menezes, A., Vanstone, S, "Guide to Elliptic Curve Cryptography," *Springer Professional Computing*, Springer, 2004.

[21] C. McIvor, M. McLoone, J. McCanny, "Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures," in *Signal, System and Computers*, 2003, pp. 379-384.

Appendix 1, Related work

Research looking into improving cryptographic pairing performance can be divided into two categories: algorithmic optimizations and hardware acceleration.

Algorithmic improvements include works of Barreto et al. [15], Duursma and Lee [16], Galbraith et al. [17] and Granger et al. [18]. These authors lowered the complexity of computing pairing functions by deleting unnecessary operations in the original Miller algorithm. Chatterjee et al. [19] and Fan et al. [20] use projective coordinate techniques to improve pairing performance by avoiding divisions. These techniques are general, applicable to most curves and most computing platforms. Some algorithmic optimization techniques are pairing dependent and they are only applicable to pairings on certain fields or elliptic curves. For instance, some pairing optimizations are made based on the fact that squaring and cubing is inexpensive in characteristic 2 and 3, respectively. Other optimizations include choosing field or subgroup order with low Hemming weight or other special properties.

Most hardware pairing accelerators are based on curves with binary or ternary fields. They usually target a specific pairing algorithm and a specific curve / field combination. A hardwired implementation of Tate pairing over the binary fields $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$ can be found in [2]. A hardwired pairing implementation using the Duursam-Lee algorithm on a curve defined over $\mathbb{F}_{3^{97}}$ is presented in [3], a similar design using η_T pairing can be found in [6]. In [5], a finite field arithmetic unit is combined with a general purpose soft processor for pairing using the Duursma-Lee algorithm. However, it is unclear how the architecture would be used to execute pairing on other curve / field combinations other than $\mathbb{F}_{3^{97}}$. Furthermore, the use of a general-purpose processor with an ancillary finite field unit imposes unnecessarily overhead and limits the performance of the system. This is because a general-purpose processor generally offers much more functionality than is necessary to control the finite field unit, making it slower and more complex. A configurable processor is presented in [7]. In this system, designers can vary different parameters for trading off parallelism and performance. In this way, the best implementation for a particular situation might be found. However, while the level of parallelization in this system is configurable, the processor only targets Eta pairing over $\mathbb{F}_{3^{97}}$.

An application specific instruction-set processor (ASIP) appears in [21]. It computes pairing defined over a specific 256-bit BN curve using optimal Eta pairing. It is not clear if the ASIP is parameterizable to other curves and it is not optimized for use with FPGAs, as it targeted towards embedded systems using application specific integrated circuits (ASIC).