An FPGA-based programmable processor for bilinear pairings

Eduardo Cuevas-Farfán[1], Miguel Morales-Sandoval[2], and René Cumplido[3]

[1]Intel Corporation, Guadalajara Mexico, 45019
[2]CINVESTAV Tamaulipas, Cd. Victoria, Mexico, 87130.
[3]INAOE, Puebla, Mexico, 72840.

Abstract

Bilinear pairings on elliptic curves are an active research field in cryptography. First cryptographic protocols based on bilinear pairings were proposed by the year 2000 and they are promising solutions to security concerns in different domains, as in Pervasive Computing and Cloud Computing. The computation of bilinear pairings that relies on arithmetic over finite fields is the most time-consuming in Pairing-based cryptosystems. That has motivated the research on efficient hardware architectures that improve the performance of security protocols. In the literature, several works have focused in the design of custom hardware architectures for pairings, however, flexible designs provide advantages due to the fact that there are several types of pairings and algorithms to compute them. This work presents the design and implementation of a novel programmable cryptoprocessor for computing bilinear pairings over binary fields in FPGAs, which is able to support different pairing algorithms and 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.

1 Introduction

Bilinear pairings appeared in cryptography in the 90s as an attack against cryptosystems based on the discrete-logarithm problem defined over elliptic curves [1, 2]. However in the decade of 2000's, pairings were used in several constructive cryptographic schemes, leading to a new kind of cryptography named Pairing-based Cryptography [11, 15].

[3]Corresponding author. *E-mail address*: miguel.morales@cinvestav.mx.

Computing bilinear pairings requires several finite field arithmetic operations, about 75% of the processing time of a bilinear pairing involves finite field arithmetic operations. Moreover, algorithms and parameters for computing bilinear pairings continue under development. Improvements are constantly reported, sometimes based on different parameters as the kind of elliptic curve, the tower field definition, and the distortion map. Such processing power requirements and changing environment motivate the development of strategies for optimal implementation of bilinear pairings computation.

Although general propose multiprocessors are very powerful, they do not bring support for finite field arithmetic, which follows different rules that conventional arithmetic. Therefore, the emulation of finite field arithmetic leads to low performance software implementations that could not be appropriate for high speed applications like real-time communications. However, software implementations have total flexibility for supporting any pairing algorithm as well as its associated parameters.

The processing time of software implementations can overcome by specialized hardware implementations. However, custom hardware architectures are not able to support 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. Flexible hardware architectures can be more suitable for different applications.

This work presents the design and evaluation in FPGA of a programmable cryptoprocessor for bilinear pairings on elliptic curves over binary fields. This is an extended and improved version of the work reported in [3]. Different to other architectures previously reported in the literature, the proposed cryptoprocessor has a key characteristic the property of programmability. The proposed cryptoprocessor is able to execute different versions of the pairing algorithm as well as different parameters in the pairing computation as the elliptic curve, tower field and distortion map. The cryptoprocessor design outperforms software implementations [4, 5], including GPU implementations [6], requires fewer area resources than custom architectures [7–9], and achieves competitive performance. The methodology followed to construct the proposed cryptogprocessor can be extended to support pairings defined over other fields as $\mathbb{F}_{3^m}$.

The rest of this paper is organized as follows: Section 2 introduces some cryptographic schemes for Pairing-based Cryptography. Also, it presents the mathematical background of bilinear pairings over elliptic curves as well as the theory related to binary field arithmetic. Section 3 presents a review of the state-of-art regarding bilinear pairing implementations. Section 4 describes the proposed flexible architecture for pairing computation, standing out the requirements considered for its design. The strategy for validating the functionality of the proposed architecture is described in Section 5. This section also presents the synthesis results and a comparison against related works. Finally, Section 6 summarizes this work and the achieved contributions. The conclusions and future work are finally presented at the end of this section.

2 Pairing-based Cryptography

The set of cryptographic schemes that make use of bilinear pairings to work defines what is known as Pairing-based Cryptography (PBC). Let G_1 be an additive group of order r and identity element ∞, let G_T be a multiplicative group of order r and identity element 1, a (symmetric) bilinear pairing is a function $e : G_1 \times G_1 \rightarrow G_T$ that maps two elements in G_1 to one element in G_T. Commonly in cryptographic applications, the group G_1 is a set of points in an elliptic curve $E(\mathbb{F}_q)$ defined over the finite field $\mathbb{F}_q$, and the group G_T is the group of elements in the extended finite field $\mathbb{F}_{q^k}$. The mapping e must satisfy the condition of bilinearity, non-degeneracy, and computability [10].

The first key agreement scheme based on bilinear pairing was proposed by Joux in [11]. That scheme called three-party one-round key agreement is based on the Diffie-Hellman problem [12]. This scheme considers that three parties A, B, C, with secrete keys $a, b, c \in \mathbb{Z}_r$ respectively require to agree a common shared key while minimizing the amount of messages being sent. In this sense, A broadcasts aP, B broadcasts bP, and C broadcasts cP. Then, A computes $K_A = e(bP, cP)^a$, B computes $K_B = e(aP, cP)^b$ and C computes $K_B e(aP, bP)^c$. The shared key is $K_{ABC} = K_A = K_B = K_C = e(P, P)^{abc}$. This key agreement scheme is based on the Diffie-Hellman problem for bilinear pairings [13]. This key agreement scheme can be extended to multiple parties by Barua et. al. in [14].

Boneh and Franklin introduced the first practical encryption scheme for PBC [15]. The Boneh and Franklin scheme, denoted as BF-IBE, works under the paradigm of identity-based encryption, which is an asymmetric key scheme where the encryption key is derived from the identity of the receiver, for example the receiver's e-mail. The decryption key is computed by a trusted third party or Private Key Generator (PKG) using also the receiver's identity (i.e., e-mail). The BF-IBE bases its security on the problem known as bilinear Diffie-Hellman problem, which is a reduction of the discrete logarithm problem over elliptic curves.

In BF-IBE, PKG chooses an random element $s \in \mathbb{Z}_r^*$, and sets $P_{pub} = sP$. Two hash functions $H_1 : \{0,1\}^* \rightarrow G_1$ and $H_2 : G_2 \rightarrow \{0,1\}^n$ are chosen and made public. The public key of PKG is P_{pub}, private key of PKG is s. Next, given an receiver's identifier $ID \in \{0,1\}^*$, the receiver's public key is computed as $Q_{ID} = H_1(ID) \in G_1$. The receiver's private key is computed by PKG as $S_{ID} = sQ_{ID}$. For encrypting a message $M \in \{0,1\}^n$, the transmitter chooses a random integer t and computes the encrypted message as the tuple $C = \langle tP, M \oplus H_2(g_{ID}^t) \rangle$, where $g_{ID} = e(Q_{ID}, P_{pub})$. For decryption, the receiver computes the original message M as $M = V \oplus H_2(e(S_{ID}, U))$, where $C = \langle U, V \rangle$ is the encrypted message. The BF-IBE scheme holds because $e(Q_{ID}, sP)^t = e(Q_{ID}, P)^{ts} = e(sQ_{ID}, tP)$.

Boneh, et. al. propose the first PBC-based short-signature scheme in [16]. For example, this scheme produces a signature of length 160 bits, which has the same security level as a 320-bit signature in DSA or 1024-bit length in RSA. The short signature scheme serves as the basis of other signature schemes as [17],

where a scheme for blind and ring signature is proposed. Boldyreva introduced the threshold signatures, multi-signature and blind signature schemes in [18], all them based on pairing computations. Sakai and Kasahara proposed a signature scheme that reduces the number of pairing computations in [19]. An scheme for aggregate signature is reported in [20].

The short signature scheme in [16] consists in three steps: KeyGen, Sing, and Validation.

- KeyGen: Choose and publish a hash function $H : \{0,1\}^* \rightarrow G_1$. Signer chooses a secret key $x \in \mathbb{Z}_r^*$, and publish a validation key $P_{pub} = xP$.
- Sign: The electronic signature for a message $M \in \{0,1\}^*$ is $\sigma = xH(M)$.
- Validation: Given the public key P_{pub}, the message M and the signature σ, the signature is verified by $e(P,\sigma) = e(P_{pub}, H(M))$. The short signature scheme holds because $e(P, xH(M)) = e(P, H(M))^x = e(xP, H(M))$.

