A programmable FPGA-based cryptoprocessor for bilinear pairings over $\mathbb{F}_{2^m}$

Eduardo Cuevas-Farfán
Computer Science Coordination
Instituto Nacional de Astrofisica,
Optica y Electronica
Puebla, México, 72840
Email: cuevas.farfan@inaoep.mx

Miguel Morales-Sandoval
Information Technology Lab
Research Center for Advanced Studies
of the National Polytechnic Institute,
Cd. Victoria, Mexico
Email: mmorales@tamps.cinvestav.mx

René Cumplido
Computer Science Coordination
Instituto Nacional de Astrofisica,
Optica y Electronica
Puebla, México, 72840
Email: rcumplido@inaoep.mx

Claudia Feregrino-Uribe
Computer Science Coordination
Instituto Nacional de Astrofisica,
Optica y Electronica
Puebla, México, 72840
Email: cferegrino@inaoep.mx

Ignacio Algredo-Badillo
University of Istmo
Tehuantepec, Oax. México. 70760
Email: algredobadillo@sandunga.unistmo.edu.mx

***Abstract*—Bilinear pairings over elliptic curves are an emerging research field in cryptography. First cryptographic protocols based on bilinear pairings were proposed by the year 2000 and currently they are not standardized yet. The computation of bilinear pairings relies on arithmetic over finite fields. In the literature, several works have focused in the design of custom hardware architectures for efficient implementation of this arithmetic, but in a non-standardized environment a flexible design is prefered in order to support changes in the specifications. This paper presents the design and implementation of a novel programmable cryptoprocessor for computing bilinear pairings over binary fields in FPGA, which is able to support different algorithms and corresponding parameters as the elliptic curve, the tower field and the distortion map. The results show that high flexibility is achieved by the proposed cryptoprocessor at a competitive timing and area usage, when it is compared to custom designs for pairings defined over singular/supersingular elliptic curves at a 128-bit security level.**

I. Introduction

Bilinear pairings on elliptic curves first appeared as a cryptanalysis technique against Elliptic Curve Cryptography (ECC) [1], [2], for more details about ECC refer to [3], [4]. Since the year 2000, bilinear pairings have served as the basis for developing constructive protocols like Three-Party Key Exchange, Pairing-Based Encryption, and Short Signatures [5]–[7]. The publication of those protocols represented a milestone on cryptography, triggering the research in this topic.

Being $E(\mathbb{F}_q)$ an elliptic curve defined over the finite field $\mathbb{F}_q$, a bilinear pairing is a function $\hat{e} : E(\mathbb{F}_q) \times E(\mathbb{F}_q) \to \mathbb{F}^*_{q^k}$ that maps two points in the curve to an element in the extended finite field $\mathbb{F}^*_{q^k}$. Several parameters like the elliptic curve, the tower field and the distortion map must be defined in order to satisfy the required conditions for bilinear pairings to work [8]. The computation of $\hat{e}$ requires some arithmetic operations in $\mathbb{F}_q$ and $\mathbb{F}_{q^k}$. Efficient implementation of these operations remains as an open research topic [9]. For hardware implementation, efficiency represents a compromise among four different aspects: processing time, area or resources needed, power consumption and functional flexibility. Several works have been published in the literature reporting hardware implementations for computing bilinear pairings [10]–[16].

In [10] an implementation of the Optimal ate pairing considering prime fields is presented, the pairing is computed using a Residue Number System able to reduce the arithmetic complexity. The work reported at [11] is an architecture able to compute the Eta pairing over binary fields for a security level of 128 bits. That architecture is based on a Karatsuba-Ofman multiplier which uses a serial-parallel approach. Operations scheduling was optimized mainly during Miller's algorithm stage. The authors of [12] proposed a custom architecture for computing the eta pairing for a security level of 128 bits, this architecture is the fastest reported in the literature to date. The architecture implements the field multiplication through the Toeplitz matrix vector products. A thoroughly optimization in the final exponentiation stage was performed. In [13] authors presented two architectures for computing the eta pairing, one for binary field and another for ternary fields. The central module of both architectures is a fully-parallel Karatsuba-Ofman multiplier. Both cases use a pipelined approach for improving the processing time, where the security level reaches 109 bits. In [14] the author explores the viability of implementing bilinear pairings over composite-extended fields, analyzing the impact of using this kind of fields in the security of the system, the area resources and the hardware implementation performance. That work developed a compact hardware architecture for ternary fields, at a security level of 128 bits. The work in [15] proposes a coprocessor for computing the final exponentiation stage in ternary fields. That design consists in an operator for multiplication, addition and cubing, along with a programmable control. Finally, in [16] a programmable architecture for the Tate, ate, and R-ate pairing over prime fields is presented. The architecture centers its programmability in configurable arithmetic units; each of them is able to perform three operations in parallel and generates its own schedule depending on the desired functionality.

The works [10], [11], [13] describe very specialized architectures, where only a couple of parameters are configurable. Due to bilinear pairings are not yet standardized, a rigid design could not be suitable for changing specifications in paring-based protocols. Although [14] shows some more flexibility, it is not intended to support different parameters. Despite [15] is more flexible in its functionality, it only covers the final exponentiation step. [16] is flexible enough to support different algorithms but it only supports prime fields. Arithmetic over binary fields is carry-free, therefore, it usually reports smaller and faster implementations compared to prime fields. For the best of the authors' knowledge, a flexible implementation for pairing computation over binary fields has not been reported.

This paper presents a programmable cryptoprocessor for computing bilinear pairing on elliptic curves defined over $\mathbb{F}_{2^m}$. Different to other works, this cryptoprocessor is flexible enough to support different parameters like the elliptic curve, the tower field and the distortion map. The proposed cryptoprocessor is able to compute different versions of Miller's Algorithm as well as different versions of the final exponentiation. Indeed, the only fixed parameters are the finite field order and the irreducible polynomial, but they can be easily modified by running a re-synthesis process without architectural changes. Even when a standard is established, implementation parameters which don't affect the security level, like the elliptic curve, the tower field and the distortion map are managed easily by the proposed architecture. In this sense, the presented cryptoprocessor is able to adapt to different protocol specifications. Additionally, the proposed cryptoprocessor uses a very compact instruction format, keeping programs smaller than other control schemes. Implementation results show that area and processing time are both competitive compared with related works.

