Hardware Architecture for Pairing-Based Cryptography

by

EDUARDO CUEVAS FARFÁN

Ing. UDLAP

A Dissertation
Submitted to the Program in Computer Science,
Computer Science Department
in partial fulfillment of the requirements for the degree of

MASTER IN COMPUTER SCIENCE

at the

National Institute for Astrophysics, Optics and Electronics
November 2013
Tonantzintla, Puebla

Advisors:

Dr. René Armando Cumplido Parra, INAOE
Dr. Miguel Morales Sandoval, CINVESTAV

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. The computation of bilinear pairings relies on arithmetic over finite fields. The bilinear pairing is the most time-consuming in Pairing-based cryptosystems which has motivated its implementation in dedicated hardware. 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 preferred in order to support changes in the specifications. This thesis 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 such 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.

Resumen

Los emparejamientos bilineales sobre curvas elípticas son un área de investigación creciente en criptografía. Los primeros protocolos basados en emparejamientos bilineales fueron propuestos en el año 2000 y actualmente no han sido estandarizados. El cálculo de emparejamientos bilineales recae en operaciones aritméticas de campo finito. El emparejamiento bilineal es la operación que más tiempo consume en los criptosistemas basados en emparejamientos, lo que ha motivado su implementación en hardware dedicado. En la literatura, diferentes trabajos se han enfocado en el diseño de arquitecturas hardware dedicadas en la implementación eficiente de dichas operaciones aritméticas, pero en un ambiente de trabajo no estandarizado un diseño flexible es preferido con el propósito de dar soporte a los cambios en las especificaciones de los protocolos. Esta tesis presenta el diseño y la implementación de un novedoso criptoprocesador programable para el cálculo de emparejamientos bilineales sobre curvas elípticas en FPGA, que es capaz de soportar diferentes algoritmos de emparejamiento y sus parámetros correspondientes como son la curva elíptica, la torre de campos y el mapa de distorsión. Los resultados demuestran que el criptoprocesador propuesto alcanza un alto grado de flexibilidad con un tiempo de procesamiento y uso de recursos competitivos cuando es comparado con diseños dedicados para emparejamientos definidos sobre curvas elípticas singulares y supersingulares, a un nivel de seguridad de 128 bits.

Agradecimientos

Quiero extender mi más profundo agradecimiento al Consejo Nacional de Ciencia y Tecnología por el financiamiento otorgado para la realización de mis estudios de maestrá a traves de la beca número 325137. De la misma manera quiero agradecer al Instituto Nacional de Astrofísica, Óptica y Electrónica por todos los servicios recibidos durante mi estancia en la institución.

Quiero agradecer de una forma muy especial a mis asesores, el Dr. René Cumplido Parra y Dr. Miguel Morales Sandoval por su paciente guía durante el desarrollo de esta tesis. Todos los consejos, sugerencias y correcciones siempre serán muy agradecidos ya que sin ellos este trabajo no hubiera rendido los frutos que al día de hoy ha rendido.

De la misma manera quiero agradecer a mi comité de sinodales, la Dra. Claudia Feregrino Uribe, el Dr. Miguel Arias Estrada y el Dr. Enrique Muñoz de Cote, por su valiosas revisiones que permitieron que este trabajo tenga la calidad que alcanzó.

Un agradecimiento especial a Dra. Claudia Feregrino, Dr. Ignacio Algredo y Dra. Alicia Morales por toda su ayuda y colaboración para la publicación de los artículos derivados de esta tesis.

Aunque ajenos a la institución, quiero agradecer a todas las personas con las que trabaje durante los 9 meses de mi estancia en Intel Guadalajara por todos las experiencias y conocimientos compartidos en este tiempo, en especial para Joaquín García, Remedios Villafranco, Laura Fuentes, Gabriel Labrada, Arie Herrera, Iyaly Hernandez, Leoswaldo Mancilla, Victor Escamilla y Omar Alvizo.

A todos mis amigos de la maestría les agradezco mucho todas las experiencias, compañia y ayuda que brindaron durante esta aventura de más de 2 años. Siempre recordaré el trabajo en los cubos, las idas a cenar, las tardes de cineclub y por supuesto el café; Paco, Harold, Miguel, Merlo, Yared, Paty, Kenshi, Marisol, Ivan, Metzli, Daniel, Richy, Alberto, Lindsey, Chang, Josue y David. Un agradecimiento especial a Harold y Miguel por ayudarme con todos los trámites para mi graduación.

Y sin duda alguna agradezco a mi familia por todo el apoyo me han dado a lo largo de los años para afrontar cada uno de los retos que me pongo. Sin su apoyo incondicional jamas podría alcanzar nada de lo que me he propuesto. Empezando con mis papas Leonor y Oscar; mis hermanos Fabricio y Elva; mis abuelos Silvia, Salvador y Leonor; mis tíos y primos. Gracias por todas sus porras.