2.1 Bilinear pairings over elliptic curves

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

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

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 ∞ [10].

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

$$rP = \underbrace{P + P + P + ... + P}_{r \text{ times } P} \tag{2}$$

A bilinear pairing on an elliptic curve $E(\mathbb{F}_q)$ 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, $\forall\, P, Q, R \in E(\mathbb{F}_q)$ [10]:

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

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 [10].

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

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: $\quad s \leftarrow x_1 + \gamma$, $x_1 \leftarrow \sqrt{x_1}$,$y_1 \leftarrow \sqrt{y_1}$
5: $\quad G \leftarrow s \cdot (x_1 + x_2 + \gamma) + y_1 + y_2 + (1+\gamma) \cdot x_1 + \delta + (s + x_2) \cdot u + v$
6: $\quad x_2 \leftarrow x_2^2$,$y_2 \leftarrow y_2^2$
7: $\quad F \leftarrow F \cdot G$
8: **end for**
9: **return** $F^{(2^{2m}-1)\cdot(2^m+1-\epsilon 2^{(m+1)/2})}$

2.1.1 The Tate and eta pairings

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 on elliptic curves defined in equation 3, where D_Q is a divisor of point Q and f_P is a function that maps D_Q to a finite field field element. Readers are referred to [22] and [10] for more detailed definitions. The computation of equation 3 is divided in two stages: first $f_P(D_Q)$ is computed by the Miller's Algorithm [22], second the exponentiation to the $(q^k-1)/r$-th power of Miller's Algorithm result, called *final exponentiation*, is required [21]. The Miller's Algorithm is a numeric method to construct the function f_P satisfying the conditions of a bilinear pairing [22].

$$\begin{aligned} \tau &: E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k})[r] \rightarrow \mathbb{F}_{q^k}^* \\ \tau(P,Q) &= f_P(D_Q)^{(q^k-1)/r} \end{aligned} \tag{3}$$

Several works have been done to optimize the Tate pairing at an algorithmic level [23–25]. A special case of the original Tate pairing for supersingular curves is the eta pairing (η_T) presented in [23]. 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* $\psi : E(\mathbb{F}_q) \rightarrow E(\mathbb{F}_q^k)$ 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 algorithm 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 [23]. 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 [23], 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 algorithm 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, 26, 27].

Nevertheless, an alternative algorithm for computing the η_T using different parameters for basis is presented in [28]. Both works [23] and [28] 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.

2.2 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 [29]. For cryptographic applications p is typically 2, 3 or a prime number [30]. 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.

2.2.1 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 4. 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 [30]. For some special types of $f(x)$ like trinomials or pentanomials, modular reduction can be computed using only a couple of additions [30].

$$C{=}A \otimes B \mod f(x){=}\sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j x^{i+j} \mod f(x) \tag{4}$$

Squaring, $\bullet^2$, is a special case of the multiplication $A \otimes B$, when $A = B$ that also requires modular reduction [30].

$$C{=}A^2{=}A \otimes A \mod f(x){=}\sum_{i=0}^{m-1} a_i x^{2i} \mod f(x) \tag{5}$$

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 [30].

An interesting identity for any two elements $A, B \in \mathbb{F}_{2^m}$ is depicted in equation 6, which states that squaring is distributive over addition. It can be shown that this identity also holds for squaring root.

$$A^2 + B^2{=}\sum_{i=0}^{m-1} (a_i + b_i)x^{2i} \mod f(x) = (A + B)^2 \tag{6}$$

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 $\mathbb{F}_{2^m}$ [30].

2.2.2 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 $\mathbb{F}_q$ [31]. A sequence of field extensions is called *tower field* [31].

With a tower field, a basis can be constructed to represent elements in $\mathbb{F}_{q^k}$ with elements of $\mathbb{F}_q$ [31]. For example, using the tower field defined in [23], 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 $G = g_0{+}g_1u{+}g_2v{+}g_3uv$ 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 [23] and equation 6, squaring in $\mathbb{F}_{q^4}$ require 4 additions and 4 squaring over $\mathbb{F}_q$:

$$G^2=(g_0 + g_1 + g_3)^2 + (g_1 + g_2)^2 u + (g_2 + g_3)^2 v + g_3^2 uv \tag{7}$$

Raising an element to the q-th power is an operation easily computed using the tower field. For tower field defined in [23] this computation requires only 5 additions over $\mathbb{F}_q$:

$$G^q=(g_0 + g_1 + g_2) + (g_1 + g_2 + g_3)u + (g_2 + g_3)v + g_3 uv \tag{8}$$

3 Related works

Computation of bilinear pairings is not an easy task. It involves a lot of finite field arithmetic operations. Several implementations trying to optimize the computation of bilinear pairings have been proposed. In this section it is presented a brief review of those works.

3.1 Software implementations

The work reported in [32] reaches the fastest implementation of pairing algorithms in software. Authors of that work modify the original algorithms in order to have a better utilization of multithreaded architectures of the new general purpose microprocessors. The Optimal *ate* pairing over prime fields is the main algorithm studied in that work. The best result was achieved with a 64 bit Intel Core i7 microprocessor with a clock frequency of 2.0 GHz. When 8 threads are used for the pairing algorithm computation, a total of 1,034,000 clock cycles are required, resulting in a latency of 0.517 ms. Notice that when only one thread is used, the latency is about 3.23 ms.

Authors of [6] make use of a Graphic Processor Unit (GPU) in order to accelerate the pairing computation. That work implemented the η_T pairing over supersingular curves for ternary fields, reaching a security level of 128 bits. Programming on GPUs generally follows the paradigm of Single Program Multiple Data, therefore authors of that work focused in compute several pairings at the same time. The best results reported a throughput of 332 pairing operations per second, which is equivalent to 3.01 ms per operation.

A different approach for software implementations are specialized software libraries for pairing computations. Specialized software libraries exploit certain features of general propose microprocessors to accelerate pairing computations; for example, special data structures may be defined to reduce the number of memory accesses as well as the memory consumption, or optimized arithmetic primitives may be implemented. The work reported in [5] is a library designed

for small devices like wireless sensor networks. The library proposed in that work was designed for ternary fields and supersingular elliptic curves. Besides the ability for computing pairing algorithms, that library also includes support for hash functions and elliptic curve arithmetic. That design was focused to optimize the memory consumption. For a security level of 66 bits, the library is able to compute the η_T pairing in 5.32 ms.

Software implementations like [32] use very powerful general propose microprocessors, but even with the latest technology they cannot achieve the performance reached with optimized architectures. According with [6], using GPUs to accelerate the pairing computation does not improve the processing time. [5] presented a library suitable for very constrained environments but it does not reach a high level of security.

3.2 No-flexible hardware implementations

Ghosh et. al. presented an architecture able to compute the *eta* pairing in binary fields for a security level of 128 bits in [7]. The architecture presented in that work is based on a Karatsuba-Ofman multiplier which uses a serial-parallel approach. A tradeoff between the serialization and the parallelization in the Karatsuba-Ofman multiplier was performed in order to find the best results. Operations scheduling were optimized mainly during Miller's algorithm stage, and parallelism was used during the final exponentiation step. The area reported in that work is 15,167 slices with a processing time of 0.190 ms, the target device was a Xilinx Virtex 6 FPGA device.

The authors of [8] proposed an custom architecture for computing the *eta* pairing for a security level of 128 bits, being this architecture the fastest one reported in the literature. That architecture implements field multiplication through an hybrid sequential/parallel approach based on the Toeplitz matrix vector products. The authors of that work use an approach based on the Karatsuba-Ofman algorithm in order to reduce the cost of the extended field multiplication. For the final exponentiation, authors optimize the computation by a thoughtful implementation, proposing specific improvements in the computation of inversion over $\mathbb{F}_{2^{4m}}$. The architecture reported in that work was implemented in a Virtex 6 FPGA, resulting in an area of 16,403 slices and a processing time of 0.102 ms. That work is the fastest implementation reported of the η_T pairing for a 128-bit security level.