The paper is organised as follows: In section II a background on arithmetic over finite fields is presented. In section III it is explained the concept of bilinear pairings, focusing in the Tate and eta pairing. In section IV the proposed programmable cryptoprocessor architecture is presented, explaining the instruction format, internal modules and its programmability. Implementation results are presented in section V. Finally, section VI concludes this work and outlines future work.

II. Finite field arithmetic

A finite field, represented by $\mathbb{F}_q$ where $q = p^m$, is an algebraic structure defined as a finite set of elements, two basic operations for those elements, and a set of properties to be satisfied. p is called the characteristic of the finite field and m is called the field extension [17]. For cryptographic applications p is typically 2, 3 or a prime number [18]. The number of elements on the set is determined by p^m. In polynomial basis, an element in $\mathbb{F}_q$ could be represented as a polynomial of degree at most $m-1$, where each coefficient of the polynomial can take its value only from the set $\{0, ..., p-1\}$. $\mathbb{F}_q$ is defined by a m-grade irreducible polynomial $f(x)$. This irreducible polynomial is used to satisfy the closure property by an operation called *modular reduction*. Finite field arithmetic refers to the operations that can be performed with elements in $\mathbb{F}_q$. When $p = 2$, the finite field is called *binary field*. Usually a binary field element is implemented with a m-length bit vector. This paper assumes the binary field case unless it is specified.

A. Arithmetic operations

Let $\mathbb{F}_{2^m}$ be the binary field generated by the irreducible polynomial $f(x)$. Consider $A, B \in \mathbb{F}_{2^m}$ each represented by a polynomial and implemented as a bit vector of length m. An *addition* operation, $A \oplus B$, is defined as a polynomial addition simply performed by a bitwise XOR among each coefficient with no carry propagation. The result is a polynomial with degree less than m which already belongs to $\mathbb{F}_{2^m}$.

A multiplication operation, $A \otimes B$, can be seen as a two steps operation, see equation 1. First, a polynomial multiplication is performed resulting a polynomial of degree $2m-1$. Second, a modular reduction $\mod f(x)$ is performed in order to $A \otimes B$ belongs to the $\mathbb{F}_{2^m}$. Multiplication is a very expensive operation, several works have been presented aiming to reduce its computational cost. Algorithms like Karatsuba-Ofman and Montogomery are two examples [18]. For some special types of $f(x)$ like trinomials or pentanomials, modular reduction can be computed using only a couple of additions [18].

$$\begin{aligned} C &= A \otimes B \mod f(x) \\ &= \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j x^{i+j} \mod f(x) \end{aligned} \tag{1}$$

Squaring, $\bullet^2$, is a special case of the multiplication $A \otimes B$, when $A = B$ that also requires modular reduction [18].

$$\begin{aligned} C &= A^2 = A \otimes A \mod f(x) \\ &= \sum_{i=0}^{m-1} a_i x^{2i} \mod f(x) \end{aligned} \tag{2}$$

Square root, $\sqrt{\bullet}$, is the inverse operation of squaring. Given an element A, it consists in computing the unique element C, such that $A = C^2 \mod f(x)$ holds. Squaring can be seen as a matrix multiplication $A^2 = MA$, so square root is also a matrix multiplication $\sqrt{A} = M^{-1}A$. In both cases, M depends exclusively on $f(x)$. Being $f(x)$ a trinomial or a pentanomial, squaring and square root can be computed by reordering the input operands and performing a couple of additions [18].

An interesting identity for any two elements $A, B \in \mathbb{F}_{2^m}$ is depicted in equation 3, which states that squaring is distributive over addition. It can be shown that this identity also holds for squaring root.

$$A^2 + B^2 = \sum_{i=0}^{m-1} (a_i + b_i)x^{2i} \mod f(x) = (A+B)^2 \quad (3)$$

Multiplicative Inverse, $(\bullet)^{-1}$, of A is defined as the unique element C, such that $1 = A \otimes C \mod f(x)$ holds. There exist several algorithms to compute this element, some of them are based on the Euclidean algorithm for computing the GCD, others use the Fermat's Little Theorem. Multiplicative Inverse is considered the most expensive operation in $\mathbb{F}_{2^m}$ [18].

B. Extended field arithmetic

As mentioned in the introduction, the result of a bilinear pairing function is an element in an extended finite field represented by $\mathbb{F}_{q^k}$. A field K_2 containing a field K_1 is called *extension field* of K_1, for example $\mathbb{F}_{2^m}$ is an extended field of $\mathbb{F}_2$. An irreducible polynomial $g(x)$ of degree k is necessary to define $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$ [19]. A sequence of field extensions is called *tower field* [19].

With a tower field, a basis can be constructed to represent elements in $\mathbb{F}_{q^k}$ with elements of $\mathbb{F}_q$ [19]. For example, using the tower field defined in [8], the basis $\{1, u, v, uv\}$ over $\mathbb{F}_q$ is constructed for representing elements in $\mathbb{F}_{q^4}$. Using this basis, an element $G \in \mathbb{F}_{q^4}$ is defined as a polynomial $G = g_0 + g_1 u + g_2 v + g_3 uv$ where $g_n \in \mathbb{F}_q$.

The basis constructed from the tower field allows to perform the arithmetic over $\mathbb{F}_{q^k}$ as operations over $\mathbb{F}_q$. Addition is straightforward as a polynomial addition. A multiplication also follows the polynomial multiplication rules, but when some coefficients are either 0 or 1, then multiplication is simplified.