Beuchat et. al. have reported several works improving the implementation of the η_T pairing computation culminating with [9]. That work presented an architecture for binary fields and another for ternary fields. The central module of both architectures is a full-parallel Karatsuba-Ofman Multiplier. Both cases use a pipelined approach for improving the processing time. A valuable contribution was the introduction of a family of irreducible polynomials which facilitates the computation of square roots. Both architectures were split in two, one architecture for computing the Miller's algorithm and another for computing the final exponentiation, integrating both under a pipeline fashion which also improves the processing time. Although that work only achieves a security

level of 105 bits in binary field with an area of 78,874 slices and a processing time of 18.8μs, the $A \cdot T$ product of 1.41 slices $\times$ ms is so far the smallest in the literature, targeting a Virtex 4 FPGA.

In [33], Cheung et. al. presented an implementation of the Optimal *ate* pairing considering prime fields. In that work a combination of the Reside Number System and the lazy reduction technique is used for reducing the complexity of the arithmetic operations in prime fields. Additionally, an optimization at both architectural and algorithmical level was performed. The target security level was 126 bits. Authors used an FPGA as technology for implementing the architecture achieving a resource consumption of 7032 slices, 32 DSPs and 101KB of memory, and a processing time of 0.573 ms. The target device was a Xilinx Virtex 6.

In [34], the author explores the viability of implementing bilinear pairings over composite-extended fields. The main idea is to represent the field $\mathbb{F}_{q=p^m}$ as a composite field where $m = n \cdot l$ for some n and some l. The arithmetic over $\mathbb{F}_{p^m}$ is implemented using operands over $\mathbb{F}_{p^n}$ which are smaller. In the same way, the arithmetic in the extended field $\mathbb{F}_{p^{km}}$ is implemented using operands over $\mathbb{F}_{p^n}$. The author analyzed the impact of using this kind of fields over the security level in order to do not compromise the security of the system. As a result, that work developed a very compact hardware architecture for ternary fields, at a security level of 128 bits. In order to manage the arithmetic operations over $\mathbb{F}_{p^n}$, the author proposed a codification scheme. This codification could lead to some flexibility, but it was not explicitly intended in that way. For a 128 bits security level using a ternary field, that work require an area of 4,755 slices and a time of 2.23 ms for a Xilinx Virtex 4.

Even [7–9, 33] are very specialized architectures reaching very compact and fast implementation results, their biggest drawback is that only a couple of parameters are configurable. Due to bilinear pairings are no yet standardized, a rigid design could not be the better answer for a changing environment. From [34] it can be inferred some flexibility but the design is not really intended in that way. Moreover, parameters like the elliptic curve, tower field, and distortion map only affect the implementation not the security of the system, so a rigid implementation cannot be adapted to different schemes when these parameters change.

3.3 Flexible hardware implementations

The work reported in [35] presented a programmable architecture for the Tate, *ate*, and R-*ate* pairing over prime fields. The architecture centers its programmability in configurable arithmetic units. Each unit is able to generate its own control sequence according to the desire functionality. Each unit is formed by three independent arithmetic operators. Each arithmetic operator has two inputs of 256 bits, and is able to compute an addition/subtraction/multiplication in $\mathbb{F}_q$. The multiplication is implemented using the Blakley algorithm [36] combined with the Montgomery Ladder technique [37]. The Karatsuba technique is used for reducing the size of the operands. The configurable arithmetic units and

the registers are interconnected by a data access unit which is able to access all registers in parallel. Neither the instruction format nor the instruction set are reported in that work. The security level scoped for that work was 128 bits. That architecture was synthesized on a Xilinx Virtex 4 resulting in an area of 52,000 slices and a processing time of 16.4 ms in the best case.

In [38] it is proposed a coprocessor for computing the η_T pairing in ternary fields. Their design consists in an unified operator for multiplication, addition and cubing over the field $\mathbb{F}_{3^m}$. Multiplication is computed in a digit-serial way, where D coefficients of the multiplication are computed in parallel, after m/D clock cycles the result is completed. Adding and Cubing are easily computed by XOR gates. The three operations are merged into a single operator sharing as much components as possible. A total of 64 working registers implemented by a dual-port RAM are used by that coprocessor to store partial results. The architecture is controlled by a 32 bits instruction. That work was implemented for a security level of 66 bits in a Xilinx Virtex 4, achieving a resource consumption of 1,851 slices and a latency of 0.137 ms. A total of 900 instructions were necessary for computing the pairing algorithm.

Later in [39], Aranha et. al. reported a novel algorithm for computing the Optimal *eta* pairing in supersingular hyper-elliptic curves over prime fields for a security level of 128 bits. The coprocessor proposed in [38] was adapted to support that new algorithm, a total of 4,518 slices and a latency of 5.52 ms were required for that architecture when targeting a Xilinx Virtex 4. A software implementations was performed in that work reaching a best time of 1.1 ms with a 64 bit Intel Core i5 540 whit a clock frequency of 2.53 GHz.

In the work [40], authors report an Application Specific Instruction-set Processor (ASIP). An ASIP consists in a set of instructions and an optimized hardware design to support those instructions. The ASIP reported in [40] focused in the ordinary curves named Barreto-Naehrig, the underlying finite field is the prime field, reaching a security level of 128 bits. The two main parts of that work are the multiplication module and the data access module. Regarding to field multiplication, authors chose a Montgomery multiplier implemented in a multi-cycle approach. The proposed data access module consists in an interface with a dual port RAM memory of 32 bits, able to load and store the pairing operands, which are 256 bits each. That work computes the Optimal *ate* pairing in 15.8 ms, and the η_T pairing in 28.8 ms. The hardware designed in that work was implemented in an ASIC, requiring a total of 97 kGates.

Works introduced in this section show more flexibility in their functionality, but there are some drawbacks that can be outperformed. Despite [38], [35] and [40] are flexible enough to support different algorithms, they only support prime and ternary fields. Arithmetic over binary fields is carry-free, therefore, it usually reports smaller and faster implementations compared to prime and ternary fields. Aranha et al in [39] show how a flexible architecture can be used for implementing new algorithms, nevertheless the processing time can be improved.

4 Cryptoprocessor architecture

The aim of the proposed pairing cryptoprocessor is to bring flexibility to the computation of bilinear pairings. Several algorithms have been proposed and still coninue being proposed for computing pairings. A flexible cryptoprocessor for pairings can allow to manage several parameters in algorithm for computing pairings such as the elliptic curve, the tower field and the distortion map. The programmable cryptoprocessor presented in this work consist of a micro-architectural datapath with its corresponding instruction set.

4.1 Design specifications

The design process was ruled by three general specifications explained in this section. Each specification in the design process is explained and the consequences of that specification in the coprocessor architecture is also detailed.

The first specification is that the architecture support arithmetic in $\mathbb{F}_{2^m}$. All operations required by Miller's algorithm and the final exponentiation can be translated into arithmetic in $\mathbb{F}_{2^m}$. As consequence, the schedule of the instructions should be enough to implement any pairing algorithm for any parameter like the elliptic curve, distortion map or tower field. It is necessary to provide enough memory for storing partial results during the pairing computation. Further, the instructions set to be considered has to cover all the operations in $\mathbb{F}_{2^m}$ used in the pairing algorithms for binary fields, both for the algorithms reviewed in the literature and future algorithms.

The second specification is that only operations among registers is supported. Pairing algorithms do not require operations with constant values rather than 0 or 1, but those constant values can be easily computed. In the case of the value 0, for any element $A \in \mathbb{F}_{2^m}, A \oplus A{=}0$. For the value 1, it can be computed by the equation $A \oplus (A \oplus 1){=}1$. Assuming that element A has a bit vector representation, the operation $A \oplus 1$ is easily computed by the negation of the least significant bit of A. As consequences, the design requires hardware to support the operation $A \oplus 1$, which indeed is just a single NOT gate. Second, only one instruction format is necessary for the arithmetic operation instructions allowing a simpler decoding and a more compact instruction format.

The third specification was defined after analyzing different pairing algorithms. It was noted that, during the Miller's algorithm, multiplication inputs are usually additions among the coordinates of points P and Q. Also, it was noted that for computing extended field arithmetic, input operands are usually additions. So, because additions are widely used during pairing computations, the multiplication, squaring and square root are always preceded by an addition. As consequence of this specification is a tradeoff to define the number of inputs in the addition that precedes the multiplication, squaring and square root. Choosing a two input addition requires less hardware in the implementation, but the pairing algorithm programming requires executing more instructions, that would lead to a longer latency in the pairing computation. On the other hand, considering more inputs in the addition increases the amount of

resources and increases the time delay. However, less instructions are necessary when programing the pairing algorithm so a shorter latency could be obtained. Moreover, the number of addition operands prior a multiplication, squaring or square root is not the same during the pairing computation; after defining a fixed number of inputs in the addition, there are cases when the addition prior a multiplication, squaring or square root requires less operands, so a mechanism to control the number of valid operands in the addition is required.

4.2 Instruction Set Architecture

Based on the design specifications presented in the previous section, an Instruction Set Architecture (ISA) was defined. The ISA has to support all the arithmetic operations reported in the pairing algorithms for binary fields. These operations are addition, multiplication, squaring and square root, which were explained theoretically in section 2.2.1. The second design specification forces the instruction set to include an instruction for the operation $A \oplus 1$. Additionally, other instructions are required for program control in order to support loops, especially the FOR loop, as well as conditional and unconditional jumps.

Prior to define the instruction format, the organization of the working registers was defined. For this, the pairings algorithms reported in [9, 23, 26, 28] were analyzed. A natural way to organize the working registers is by grouping them such that each group stores a single element of the extended field. Each group of registers is called *Bank*. The size of each bank is defined by the size of the extension of $\mathbb{F}_{2^m}$, which is in fact the *embedded degree*. So far, the literature only report pairings over binary fields using supersingular elliptic curves. The embedded degree k of supersingular curves over binary fields is bounded by $k \leq 4$ [21]. Therefore, each bank comprises until four registers each of size m bits. Each bank is intended to store an element in $\mathbb{F}_{2^{km}}$.

This algorithm analysis was used also to define the number of inputs in the addition of third design specification. It was decided to consider four input operands for the additions prior to multiplication, squaring and square root. In this sense, the addition is always computed among the four registers of a specific bank. Each register within a bank is designed with a read enable signal for those cases when the addition requires less inputs.

The complete instruction set is shown in table 1 where S indicates the name of the source bank where the input operands are stored. Inside brackets indicate which registers of bank S are read, for example: $F[0]$ indicates the register 0 of the bank F, and $G[0, 2, 3]$ indicates the registers 0, 2, 3 of bank G. D indicates the destination bank where the result is stored. Inside brackets indicate which registers of bank D are written. Notice that more than one register of bank D can be written at the same time with the same result.

The instruction format is coded into a 16-bit word. Figure 1 illustrates the instruction format. For all instructions: CMD is a 4-bit field indicating the functionality, $OP2$ is a 6-bit field used to indicate the destination bank, $OP1$ is a 6-bit field used to indicate the source bank. The subfields $S1$ and $S0$ are used to select the bank register. Subfields $R3$ to $R0$ are used to select the

Table 1: Instruction set for the proposed architecture.

Name	Description
Addition$(D[], S[])$	Addition up to four elements in $\mathbb{F}_{2^m}$ stored in a bank
Squaring$(D[], S[])$	Squaring of the addition up to four elements in $\mathbb{F}_{2^m}$ stored in a bank.
Square Root$(D[], S[])$	Square root of the addition up to four elements in $\mathbb{F}_{2^m}$ stored in a bank.
LoadMult$(S2[], S1[])$	Load and start a new multiplication. Each operand is the addition up to four elements in $\mathbb{F}_{2^m}$ stored in a bank.
Inc$(D[], S[])$	Increment in 1 the addition up to four elements in $\mathbb{F}_{2^m}$ stored in a bank.
MoveBank$(D[], S[])$	Copy values from the bank S to the bank D.
StoreMult$(D[])$	Store the result of a multiplication operation in the bank D.
SquareGs$(D[])$	Squaring the content for register G_s
Wait(n)	Freeze the IP register for n clock cycles.
Jmp(n)	Unconditional jump to the instruction at address n.
For(n)	Hardware support to FOR-Loop n.
Jz(n)	Conditional jump to the instruction at address n.

specific registers with in the bank, notice that more than one register within the bank can be read at the same time. The same format is used for the control instructions *Jmp*, *For*, *Wait* and *Jz*. For these instructions $OP2$ and $OP1$ act like a 12-bit constant. This coding allows a total of four banks used as source banks and other four banks used as destination banks. If needed, more banks could be addressed by incrementing the number of bits in the fields $OP1$ and $OP2$.

Different to the work previously reported in [3], the proposed instruction set modifies the instruction *IncG0*() to become in the new instruction *Inc*$(D[], S[])$. In the previous work, the hardware for bring support to the operation $A \oplus 1$ was attached to the register G_0; in this new version, the hardware is integrated to the datapath allowing more flexibility to the instruction set. In other hand, the the instruction *SquareGs*$(D[])$ is introduced in this new version. This instructions computes the square root of the value stored in register G_s.

4.3 $\mathbb{F}_{2^m}$ arithmetic modules

4.3.1 Addition

In polynomial basis, an element A in the field $\mathbb{F}_{2^m}$ can be represented as a $(m-1)$-degree polynomial as follows:

$$A = \alpha_{m-1}x^{m-1} + \alpha_{m-2}x^{m-2} + \cdots + \alpha_1 x + \alpha_0 = \sum_{i=0}^{m-1} \alpha_i x^i \tag{9}$$

where A is normally represented as a m-bit vector containing all coefficients defining its corresponding polynomial, that is, $A = (\alpha_{m-1}, \alpha_{m-2}, \cdots, \alpha_1, \alpha_0)$. Due to the polynomial representation, addition in $\mathbb{F}_{2^m}$ is computed using a single bit-wise XOR operation. Notice that there is no carry propagation in field addition.

4.3.2 Square root

Square root is also a cheap operation when using trinomials as irreducible polynomial. Following the algorithm described in section 2.2.1, 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 needed. For the irreducible trinomial $f(x) = x^m + x^a + 1$, the equation 10 performs the computation of $D = \sum d_i x^i{}_{i=0}^{m-1}$, such that $D^2 = A \mod f(x)$ [30].

$$d_i = \begin{cases} a_{2i} & i < (a+1)/2 \\ a_{2i} + a_{2i-a} & (a+1)/2 \leq i < (m+1)/2 \\ a_{2i-a} + a_{2i-m} & (m+1)/2 \leq i < (m+a)/2 \\ a_{2i-m} & (m+n)/2 \leq i < m \end{cases} \tag{10}$$

For the square root module, hardware cost is deduced from equation 10, it can be verified that an addition if required for $(a+1)/2 \geq i < (m+a)/2$. This results in a total of $(m-1)/2$ additions, namely XOR gates.

4.3.3 Squaring

As it is shown in equation 5, 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. Figure 2 illustrate the design. For the squaring module, the cost in area is indeed only the cost of the modular reduction.

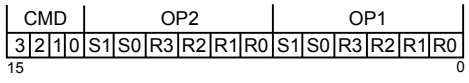

Figure 1: Proposed Instruction Format.

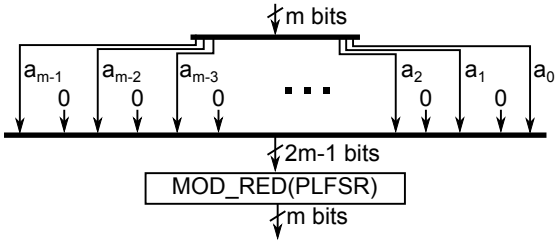

Figure 2: Squaring operation over $\mathbb{F}_{2^m}$.