Using the tower field defined in [8] and equation 3, squaring in $\mathbb{F}_{q^4}$ require 4 additions and 4 squaring over $\mathbb{F}_q$:

$$G^2 = (g_0 + g_1 + g_3)^2 + (g_1 + g_2)^2 u + (g_2 + g_3)^2 v + g_3^2 uv \quad (4)$$

Raising an element to the q-th power is an operation easily computed using the tower field. For tower field defined in [8] this computation requires only 5 additions over $\mathbb{F}_q$:

$$G^q = (g_0 + g_1 + g_2) + (g_1 + g_2 + g_3)u + (g_2 + g_3)v + g_3 uv \quad (5)$$

III. Bilinear pairings over elliptic curves

A. Elliptic curves

An elliptic curve is defined as a set of points (x, y) that satisfies the Weierstrass equation over $\mathbb{F}_q$:

$$E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \quad (6)$$

The number of points in the curve is given by $\#E(\mathbb{F}_q) = q + 1 - t$ where $|t| \leq 2\sqrt{q}$. When the field characteristic p divides t, denoted by $p|t$, the curve is called *supersingular*, other way it is called *ordinary*. $E(\mathbb{F}_q)$ and a rule for adding two elements in $E(\mathbb{F}_q)$ construct an algebraic structure called *cyclic additive group*. The identity element in this additive group is named the point at infinity denoted by ∞ [20].

The *scalar multiplication* is the operation denoted as rP, where $r \in \mathbb{N}$, and $P \in E(\mathbb{F}_q)$, and it is defined as:

$$rP = \underbrace{P + P + P + ... + P}_{r} \quad (7)$$

B. Bilinear pairings over elliptic curves

A bilinear pairing over elliptic curves is a function $\hat{e}: E(\mathbb{F}_q) \times E(\mathbb{F}_q) \rightarrow \mathbb{F}_{q^k}^*$ that takes two elements from $E(\mathbb{F}_q)$, and maps them to an element in a subset of the extended finite field $\mathbb{F}_{q^k}^* = \mathbb{F}_{q^k} - \{0\}$. Bilinear pairings must satisfy the following conditions $\forall\, P, Q, R \in E(\mathbb{F}_q)$ [20]:

$$\begin{aligned} \text{Bilinearity: } & \hat{e}(P+R, Q) = \hat{e}(P,Q)\hat{e}(R,Q) \text{ and} \\ & \hat{e}(P, Q+R) = \hat{e}(P,Q)\hat{e}(P,R) \\ \text{Non-degeneracy: } & \hat{e}(P,P) \neq 1 \\ \text{Computability: } & \hat{e} \text{ is efficiently computed} \end{aligned}$$

Under certain circumstances, bilinear pairings are defined over two different curves $\hat{e}: E(\mathbb{F}_q) \times E(\mathbb{F}_{q^k}) \rightarrow \mathbb{F}_{q^k}^*$. This kind of pairing is called *asymmetric* pairing, while the former is called *symmetric* [20].

C. The Tate and eta pairing

The smallest possible value of r that makes $rP = \infty$ is called the *order of P*. The subset of points in $E(\mathbb{F}_q)$ of order r is named the *r-Torsion* subgroup, denoted by $E[r]$. Given an elliptic curve $E(\mathbb{F}_q)$ and a point $P \in E(\mathbb{F}_q)$ of order r such that $GDC(r, q) = 1$, the *embedding degree* of the curve is the smallest integer k that satisfies $r | q^k - 1$. For binary fields and supersingular curves the maximum embedding degree achievable is $k = 4$ [21].

The Tate pairing is an asymmetric bilinear pairing over elliptic curves defined in equation 8, where: D_Q is a divisor of point Q, and f_P is a function over the elliptic curve that returns a finite field element. Readers are referred to [22] and [20] for more detailed definitions. The computation of equation 8 is divided in two stages: first $f_P(D_Q)$ is computed by the Miller's Algorithm [22], second an exponentiation to the $(q^k - 1)/r$-th power, called *final exponentiation*, is required [21]. The Miller's Algorithm is a numeric method to construct the function f_P satisfying the necessary conditions of a bilinear pairing [22].

$$\begin{aligned} \tau :\; & E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k})[r] \rightarrow \mathbb{F}_{q^k}^* \quad (8) \\ \tau(P, Q) = \; & f_P(D_Q)^{(q^k - 1)/r} \end{aligned}$$

Several works have been done to optimize the Tate pairing at an algorithmic level [8], [23], [24]. An especial case of the original Tate pairing for supersingular curves is the eta pairing (η_T) presented in [8]. η_T reduces the Miller's Algorithm to the half becoming the most popular algorithm for bilinear pairings over binary fields. η_T requires a *distortion map* $\psi : E(\mathbb{F}_q) \rightarrow E(\mathbb{F}_q)$ for the point Q in order to satisfy $E(\mathbb{F}_q)$ being a cyclic group. Additionally, it can be shown that the groups $E(\mathbb{F}_q)$ and $E(\mathbb{F}_{q^k})$ are isomorphic, becoming η_T a symmetric pairing.

In Figure 1 is depicted the algorithm for computing the η_T over $\mathbb{F}_{2^m}$. Several parameters depend on the elliptic curve and finite field [8]. Consider the supersingular curve $E : y^2 + y = x^3 + x + b$ over $\mathbb{F}_{2^m}$, where $b = \{1, 0\}$ and m is odd, embedded degree $k = 4$, tower field defined as [8],

```
Require: $P = (x_1, y_1), Q = (x_2, y_2) \in E(\mathbb{F}_{2^m})$.
Ensure: $\eta_T(P, Q) \in \mathbb{F}_{2^{km}}$.
1: $s \leftarrow x_1 + \alpha$
2: $F \leftarrow s \cdot (x_1 + x_2 + 1) + y_1 + y_2 + \frac{1-\beta}{2} + (y_2 + s) \cdot u + v$
3: for $i = 1$ to $(m+1)/2$ do
4:     $s \leftarrow x_1 + \gamma$, $x_1 \leftarrow \sqrt{x_1}$, $y_1 \leftarrow \sqrt{y_1}$
5:     $G \leftarrow s \cdot (x_1 + x_2 + \gamma) + y_1 + y_2 + (1 + \gamma) \cdot x_1 + \delta + (s + x_2) \cdot u + v$
6:     $x_2 \leftarrow x_2^2$, $y_2 \leftarrow y_2^2$
7:     $F \leftarrow F \cdot G$
8: end for
9: return $F^{(2^{2m}-1)\cdot(2^m+1-\epsilon 2^{(m+1)/2})}$
```

Fig. 1. Computation of $\eta_T(P, Q)$ over $\mathbb{F}_{2^m}$.

and the distortion map $\psi(x, y) = (x + u + 1, y + xu + v)$. Lets define $\beta = -1$ when $m \equiv 1, 7 \mod 8$ and $b = 1$ or $m \equiv 3, 5 \mod 8$ and $b = 0$, or $\beta = 1$ in all other cases. $\alpha = 0, \gamma = 1$ when $m \equiv 1, 5 \mod 8$ otherwise $\alpha = 1, \gamma = 0$. $\delta = 1$ if $m \equiv 5, 7 \mod 8$, otherwise $\delta = 0$. And finally, $\epsilon = (-1)^b$ if $m \equiv 1, 7 \mod 8$ or $\epsilon = (-1)^{(1-b)}$ if $m \equiv 3, 5 \mod 8$.