4.3.4 Modular reduction

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 11, 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 3. 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 [41].

$$xA(x) \mod f(x) = \tag{11}$$
$$(a_{m-2} \oplus [f_{m-1} \odot a_{m-1}], a_{m-3} \oplus [f_{m-2} \odot a_{m-1}], ..., a_0 \oplus [f_1 \odot a_{m-1}], a_{m-1})$$

For computing a modular reduction using PLFSR, consider a polynomial $g(x)$ of degree $2m-1$, The polynomial $g(x)$ can be written as $g(x) = g_2(x)x^m + g_1(x)$. The polynomial $g_1(x)$ does not require modular reduction, but $g_2(x)x^m$ does. The complete modular reductiond is computed following the equation $g(x) \mod f(x) = g_2(x)x^m \mod f(x) + g_1(x)$. The PLFSR cost is m XOR gates as stated above but only computes the first part of the modular reduction; additionally, m XOR gates are necessary to complete the operation. A total of $2m$ XOR gates are required for computing a modular reduction using PLFSR.

4.3.5 Multiplication

Multiplication was implemented using a serial-parallel approach of the field multiplication Karatsuba-Ofman Algorithm (KOA) [7]. 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 (fp-KOA) of $m/4$

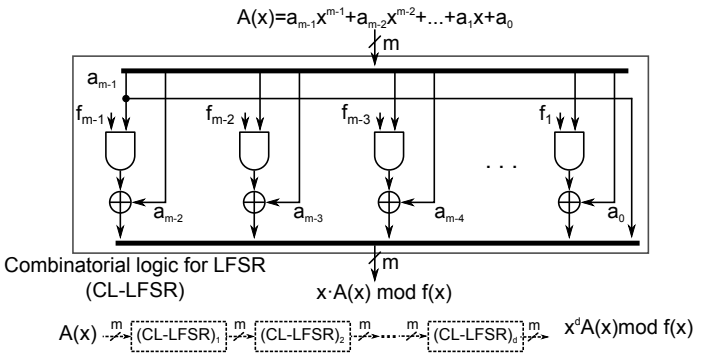

Figure 3: Modular Reduction using Linear Feedback Shifts Registers.

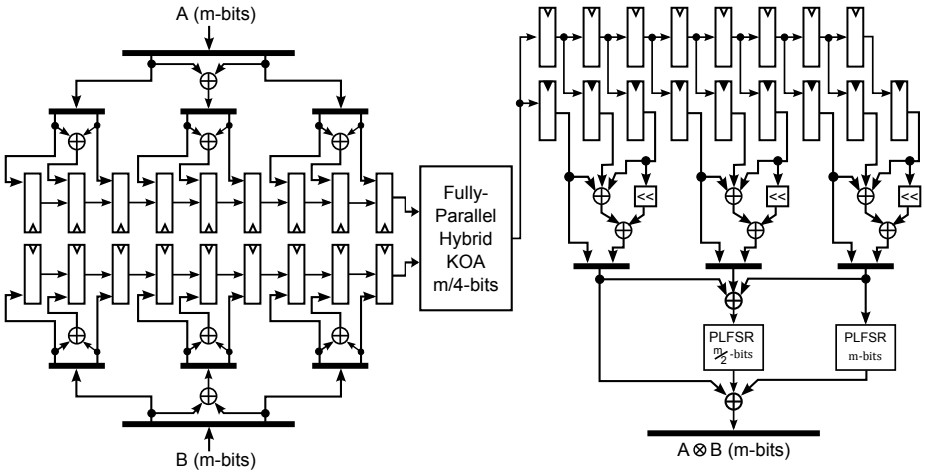

Figure 4: Serial-parallel multiplier based on Karatsuba-Ofman algorithm with modular reduction by Parallel Linear Feedback Registers.

bits. Finally, the 9 partial results are merged according to KOA to complete the multiplication. 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. The output of the serial multiplier was designed to be registered, so after the field multiplication is computed, the result remains available at the module's output.

Further improvements over the original Karatsuba-Ofman algorithm were considered into the design of the serial multiplier. Firstly, the modular reduction was incorporated into the KOA using the technique KOA-LFSR introduced in [42]. The KOA-LFSR only affects the first recursive call of the original KOA, for the successive calls it was implemented the improvement proposed by Fan et. al. in [43] called overlap-free. The overlap-free technique split the inputs bits in

odd indexes and even indexes instead of splitting in the most significant part and less significant part, saving one level of logic in each recursive call consuming the same hardware. Additionally, Zhou et. al. shown in [44] that for small inputs size the schoolbook algorithm present better results in resources and time than KOA. Based on that report, the recursive KOA calls were truncated after s recursions and then the schoolbook algorithm was used for the smaller multipliers. In this case, the truncation was implemented in the fully-parallel KOA of $m/4$ bits.

Different to the serial multiplier proposed in [3], the integrates the modular reduction computation inside the field multiplier as is explained in [42]. The resource consumption and time processing of the multiplier module is improved.

The theoretical cost of the serial multiplier is divided into three parts: the cost of the fully-parallel KOA (fp-KOA) module, the cost before the fp-KOA module, and the cost after the fp-KOA module, see figure 4. To simplify this analysis, only the special case of m being an even number is considered, that is $\lceil m/2 \rceil = m/2$.

Each input of the serial multiplier is first split using the KOA-LFSR strategy; at this level inputs size is m bits and $m/2$ XOR gates are required by each input, so first level needs m XOR gates. In the second split the overlap-free strategy is used for three partial multiplications, at this level inputs size is $m/2$, so $m/4$ XOR gates are required by each input of each partial multiplication, then second level needs $3m/2$ XOR gates. Prior the fp-KOA module, the serial multiplier requires $5m/2$ XOR gates.

The output of the fp-KOA module is of size $m/2$ bits, those outputs are merged to get 3 partial results of size m using the overlap-free technique. Each merge require 3 additions with operands pf $m/2$ bits, requiring $9m/2$ XOR gates. The final merge has to be done using the KOA-LFSR technique. Two PLFSR are required, one for m shifts and other for $m/2$ with a cost of $3m/2$ XORs gates for both PLFSR. Four more additions are required in the final merge, that is $4m$ XOR gates are additionally require. The serial multiplier requires $10m$ XOR gates after fp-KOA module. No extra hardware is necessary for the modular reduction as this operation was integrated using PLFSRs.

Regarding the number of registers, each input requires nine register of size $m/4$ to store the partial inputs. Partial results are momentarily stored in eight registers of size $m/2$, the ninth partial result is taken directly form the output of the fp-KOA module. Nine more register of size $m/2$ are used to store finally the partial results keeping the result available at any time. A total of $13m$ registers are needed for the serial multiplier.

Now lets analyze the hardware cost of the fp-KOA module which truncates the recursion of KOA after s levels of recursion. A single recursion level of KOA adds $4n - 3$ XOR gates where n is the inputs size; for the next recursion level, inputs are halved but three calls are invoked. The equation 12 expresses the

amount of XOR gates required for s recursive calls of KOA.

$$S = \underbrace{(4n-3)}_{\text{1st level}} + \underbrace{3\left(4\frac{n}{2}-3\right)}_{\text{2nd level}} + \underbrace{9\left(4\frac{n}{4}-3\right)}_{\text{3th level}} + \cdots + \underbrace{3^{s-1}\left(4\frac{n}{2^{s-1}}-3\right)}_{\text{s-th level}}$$

$$= \sum_{i=1}^{s} 3^{i-1}\left(4\frac{n}{2^{i-1}}-3\right) \tag{12}$$

After s recursive calls, the KOA algorithm is truncated and the schoolbook algorithm is used. The schoolbook algorithm cost is quadratic, that is w^2 AND gates and $(w-1)^2$ XOR gates are required, where w is the input size of the schoolbook algorithm [44]. After s recursive calls of KOA, $w = n/2^{(s+1)}$. Then, considering that the actual input of the fph-KOA module is size $m/4$, the hardware cost of the fp-KOA module is then expressed by equation 13.

$$S = \left[\sum_{i=1}^{s} 3^{i-1}\left(\frac{m}{2^{i-1}}-3\right) + 3\left(\frac{m}{4}-1\right)^2\right] \cdot XOR + \frac{3m^2}{8} \cdot AND \tag{13}$$

For the entire serial multiplier, the total amount of resources required is given in table 2. Cost is divided in the number of XOR gates, AND gates and registers.

Table 2: Theoretical cost of the serial KOA multiplier.

	XORs	ANDs	REGs
Area	$\sum_{i=1}^{s} 3^{i-1}\left(\frac{m}{2^{i-1}}-3\right) + 3\left(\frac{m}{4}-1\right)^2 + \frac{23m}{2}$	$(3m^2)/8$	$13m$

4.4 Cryptoprocessor datapath

Figure 5 shows the proposed datapath. It contains six bank registers F, G, H, I, V, W. Only banks F, G, H and I can be used as source banks for arithmetic operations. For the multiplication, one operand comes from bank F or H and the other comes from bank G or I. Only banks V, W and G can be used as destination banks for arithmetic operations. 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, and from W to G or I. There are two extra registers, F_s and G_s, used as alternative inputs for multiplication. Additionally, the register G_s is connected to the squaring module.

An 4-input addition is performed at the output of banks F and H, and other one is placed at the output of banks G and I using a 4-input XOR each. A 3-input multiplexer at the input of each addition input is used in order to indicate which registers are being added and which not. Notice that banks F

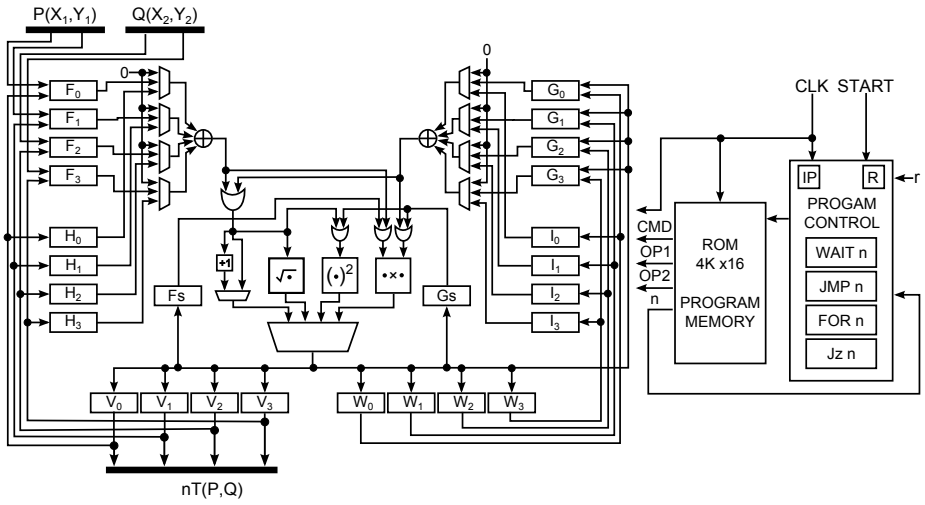

Figure 5: Proposed architecture for computing bilinear pairings over binary fields.

and G also have two inputs, which means that there is a multiplexer at the input for selecting which of the two inputs will be written. Some OR gates are used to drive the right input to the arithmetic module. All arithmetic modules compute their respective operation at the same time, a 4-input multiplexer is used for selecting the right result. The extra hardware for computing $A \oplus 1$ is placed in the path of the addition.

Different to the datapath presented in [3], in this new version the bank H and I are configured to be source banks for arithmetic operations. With this modification, the datapath reaches more flexibility in their operation.

4.4.1 Program control

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 the next instruction to be executed. A total of 4K instructions can be addressed. The "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 next 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 of the points P and Q, see algorithm 1 of [23]. But so far the proposed pairing 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.

4.5 Theoretical cost analysis

The theoretical cost in hardware of the complete cryptoprocessor is the addition of each arithmetical module cost, plus the extra hardware required for the interconnection. In the following cost analysis, only is considered the usage of trinomial as irreducible polynomial for defining $\mathbb{F}_{2^m}$.

The addition at the output of banks F and G is performed by a 4-input XOR, which in fact can be computed by three XOR gates of 2-inputs. Each addition is of size m bits. In total $6m$ XOR gates are required for those additions. As depicted in figure 5, four OR gates of m bits are used to drive correctly the data. $4m$ OR gates are required in the cryptoprocessor datapath.

Without counting the cost of arithmetic modules, the most hardware usage relays on a series of multiplexers. One multiplexer of 4 inputs is used to select the output of the arithmetic modules. A single multiplexer of 4-inputs can be implemented with three multiplexers of 2 inputs. Prior the additions, there are eight multiplexers of 3 inputs, each one implemented with two multiplexers of 2 inputs. At the input of banks F and G there are eight multiplexers of 2 bits per bank. Each multiplexer is of size m bits. The multiplexer for selecting. In total there are $27m$ multiplexers of 2 bits. The amount of registers is very straight forward. There 6 banks of 4 registers each, plus 2 extra registers (Fs and Gs), each one of m bits. $26m$ registers are required in the cryptoprocessor datapath.

The hardware support for control instructions is performed using 12-bits comparators, 12-bits multiplexers and 12-bits registers. The hardware cost and time delay of this module is very small compared with the cost of the arithmetic modules and the rest of the architecture. For these reasons the cost analysis has been depreciated as do not represent a significant cost.

In table 3, it is summarized the total cost of the proposed cryptoprocessor. Notice that most of the area computed theoretically is due to the multiplication module. The other modules with more area consumption are the multiplexers used in the datapath. Also notice that despite the datapath uses several registers, the serial multiplication consumes a third part of the total registers.

4.6 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 F_0 to F_3 contain the values x_1, y_1, x_2, y_2 as shown in figure 5. The addition $G_0 = F_0 + F_2$ is computed by the instruction $Addition(G[0], F[0, 2])$, while $W_2 = G_0+G_1+G_2+G_3$ 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 $G_0 = y_1 + y_2$. When $\beta = -1$ the instruction $Inc(G[0], F[1, 3])$ is required for computing $G_0 = y_1 + y_2 + 1$.

Table 3: Theoretical cost of the proposed cryptoprocessor.

Module	XORs	ANDs	ORs	MUXs	REGs
Addition	$6m$	-	-	-	-
Multiplication	$\sum_{i=1}^{s} 3^{i-1}\left(\frac{m}{2^{i-1}}-3\right) + 3\left(\frac{m}{4}-1\right)^2 + \frac{23m}{2}$	$(3m^2)/8$	-	-	$13m$
Squaring	$2m$	-	-	-	-
Square Root	$(m-1)/2$	-	-	-	-
Datapath	-	-	$4m$	$27m$	$26m$
Total	$\sum_{i=1}^{s} 3^{i-1}\left(\frac{m}{2^{i-1}}-3\right) + 3\left(\frac{m}{4}-1\right)^2 + \frac{(40m-1)}{2}$	$(3m^2)/8$	$4m$	$27m$	$39m$

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) *Inc* $(G[0], F[0])$: compute $s = x_1 + 1$ and store it in G_0
2) *Addition*$(F_s, G[0])$: move s to F_s
3) *SquareRoot*$(G[0], F[0])$: compute $G_0 = \sqrt{x_1}$
4) *Inc*$(G[0], G[0])$: compute $G_0 = \sqrt{x_1} + 1$
5) *Addition*$(G[1], F[2])$: move x_2 to $G1$
6) *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 $G_0 = \sqrt{x_1}$
2) *Addition*$(G[1], F[2])$: move x_2 to G_1
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 [30], 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 8 computes this operation for the tower field presented in [23]. Nevertheless, in [28] a different tower field is used, so raising an element to the q-th power is computed by equation 14. Both equations, 8 and 14 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.

$$G^q = (g_0 + g_2) + (g_2)x + (g_1 + g_3)x^2 + g_3x^4 \tag{14}$$

5 Experimental results

5.1 Validation estrategy

The proposed architecture was implemented using VHDL as a description language. For design validation of arithmetical modules, C/C++ routines based on the library Miracl[1] were used to generate 1000 random test data vectors for each module. A test bench was written to read the test vector from a file, to instantiate a particular module and to simulate its behavior for all the test vector generated. Xilinx ISim 13.2 was used as simulation environment. For all experiments, the underlying finite field was $\mathbb{F}_{2^{1223}}$ defined by the trinomial $f(x) = x^{1223} + x^{255} + 1$, in order to reach a security level of 128-bits [27].