In Figure 1, lines 1 throw 8 are the Miller's Algorithm stage. Lines 2 and 5 set $F, G \in \mathbb{F}_{2^{4m}}$. Line 7 is a multiplication over the extended field, thanks to the structure of G, this multiplication can be simplified. Finally, line 9 is the final exponentiation that can be computed using several techniques [21], [25], [26].

Nevertheless, an alternative algorithm for computing the η_T using different parameters for basis is presented in [27]. Both works [8] and [27] are able to compute bilinear pairings in a very different way. The necessity of a flexible solution able to manage this variety of parameters and algorithms emerges because development of algorithms and improvements are still in process and further, the lack of an standard for computing bilinear pairings in cryptographic applications.

IV. Cryptoprocessor architecture

The aim of the proposed pairing cryptoprocessor is to bring flexibility to the computation of bilinear pairings over binary fields. Several algorithms have been proposed for computing pairings, and for the best of the authors knowledge there is no protocol or standard that defines an specific algorithm or parameters to be considered. So a flexible cryptoprocessor for pairings that allows to manage several parameters such as the elliptic curve, the tower field and the distortion map, or even different algorithms is desired. The proposed solution consists in a programmable cryptoprocessor composed by a micro-architectural datapath with its corresponding instruction set.

All operations required by the pairing algorithm can be translated into arithmetic in $\mathbb{F}_{2^m}$ no matter the curve selected.

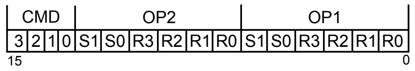

Fig. 2. Proposed Instruction Format.

TABLE I
INSTRUCTION SET FOR THE PROPOSED ARCHITECTURE.

Name	Description
StoreMult$(D[])$	Store a multiplication result.
Addition$(D[], S[])$	Addition of registers of a Bank.
Squaring$(D[], S[])$	Squaring an addition of registers of a Bank.
Square Root$(D[], S[])$	Square root an addition of registers of a Bank.
LoadMult$(S2[], S1[])$	Load and start a new multiplication.
IncG0$()$	Increment $B0$ in 1.
MoveBank$(D[], S[])$	Copy values from source to the dest. Bank.
Wait(n)	Freeze the IP register for n clock cycles.
Jmp(n)	Unconditional Jump to instruction n.
For(n)	Hardware support to FOR-Loop n.
Jz$()$	Conditional Jump

Even the arithmetic for extended fields can also be translated to simpler arithmetic operations in $\mathbb{F}_{2^m}$ no matter the tower field used. So the schedule of the instructions in the program is enough to implement some pairing algorithm with some parameters. In this sense, the first restriction is that the micro-architecture should only support arithmetic in $\mathbb{F}_{2^m}$.

In order to satisfy the first design restriction, the instructions set has to cover all the operations in $\mathbb{F}_{2^m}$ used in the pairing algorithms for binary fields reviewed in the literature. These operations are the addition, multiplication, squaring and square root discussed in previous sections. Additionally, some instructions are required for program control in order to support loops, as well as conditional and unconditional jumps. The complete instruction set is shown in table I where D means the destination bank, S means the source bank, and between brackets the registers accessed within the bank are indicated: $F[0, 1]$ means to access registers $F0$ and $F1$.

In extended field arithmetic, input operands are usually additions over the coefficients of an extended field element. In Miller's Algorithm, multiplication inputs are usually additions among the coordinates of points P and Q. A second restriction is that multiplication, squaring and square root are always preceded by an addition of at most four elements.

Pairing algorithms do not require operations with constant values rather than 0 or 1. A third requirement is that only operation among registers is supported. Constant values 0 and 1 are easily computed, for any element $A \in \mathbb{F}_{2^m}$, $A \oplus A$=0. Using the bit vector representation, the operation $A \oplus 1$ is performed by the negation of the LSB of A, then $A \oplus (A \oplus 1)$=1.

The proposed architecture uses a 16-bit instruction. Figure 2 illustrates the instruction format. For all instructions, CMD is a 4-bit field indicating the functionality, $OP2$ is a 6-bit field that indicates the destination register, $OP1$ is a 6-bit field that indicates the source register. The subfields $S1$ and $S0$ are used to select the bank register. Subfields $R3$ to $R0$ are used to select the specific registers within the bank. Note that more than one register within the bank can be read at the same time. Each bank has 4 registers in order to store one extended field

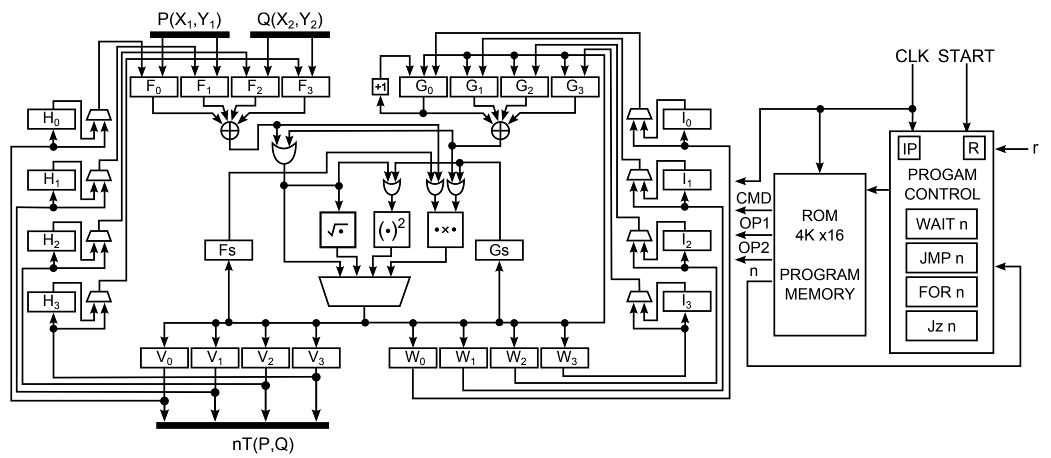

Fig. 3. Proposed architecture for computing bilinear pairings over binary fields.

element. Architecture supports four control instructions: *Jmp*, *For*, *Wait* and *Jz*. For these instructions, $OP2$ and $OP1$ act like a 12-bit constant. Instruction format only allows one bank access at a time except for multiplication.