The full cryptoprocessor was validated using a similar strategy. Two versions of the η_T pairing were used for testing the correct functionality of the datapath. First version named Barreto-Beuchat uses the Miller's algorithm presented in [23] and the final exponentiation introduced in [26]. Second version named Ronan uses the version of the η_T algorithm introduced in [28]. Finally, in order to validate the cryptoprocessor under some Pairing-based Cryptography application, the Identity-based Key Encapsulation Mechanism (ID-KEM) introduced in [45] was implemented in software using the C/C++ Miracl library. ID-KEM scheme consists in four algorithms which in conjunction have the purpose to establish a shared key among two parties, starting from the receiver's public key. In this case, the receiver's public key is an identifier related to the receiver's identity. ID-KEM requires the computation of two bilinear pairings, one extra pairing can be computed in order to ensures that the receiver's private key is well generated. A total of 35 different identifiers were tested, resulting in a total of 105 bilinear pairing computations. This validation was performed for the two versions of the η_T pairing algorithm presented previously.

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

5.2 Implementation results

The design was implemented in FPGA devices, targeting both Xilinx Virtex-6 (xc6vlx130t) and Xilinx Virtex-4 (xc4vlx200) devices. FPGAs was chosen with comparative purposes, the design is not specific for FPGA devices. Even the Virtex-4 is old device, it was chosen for comparative purposes.

Program memory was implemented with Xilinx's Block Memory Generator LogiCORE. Xilinx ISE 13.2 was used as synthesis tool using flags by default except for flags *-iobuf FALSE* and *-register_balancing YES*. The *-iobuf* states if the synthesis tool attempts to match the inputs and outputs of the design with real pins on the target devices. The flag *-register_balancing* moves registers through combinatorial logic to evenly distribute the paths delay between registers, increasing the maximum clock frequency.

5.2.1 Serial multiplier

As mentioned in section 4.3, for implementing the fully-parallel hybrid KOA module inside the serial multiplier module, the KOA algorithm was truncated after s recursive calls. In [44] it was shown experimentally that the optimal value of s depends on the implementation technology. In this sense, the serial multiplier was synthesized using several values of s.

Table 4: Implementation results for serial multiplier using different values of s.

s	LUTs	FFs	Minimum period	Latency	$A \cdot T$
1	40,969	18,244	2.817 ns	25.353 ns	1.04
2	34,734	17,306	2.647 ns	23.823 ns	0.83
3	31,262	17,344	2.882 ns	25.938 ns	0.81
4	28,518	17,507	3.144 ns	28.296 ns	0.80
5	28,990	15,904	7.902 ns	35.559 ns	1.03
6	31,720	15,904	9.325 ns	83.925 ns	2.66

Since the proposed design is based on the serial multiplier reported by Ghosh et. al. in [7], a results comparison against that work is performed. In table 5 the best implementation achieved for the serial multiplier is compared against [7]. Notice that the proposed serial multiplier includes the modular reduction inside the multiplier, different to the multiplier reported in [7] which additionally requires a modular reduction module.

Table 5: Comparative of $\mathbb{F}_{2^m}$ serial multipliers.

Design	LUTs	Minimum period	Clock cycles	Latency	$A \cdot T$	Requires reduction?
This	28,518	3.144 ns	9	28.296 ns	0.80	NO
[7]	30,148	4 ns	10	40ns	1.21	YES

5.2.2 Cryptoprocessor

Table 6 shows the implementation results of the synthesis process. The required area is 13,967 slices for a Virtex 6 device. From this area, about 51.04% 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 35.02%. The remaining 13.94% of the area is used for the rest of the arithmetic modules and the program control unit. Notice how the area is closely related with the technology, for a Virtex 4 a total of 45,917 slices 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 11 levels of logic. The maximum frequency that the architecture can operate with is 183.8 MHz for a Virtex 6. 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 72.31 MHz.

Results reported in this work outperforms the results previously reported in [3]. With the improvements done, the amount of hardware resources in about 15% in the case of a Virtex 6. While for a Virtex 4, resource consumption is reduced in about 10%. The reason of this reduction relies mainly in the new organization of the working registers, which allows the synthesis tool to find a more compact implementation. Maximum frequency in any case remains almost the same.

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

Table 6: Implementation results of the proposed architecture for two different version of η_T algorithm.

η_T version	Device	Area (Slices)	Program memory (kbits)	Clock cycles ($\times 10^3$)	Maximum frequency (MHz)	Latency (us)
Barreto-Beuchat	Virtex 6	13,967	5.3	51.5	183.8	280
	Virtex 4	45,917	5.3	51.5	72.31	712
Ronan	Virtex 6	13,967	10.3	57.6	183.7	313
	Virtex 4	45,917	10.3	57.6	72.31	797

A comparative with state-of-art in software implementations for computing bilinear pairings is presented in table 7. Software implementations are in fact

flexible implementations that uses general purposes microprocessors. This comparative includes the fastest implementation of pairing algorithms in GPUs, and a specialized software library for pairing computations. This comparison only considers the processing time. In all cases the proposed cryptoprocessor computes the pairing algorithm faster. Although general purposes microprocessors or GPUs are very powerful technologies, they are limited to their own general purpose instruction set and fixed size operands. In this way, the proposed cryptoprocessor may be used as a specialized co-processor for pairing computations, leaving the rest computations of any Pairing-based protocol to be executed by the general purpose microprocessor.

Table 7: Comparative of the proposed architecture with software implementations.

Ref.	Device	Field	Maximum frequency (MHz)	Latency (us)
Barreto-Beuchat	Virtex 6	$\mathbb{F}_{2^m}$	183.8	280
Ronan	Virtex 6	$\mathbb{F}_{2^m}$	183.8	313
[32]	Intel Core i7	$\mathbb{F}_{2^m}$	2,000	517
[32]	Intel Core i7	$\mathbb{F}_{2^m}$	2,000	3,228
[6]	NVidia GTX 480	$\mathbb{F}_{3^m}$	1,401	3,010
[5]	MICAz	$\mathbb{F}_{3^m}$	7.383	2.45×10^6

A comparison against state-of-art custom implementations of the η_T pairing for binary fields is presented in table 8. All works reported in this table reach a security level of 128 bits except for [9], which achieve a security level of 105 bits. For this comparison, only the implementation results of the Barreto-Beuchat version of the η_T pairing is considered. It is noticed that custom implementations are faster than the proposed architecture, which is expected because custom implementations make use of parallelization and other techniques in order to achieve faster results. A comparison with [9] make this statement more evident, but the cost of faster architectures is the use of more hardware resources, which is also evident in this comparison. It can be observed that the area consumed by the proposed cryptoprocessor is very similar to works [7] and [8]. The $A \cdot T$ product reached in this work is 3.91, which is just 1.35x bigger than [7] and 2.3x bigger than [8]. Compared to [9] the $A \cdot T$ product is 2.77x bigger, but notice that [9] only reaches a security level of 105 bits. These results show that custom architectures are slightly faster/smaller than the proposed design, but with all the flexibility achieved with the proposed architecture it is a fair cost.

Table 9 compares the proposed cryptoprocessor against works in the literature that exhibit some degree of flexibility. Notice that in the literature it is no reported a flexible solution for binary fields. In table 9, the results compared are from the Virtex 4 as works [35, 38] used the same device. For the case of work [40], authors implement their architecture in ASIC using a 30 nm standard cell library. Even the work reported in [38] reports a smaller area and faster

Table 8: Comparative of the proposed architecture against custom architectures for binary fields.