Figure 3 shows the proposed datapath. It contains six bank registers F, G, H, I, V, W. Only banks F and G can be used as source banks for arithmetic operations. For the multiplication, one operand comes from bank F and the other comes from bank G. Only banks V, W and G can be used as destination banks for arithmetic operations. Banks H and I are used as temporal storage of banks V and W respectively. The *MoveBank* instruction is used to copy the values from one bank to another but just certain movements are supported: from V to F or H, from H to F, from W to G or I, and from I to G. There are two extra registers, Fs and Gs, used as alternative inputs for multiplication. The register $G0$ is equipped with extra hardware for computing $G0 = G0 \oplus 1$.

A. Architectural Modules

Addition is computed using a single bitwise XOR gate. An addition is directly performed at the output of banks F and G using a 4-input XOR each. In order to indicate which registers are being added and which not, each register within bank F and G has a read enable signal.

Multiplication was implemented using a serial-parallel approach of the Karatsuba-Ofman Algorithm (KOA) [11]. Inputs of size m are split twice using the KOA resulting in 9 partial operands of size $m/4$. These 9 partial multiplications are computed serially by a fully-parallel KOA of $m/4$ bits. Finally, the 9 partial results are merged according to KOA to complete the multiplication. State-of-art improvements are implemented for the $m/4$-bits multiplier [28]. Figure 4 shows the architecture for the $\mathbb{F}_{2^m}$ multiplication used in this work. This multiplier requires 9 clock cycles for computing a field multiplication.

Modular reduction is required within both multiplication and squaring modules. Representing $f(x)$ as a bit-vector notice that $f_m = f_0 = 1$. Thus, $xA(x)$ becomes a shift to the left operation on $A(x)$ leading to a $(m+1)$-bit vector, $xA(x) = (a_{m-1}, a_{m-2}, \cdots, a_1, a_0, 0)$. The resulting bit vector is the same with an extra 0 at the least significant position. If $a_{m-1} = 0$, a reduction is not necessary. However, if $a_{m-1} = 1$, the resulting polynomial is reduced $\mod f(x)$, following equation 9, which defines $xA(x) \mod f(x)$ considering $f_m = f_0 = 1$, where $\oplus$ represents a bitwise XOR operation and $\odot$ represents a bitwise AND operation. This expression is well modeled by the Linear Feedback Shift Register (LFSR) shown in figure 5. The combinatorial logic, CL-LFSR performs the required arithmetic to compute $xA(x)$ $\mod f(x)$. Therefore, d CL-LFSR blocks could be connected in a cascade fashion to implement a parallel LFSR (PLFSR) and to obtain $x^d A(x) \mod f(x)$ in just one iteration. More details on the LFSR and the PLFSR are described in [29].

$$\begin{aligned} & xA(x) \mod f(x) = \qquad (9) \\ & (a_{m-2} \oplus [f_{m-1} \odot a_{m-1}], a_{m-3} \oplus [f_{m-2} \odot a_{m-1}], ..., \\ & a_0 \oplus [f_1 \odot a_{m-1}], a_{m-1}) \end{aligned}$$

As it is shown in equation 2, *squaring* consists in an expansion of the input vector interleaving a ‘0’ between each bit, followed by a modular reduction. PLFSR are used for this purpose. *Square Root* is also a cheap operation when using trinomials as irreducible polynomial. Following the algorithm described in section II-A, matrix M^{-1} can be computed off-line because the irreducible polynomial is the same when computing the pairing. The matrix multiplication $M^{-1}A$ is very sparse, so just a couple of additions are only needed.

Program Control module is used to implement the instructions *Jmp*, *For*, *Wait* and *Jz*. These instructions make use of a 12-bit constant contained in the instruction itself. Inside this module there is the 12-bit Instruction Pointer register (IP) used to indicate next instruction to be executed. A total of 4K instructions can be addressed. “START” signal resets the IP register to ‘0’. Normally the IP register increments its value every clock cycle. When a control instructions is loaded, the value of the IP register depends on the instruction.

A generic implementation of the Miller’s Algorithm requires a test over r, the binary representation of the order

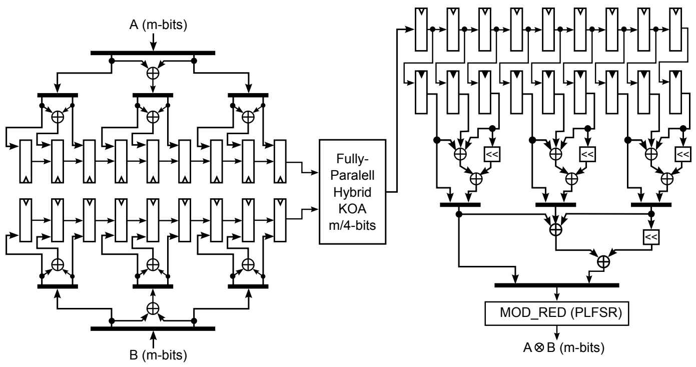

Fig. 4. Serial-parallel multiplier based on Karatsuba-Ofman algorithm with modular reduction by Parallel Linear Feedback Registers.

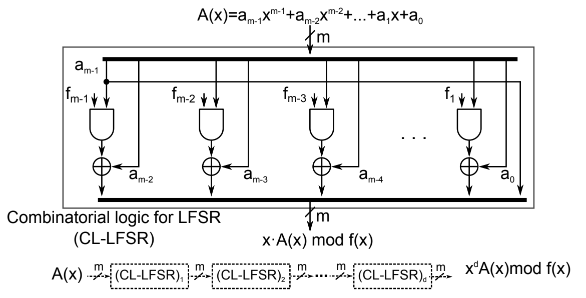

Fig. 5. Modular Reduction using Linear Feedback Shifts Registers.

of the points P and Q, see algorithm 1 of [8]. But so far the proposed algorithms for binary fields do not require it, the instruction *Jz*() is intended to cover that requirement if needed by a pairing algorithm for binary fields. A register named R inside the program control module is used for loading the input r with the "START" signal.

B. Programmability

The instructions set along with the datapath allow a lot of flexibility for pairing computing because of its programmability. Consider the algorithm depicted in figure 1 and assume that registers $F0$ to $F3$ contain the values x_1, y_1, x_2, y_2 as shown in figure 3. The addition $G0 = F0 + F2$ is computed by the instruction *Addition*$(G[0], F[0,2])$, while $W2 = G0 + G_1 + G_2 + G_3$ is computed by the instruction *Addition*$(W[0], F[0,2,3,4])$. Notice that when only one register is accessed at the source bank and destination bank, the instruction *Addition*$(D[], S[])$ is equivalent to just move one register to other.

Consider $y_1 + y_2 + \frac{1-\beta}{2}$ of the line 2 in algorithm 1 where the result depends on the value of β. When $\beta = 1$ the instruction *Addition*$(G[0], F[1,3])$ is enough for computing $G0 = y_1 + y_2$. When $\beta = -1$ the instruction *Addition*$(G[0], F[1,3])$ followed by the instruction *IncG0*() are required for computing $G0 = y_1 + y_2 + 1$.

Now consider the multiplication $s \cdot (x_1 + x_2 + \gamma)$ in line 5 of algorithm 1. Both operands depend on γ and previous multiplication is required to compute $x_1 = \sqrt{x_1}$.

1) *Addition* $(G[0], F[0])$:	move x_1 to $G0$
2) *IncG0*():	compute $s = x_1 + 1$
3) *Addition*$(Fs, G[0])$:	move s to Fs
4) *SquareRoot*$(G[0], F[0])$:	compute $G0 = \sqrt{x_1}$
5) *IncG0*():	compute $G0 = \sqrt{x_1} + 1$
6) *Addition*$(G[1], F[2])$:	move x_2 to $G1$
7) *LoadMult*$(Fs, G[0,1])$:	begin $s \cdot (\sqrt{x_1} + x_2 + 1)$

when $\gamma = 0$, consider the next sequence instead:

1) *SquareRoot*$(G[0], F[0])$:	compute $G0 = \sqrt{x_1}$
2) *Addition*$(G[1], F[2])$:	move x_2 to $G1$
3) *LoadMult*$(F[0], G[0,1])$:	begin $s \cdot (\sqrt{x_1} + x_2)$

Notice here that the program complexity is closely related with the amount of operations and the data dependency. Also notice that other operations can be computed while the multiplication is being executed. For this example in line 5 of algorithm 1, $y_1 + y_2 + (1+\gamma) \cdot x_1 + \delta$ can be computed in parallel with the multiplication $s \cdot (\sqrt{x_1} + x_2 + \gamma)$.

The programmability of the proposed architecture also brings support to compute the multiplicative inverse operation. This operation is very expensive for hardware implementation because it requires several operations in an iterative loop. Algorithms like the Binary Euclidean Algorithm require comparators and shifters. However, the Itoh-Tsujii Algorithm computes a multiplicative inverse operation using squarings and multiplications [18], so no extra hardware is required for computing this algorithm in the proposed architecture.

Arithmetic in the extended field is easily supported by the proposed cryptoprocessor independently of the tower field. Consider raising an element to the q-th power. Equation 5 computes this operation for the tower field presented in [8]. Nevertheless, in [27] a different tower field is used, so raising an element to the q-th power is computed by equation 10. Both equations, 5 and 10 consist in computing four coefficients over $\mathbb{F}_{2^m}$. Implementing this operation with the proposed architecture consists in four instructions *Additions*$(D[], S[])$, the only consideration is that all coefficients of the element G be in the same bank.

$$G^q = (g_0 + g_2) + (g_2)x + (g_1 + g_3)x^2 + g_3x^4 \qquad (10)$$

Finally, let discuss the program control instructions: *Jmp*, *For*, *Wait* and *Jz*. Instruction *Jmp*(n) is used to perform an unconditional jump to the address specified in the instruction. Instruction *Wait*(n) is used to freeze the IP register for n clock cycles. Instruction *For*(n) is used to support a For-Loop in hardware by setting an internal counter to value n and testing: if $n = 0$, IP register increments by 1, if not, IP register increments by 2. Instruction *Jz*() performs a test in the LSB of register R, if $R_0 = 0$, IP register increments by 1, if not, IP register increments by 2.

V. Experimental results.

The proposed architecture was implemented using VHDL as a description language. Program memory was implemented with Xilinx's Block Memory Generator LogiCORE. For design validation, C/C++ routines based on the library Miracl[1]

[1]Copyright 2012 CertiVox IOM Ltd. Online available: https://certivox.com/solutions/miracl-crypto-sdk/

TABLE II
IMPLEMENTATION RESULTS FOR η_T PAIRING WITH 128 BITS OF SECURITY LEVEL.