Ref.	Device	Area (Slices)	Maximum frequency (MHz)	Clock cycles ($\times 10^3$)	Latency (us)	$A \cdot T$ (Slices $\times$Seg.)
Barreto-Beuchat	Virtex 6	13,967	183.8	51.5	280	3.91
	Virtex 4	45,917	72.31	51.5	712	32.69
[7]	Virtex 6	15,167	250	47.6	190	2.88
[8]	Virtex 6	16,403	267	27.3	102	1.7
[9]*	Virtex 4	78,874	130	2.4	18.8	1.41

*That work targeted a security level of 105 bits.

computation time, that work only achieve a security level of 66 bits whereas the proposed architecture achieves a security level of 128 bits. Comparing area consumption and processing time with the work reported in [35], the proposed cryptoprocessor outperforms that work in both parameters. Comparison with the ASIP reported in [40] is harder because the target devices of this thesis is a FPGA, not an ASIC; anyway it can be notice that the proposed architecture is able to execute a pairing algorithm 20x faster.

Table 9: Comparative of the proposed architecture against works in the literature with some degree of flexibility.

Ref.	Field	Area (Slices)	Maximum frequency (MHz)	Clock cycles ($\times 10^3$)	Latency (us)	$A \cdot T$ (Slices $\times$Seg.)
Barreto-Beuchat	$\mathbb{F}_{2^m}$	45,917	72.31	51.5	712	32.69
Ronan	$\mathbb{F}_{2^m}$	45,917	72.31	57.6	797	36.59
[38]*	$\mathbb{F}_{3^m}$	1,851	203	27.8	137	0.25
[35]	$\mathbb{F}_p$	52,000	50	1,729	34,600	1,799
[40]**	$\mathbb{F}_p$	97kGates	338	N/R	15,800	N/A

*That work targeted a security level of 66 bits.
**That work targeted an ASIC using a 30 nm standard cell library.

From the results obtained, the proposed cryptoprocessor is a very feasible solution for bilinear pairing computation over binary fields. The design of a programmable hardware 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.

6 Conclusion and future work

This work discussed the design and evaluation of a flexible cryptoprocessor able to manage several parameters for computing bilinear pairings, as the elliptic curve, the tower field, the distortion map, or the version of the pairing algorithm. These parammeters do not interferer with the security of cryptographic applications but affect the performance.

The design follows a modular approach so it can be constructed to support bilinear pairings defined on other fields as $\mathbb{F}_{3^m}$. The results show that the proposed design requires a similar amount of resources compared to related works. In addition, the processing time is shorter than other flexible architectures, inclusive the proposed architecture is faster that software implementations. Also, the proposed architecture is very competitive in area and processing time against custom architectures of the state-of-art. In conclusion, the proposed design preserves the flexibility of the software while conserves the acceleration in pairing computations of custom architectures. The compact instruction format allows smaller programs than related works, therefore using less program memory.

A thorough optimization process can be done to improve the maximum clock frequency, being the pipelining technique a first approach. As this pairing cryptoprocessor was originally conceived within a whole system, the control unit can be replaced by a general soft-processor like a PicoBlaze and a communication interface to send/receive the operands will be implemented in a HW/SW codesing; 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] G. Frey and H.-G. Ruck, "A Remark Concerning m-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves," *Mathematics of Computation*, vol. 62, no. 206, p. 865, Apr. 1994.

[2] 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.

[3] E. Cuevas-Farfán, M. Morales-Sandoval, R. Cumplido, C. Feregrino-Uribe, and I. Algredo-Badillo, "A programmable FPGA-based cryptoprocessor for bilinear pairings over $\mathbb{F}_{2^m}$." in *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on*. Darmstadt, Germany: IEEE, 2013.

[4] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López, "Faster Explicit Formulas for Computing Pairings over Ordinary Curves," in *Advances in Cryptology - EUROCRYPT 2011*, Kenneth G. Paterson, Ed. Tallinn, Estonia: Springer Berlin / Heidelberg, 2011, pp. 48–68.

[5] X. Xiong, D. S. Wong, and X. Deng, "TinyPairing: A Fast and Lightweight Pairing-Based Cryptographic Library for Wireless Sensor Networks," in *2010 IEEE Wireless Communication and Networking Conference*. IEEE, Apr. 2010, pp. 1–6.

[6] Y. Katoh, Y.-j. Huang, C.-m. Cheng, and T. Takagi, "Efficient Implementation of the η_T Pairing on GPU," *Cryptology ePrint Archive: Report 2011/540*, no. 540, 2011. [Online]. Available: http://eprint.iacr.org/2011/540

[7] 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.

[8] 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.

[9] 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.

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

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

[12] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[13] R. Dutta, R. Barua, and P. Sarkar, "Pairing-Based Cryptographic Protocols: A Survey," *Cryptology ePrint Archive: Report 2004/064*, no. 64, 2004. [Online]. Available: http://eprint.iacr.org/2004/064

[14] R. Barua, R. Dutta, and P. Sarkar, "Extending Jouxs Protocol to Multi Party Key Agreement," in *Progress in Cryptology - INDOCRYPT 2003*, 2003, pp. 205–217.

[15] D. Boneh and M. Franklin, "Identity-Based Encryption from the Weil Pairing," in *Advances in Cryptology - CRYPTO 2001*, vol. 2139, 2001, pp. 213–229.

[16] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," *Journal of Cryptology*, vol. 17, no. 4, pp. 297–319, Jul. 2004.

[17] F. Zhang and K. Kim, "ID-Based Blind Signature and Ring Signature from Pairings," in *Advances in Cryptology - ASIACRYPT 2002*. Springer Berlin / Heidelberg, 2002, pp. 533–547.

[18] A. Boldyreva, "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme," in *Public Key Cryptography - PKC 2003*, 2003, pp. 31–46.

[19] R. Sakai and M. Kasahara, "ID based Cryptosystems with Pairing on Elliptic Curve," *Cryptology ePrint Archive: Report 2003/054*, no. 54, 2003. [Online]. Available: http://eprint.iacr.org/2003/054

[20] X. Cheng, J. Liu, and X. Wang, "Identity-Based Aggregate and Verifiably Encrypted Signatures from Bilinear Pairing," in *Computational Science and Its Applications - ICCSA 2005*. Springer Berlin / Heidelberg, 2005, pp. 1046–1054.

[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] 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.

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

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

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] G. R. Blakely, "A Computer Algorithm for Calculating the Product AB Modulo M," *IEEE Transactions on Computers*, vol. C-32, no. 5, pp. 497–500, May 1983.

[37] M. Joye and S.-M. Yen, "The Montgomery Powering Ladder," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, ser. Lecture Notes in Computer Science, B. S. Kaliski, c. K. Koç, and C. Paar, Eds., vol. 2523. Berlin, Heidelberg: Springer Berlin Heidelberg, Feb. 2003, pp. 291–302.

[38] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi, "Algorithms and Arithmetic Operators for Computing the ηT Pairing in Characteristic Three," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1454–1468, Nov. 2008.

[39] D. F. Aranha, J.-L. Beuchat, J. Detrey, and N. Estibals, "Optimal Eta Pairing on Supersingular Genus-2 Binary Hyperelliptic Curves," in *Topics in Cryptology CT-RSA 2012*, ser. Lecture Notes in Computer Science, vol. 7178, 2012, pp. 98–115.

[40] D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, and R. Mathar, "Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves," in *Cryptographic Hardware and Embedded Systems - CHES 2009*, C. Clavier and K. Gaj, Eds. Springer Berlin / Heidelberg, 2009, pp. 254–271.

[41] 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.

[42] E. Cuevas-Farfán, M. Morales-Sandoval, A. Morales-Reyes, C. Feregrino-Uribe, I. Algredo-Badillo, P. Kitsos, and R. Cumplido, "Karatsuba-Ofman Multiplier with Integrated Modular Reduction for $\mathbb{GF}(2^m)$," *Advances in Electrical and Computer Engineering*, vol. 13, no. 2, pp. 3–10, 2013.

[43] H. Fan, J. Sun, M. Gu, and K. Lam, "Overlap-free KaratsubaOfman polynomial multiplication algorithms," *IET Information Security*, vol. 4, no. 1, p. 8, 2010. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-ifs.2009.0039

[44] 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.

[45] L. Chen, Z. Cheng, J. Malone-Lee, and N. P. Smart, "Efficient ID-KEM based on the SakaiKasahara key construction," *IEE Proceedings - Information Security*, vol. 153, no. 1, p. 19, 2006.