Reference	Is Flexible?	Device	Finite field	Area (slices)	Memory (kbits)	Maximum Frequency (MHz)	Clock Cycles ($\times 10^3$)	Latency (us)	$A \cdot T$ Slices× Seg.
Exp. 1	YES	Virtex 6	Binary	16,402	5.3	180[1]	51.5	286	4.69
	YES	Virtex 4	Binary	50,968	5.3	73	51.5	703	36.1
Exp. 2	YES	Virtex 6	Binary	16,402	10.3	180[1]	57.6	320	5.24
	YES	Virtex 4	Binary	50,968	10.3	73	57.6	787	40.1
[10][2]	NO	Virtex 6	Prime	7,032+32DSP	810	250	143.1	573	4.03
[11]	NO	Virtex 6	Binary	15,167	N/R	250	47.6	190	2.88
[12]	NO	Virtex 6	Binary	16,403	N/R	267	27.3	102	1.7
[13][3]	NO	Virtex 4	Binary	78,874	N/R	130	2.4	18.8	1.41
[14]	Some	Virtex 4	Ternary	4,755	24	192	260	2,227	10.59
[16]	YES	Virtex 4	Prime	52,000	N/R	50	1,729	34,600	1,799
[30]	Software	Intel Core i7	Binary	8 Threads	N/R	2,000	1,034	517	N/A
[30]	Software	Intel Core i7	Binary	1 Thread	N/R	2,000	6,455	3,228	N/A

[1] Synthesis using flag *-register_balancing*. [2] This works implements Optimal Ate pairing. [3] This works reach a security level of 105 bits.

were used to generate test data vectors. Xilinx ISim 13.2 was used as simulation environment. For synthesis, Xilinx ISE 13.2 was used targeting both Xilinx Virtex-6 (xc6vlx130t) and Xilinx Virtex-4 (xc4vlx200) devices using flags by default.

Two experiments were carried out in order to validate the correct functioning of the proposed cryptoprocessor. Two different versions of the η_T were used in each experiment. Experiment 1 uses Miller's algorithm and extended field basis from [8] and the final exponentiation from [25]. Experiment 2 implements the algorithm and all the parameters presented in [27]. For both experiments, the underlying finite field was $\mathbb{F}_{2^{1223}}$ defined by the trinomial $f(x) = x^{1223} + x^{255} + 1$, for a security level of 128-bits [26].

Table II shows the implementation results of the synthesis process. The required area is 16,402 slices for a Virtex 6 device. From this area, about 42% is used by the field multiplier; being this, as expected, the biggest individual module from all arithmetic modules. Nevertheless, the multiplexers inside each bank register consume a great amount of resources, a total of 44%. The remaining 14% of the area is used for the rest of the arithmetic modules and control signals. Notice how the area is closely related with the technology, for a Virtex 4 a total of 50,968 were used. This is due to one Virtex 6 slice contains 4 Look-Up Table (LUT) of 6 bits input, in contrast one Virtex 4 slice contains only 2 LUTs of 4 bits inputs.

The maximum clock frequency depends on the longest path delay among two registers, for the proposed architecture this path is inside the multiplier with 7 levels of logic. The maximum frequency that the architecture can operate with is 132 MHz for a Virtex 6. Nevertheless this frequency can be improved during the synthesis process using the flag *register_balancing*. This flag moves registers through combinatorial logic to evenly distribute the paths delay between registers, increasing the maximum clock frequency. With this synthesis flag, a maximum frequency of 180 MHz is reached. In the same way, time is closely related with the technology, Virtex 6 is a 40 nm device able to work with a clock frequency up to 1,600 MHz, while Virtex 4 is 90nm technology able to work with a clock frequency up to 500 MHz.

The amount of memory required by experiment 2 is almost the double than the required by experiment 1, this is because the operations in experiment 2 in general are more dependents so more instructions $\textit{MoveBank}(D[], S[])$ were required. Additionally, the final exponentiation in experiment 2 requires a total of five multiplications over $\mathbb{F}_{q^k}$, while final exponentiation in experiment 1 only performs one multiplication over $\mathbb{F}_{q^k}$, so less code was needed. Notice that the proposed design compared with related works requires less memory.

Table II also compares the implementation results with related works. When comparing area for a Virtex 6, it can be seen that this work consumes a similar amount of slices than [11] and [12]. Even [10] reports less area, it requires 32 DSP blocks and more than 100 KB of memory. Neither [11], [10] nor [12] are flexible architectures, they are optimized to support specific tower field and distortion map. For a Virtex 4, only [14] reports less area than the proposed work but the architecture proposed in this work is almost 3x faster. Comparing processing time is hard because the proposed architecture depends on the version of the chosen algorithm and programming skills. The results in table II show that the proposed cryptoprocessor is faster compared with other flexible implementations including the state-of-art in software [30]. On the other hand, the proposed architecture is very competitive compared with custom architectures. The $A \cdot T$ product reached in this work is 5.24, which is just 1.3x bigger than [10], 1.82x bigger than [11] and 3.08x bigger than [12]. Compared with [13] the $A \cdot T$ product is 3.72x bigger, but notice that [13] only reaches a security level of 105 bits. These results show that optimized architectures are slightly faster/smaller than the proposed design, but with all the flexibility achieved, it is a fair cost.

From the results obtained, the proposed cryptoprocessor is a very feasible solution for bilinear pairing computation over binary fields. The programmability reached by this architecture allows to compute bilinear pairings independently of the elliptic curve, tower field, distortion map and the version of the pairing algorithm required by the application.

VI. Conclusion and work in progress.

Algorithms and parameters for computing bilinear pairings are still under development. Improvements are constantly reported, sometimes based on different parameters. Custom architectures are not able to support such changes, so a flexible solution able to manage several parameters like the elliptic curve, the tower field, the distortion map, or the version of the pairing algorithm is better preferred, being flexible architectures more suitable for different applications.

This paper has introduced a programmable architecture for computing bilinear pairings in binary fields. Different to other architectures this one is able to compute different versions of the pairing algorithm considering different elliptic curves, tower fields, and distortion maps, all these with the same hardware. Implementation results show that the proposed design requires a competitive amount of resources compared with related work. In addition, the processing time is shorter than the one achieved by other flexible architectures and almost as good as the custom architectures of the state-of-art. The compact instruction format allows smaller programs than related works, therefore it consumes less memory.

Several works are being performed to achieve better results. A thorough optimization process is being performed in order to improve the maximum clock frequency, being pipelining technique used as a first approach. Resource consumption is being analyzed in order to reach a smaller design, especially in the areas where more consumption is located. As this pairing cryptoprocessor was originally conceived within a whole system, a communication interface to send/receive the operands will be implemented, for this last purpose an initial idea is to use a shared memory approach so that a master processor uses this memory to transmit data and also to load the desired program.

References

[1] A. Menezes, T. Okamoto, and S. A. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field," *IEEE Transactions on Information Theory*, vol. 39, no. 5, pp. 1639–1646, 1993.

[2] G. Frey, M. Muller, and H.-G. Ruck, "The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems," *IEEE Transactions on Information Theory*, vol. 45, no. 5, pp. 1717–1719, Jul. 1999.

[3] V. S. Miller, "Use of Elliptic Curves in Cryptography," *Advances in Cryptology - CRYPTO 1985*, pp. 417–426, 1986.

[4] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[5] A. Joux, "A One Round Protocol for Tripartite Diffie Hellman," *Algorithmic Number Theory*, vol. 1838, pp. 385–393, 2000.

[6] D. Boneh and M. Franklin, "Identity-Based Encryption from the Weil Pairing," in *Advances in Cryptology - CRYPTO 2001*, pp. 213–229.

[7] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," *Journal of Cryptology*, vol. 17, no. 4, Jul. 2004.

[8] P. S. L. M. Barreto, S. D. Galbraith, C. O. Héigeartaigh, and M. Scott, "Efficient pairing computation on supersingular Abelian varieties," *Designs, Codes and Cryptography*, vol. 42, no. 3, pp. 239–271, Feb. 2007.

[9] A. Menezes, "An Introduction to Pairing-Based Cryptography," *Recent trends in cryptography*, vol. 447, pp. 47–65, 2009.

[10] 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 *Cryptographic Hardware and Embedded Systems - CHES 2011*, no. 07, 2011, pp. 421–441.

[11] S. Ghosh, D. Roychowdhury, and A. Das, "High Speed Cryptoprocessor for ηT Pairing on 128-bit Secure Supersingular Elliptic Curves over Characteristic Two Fields," in *Cryptographic Hardware and Embedded Systems - CHES 2011*, 2011, pp. 442–458.

[12] J. Adikari, M. A. Hasan, and C. Negre, "Towards Faster and Greener Cryptoprocessor for Eta Pairing on Supersingular Elliptic Curve over $F_{2^{1223}}$," in *19th International Conference, Selected Areas in Cryptography 2012*, 2012, pp. 166–183.

[13] J.-L. Beuchat, J. Detrey, N. Estibals, E. Okamoto, and F. Rodríguez-Henríquez, "Fast Architectures for the η_T Pairing over Small-Characteristic Supersingular Elliptic Curves," *IEEE Transactions on Computers*, vol. 60, no. 2, pp. 266–281, Feb. 2011.

[14] N. Estibals, "Compact Hardware for Computing the Tate Pairing over 128-Bit-Security Supersingular Curves," in *Pairing-Based Cryptography - Pairing 2010*, vol. 6487, 2010, pp. 397–416.

[15] J.-L. Beuchat, N. Brisebarre, M. Shirase, T. Takagi, and E. Okamoto, "A Coprocessor for the Final Exponentiation of the η T Pairing in Characteristic Three," in *International Workshop on the Arithmetic of Finite Fields (WAIFI 2007)*, 2007, pp. 25–39.

[16] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "High Speed Flexible Pairing Cryptoprocessor on FPGA Platform," in *Pairing-Based Cryptography - Pairing 2010*, vol. 6487, 2010, pp. 450–466.

[17] I. N. Herstain, *Abstract Algebra*, 3rd ed. Wiley, 1996.

[18] F. Rodríguez-Henríquez, A. Díaz-Pérez, N. A. Saqib, and C. K. Koc, *Cryptographic Algorithms on Reconfigurable Hardware*, ser. Signals and Communication Technology. Boston, MA: Springer US, 2006.

[19] J.-P. Escofier, *Galois Theory*, ser. Graduate Texts in Mathematics. New York, NY: Springer New York, 2001, vol. 204.

[20] J. H. Silverman, *The Arithmetics of Elliptic Curves*, 2nd ed. Springer, 2009.

[21] P. S. L. M. Barreto, H. Kim, B. Lynn, and M. Scott, "Efficient Algorithms for Pairing-Based Cryptosystems," in *Advances in Cryptology - CRYPTO 2002*, vol. 2442, 2002, pp. 354–369.

[22] V. S. Miller, "The Weil Pairing, and Its Efficient Calculation," *Journal of Cryptology*, vol. 17, no. 4, pp. 235–261, Aug. 2004.

[23] S. D. Galbraith and J. F. Mckee, "Pairings on Elliptic Curves over Finite Commutative Rings," *Cryptography and Coding*, vol. 3796, pp. 392–409, 2005.

[24] F. Hess, N. Smart, and F. Vercauteren, "The Eta Pairing Revisited," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4595–4602, Oct. 2006.

[25] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodríguez-Henríquez, "A Comparison Between Hardware Accelerators for the Modified Tate Pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$," in *Pairing-Based Cryptography - Pairing 2008*, 2008, pp. 297–315.

[26] D. Hankerson, A. Menezes, and M. Scott, "Software Implementation of Pairings," in *Identity-Based Cryptography*, M. Joye and G. Neven, Eds. IOS Press, 2008, ch. XI, pp. 188 – 206.

[27] R. Ronan, C. O'hEigeartaigh, C. Murphy, M. Scott, and T. Kerins, "FPGA acceleration of the tate pairing in characteristic 2," in *2006 IEEE International Conference on Field Programmable Technology*. IEEE, Dec. 2006, pp. 213–220.

[28] G. Zhou, H. Michalik, and L. Hinsenkamp, "Complexity Analysis and Efficient Implementations of Bit Parallel Finite Field Multipliers Based on Karatsuba-Ofman Algorithm on FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 7, pp. 1057–1066, Jul. 2010.

[29] M. Morales-Sandoval, C. Feregrino-Uribe, and P. Kitsos, "Bit-serial and digit-serial GF(2^m) Montgomery multipliers using linear feedback shift registers," *IET Computers & Digital Techniques*, vol. 5, no. 2, pp. 86–94, 2010.

[30] D. F. Aranha, E. Knapp, A. Menezes, and F. Rodríguez-Henríquez, "Parallelizing the Weil and Tate Pairings," in *13th IMA International Conference, IMACC 2011*, 2011, pp. 275–295.