Y por último pero no menos importante, quiero dar un agradecimiento muy especial a Kenia Ortiz por estar a mi lado desde hace algunos años, por siempre motivarme, inspirarme y siempre estar ahí a pesar de la distancia. Gracias por todo tu cariño y paciencia.

With love to Kenia, for being my lighthouse in the nights.

Contents

Preface

Information is perhaps the most important asset in current times. Several technologies have been developed in order to bring protection to such important asset, being cryptography the most powerful tool of information security technologies. Four fundamental aspects are undertaken by cryptography: confidentiality, data integrity, authentication and non-repudiation. Public key cryptosystems used nowadays are based on keys contained in certificates, which require a complex infrastructure for real life applications.

Identity-based cryptography is a new trend of asymmetric key cryptosystems. Different to other asymmetric schemes, Identity-based cryptography states that the public key of any entity should be derived from the identity of that entity; with this paradigm, the infrastructure required by other kinds of cryptography like the Public Key Cryptography is simplified. Although the paradigm of Identity-based cryptography was early proposed in 1985 by Shamir [1], it was until 2001 when Boneh and Franklin proposed the first practical scheme for encryption based on identity [2]. The encryption scheme proposed by Boneh and Franklin makes use of bilinear pairings over elliptic curves to work. Since that work, a growing amount of research focused on cryptographic schemes based on bilinear pairings has been carried out, leading to the new kind of cryptography named Pairing-based cryptography.

Nevertheless, algorithms and parameters for computing bilinear pairings are still under development. Improvements are constantly reported, sometimes based on different parameters as the kind of elliptic curve, the tower field definition or distortion maps. 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 preferred, being flexible architectures more suitable for different applications.

This thesis proposes a programmable cryptoprocessor able to compute bilinear pairings in elliptic curves over binary fields $\mathbb{F}_{2^m}$. Different to other architectures previously reported in the literature, the proposed cryptoprocessor is the only one being programmable and flexible. Furthermore, it is the only one designed for bilinear pairings on elliptic curves defined over $\mathbb{F}_{2^m}$. The proposed cryptoprocessor is able to execute different versions of the pairing algorithm. Also, it is able to support different parameters in the pairing computation as the elliptic curve, tower field and distortion map. The proposed cryptoprocessor is able to support different parameters such as the elliptic curve, tower field and distortion map as well as different versions of the pairing algorithm. The proposed cryptoprocessor outperforms software implementations, including GPU implementations, requires fewer area resources than custom architectures, and achieves competitive performance.

Chapter 1

Introduction

1.1 Information security

Information is one of the most important assets for current society, from common people to global organizations. For common people, their personal data represents their privacy; for global companies, information represents their competitive advantage over other companies; for governments, information represents the security for their citizen. Information in the incorrect hands can be used in many malicious ways. Most of the information in our days is in digital format, which makes information protection a real challenge. Digital information can be stored in very small devices without being noticed by anybody. Many times, it is not required a physical contact in order to access such a high value asset.

The current information security solutions are a variety of standardized security protocols. Nevertheless, with current information technologies like cloud computing, pervasive computing, mobile computing, embedded and distributed systems, etc., new types of attacks and vulnerabilities emerge, being necessary to make adjustments and/or to propose new protocols in order to keep information secure.

1.1.1 Modern cryptography

Cryptography is defined as the art and science of keeping messages secure. The history of cryptography is very ancient, but in modern days it has been formalized as a branch of applied mathematics that makes use of algorithms in order to provide information security services.

Information security systems are designed to cover four fundamental aspects: confidentiality, data integrity, authentication and non-repudiation. These four aspects are called information security services [3]. Confidentiality is the property that gives the warranty that the information is only accessible by authorized entities. Even if an unauthorized entity is able to access a confidential file, that entity should not be able to understand its content. Integrity refers to the warranty that the information has not been modified by some unauthorized entity. Even if the information comes from a reliable source, an intruder may be able to modify the information. Authentication consists in the ability of verifying the origin of the information. This service validates the identity of the source, and also prevents phishing. Non-repudiation refers to the warranty that the source of some information cannot falsely deny the origin of that information.

Confidentiality then is achieved through an encryption algorithm whose objective is to make the original file "unreadable", so the information can be stored or transmitted without risk of being disclosed. The encryption algorithm encodes and mixes the bits of information by substitutions and transpositions, with the goal of obtaining statistic diffusion and confusion, so the information becomes "unreadable". Only the authorized entities are able to revert the process by a decryption algorithm and so retrieve the information [4].

In the past, cryptographic algorithms were kept secret but this practice was not reliable because, once the algorithm was disclosed the system was compromised. Nowadays, cryptographic algorithms are public and the security relays on a parameter called *key*. A key is an input parameter in the encryption and decryption algorithms which determines the output. A security system should have a very large amount of keys, and only the authorized entities should know the correct key to encrypt and decrypt the information. According with Schneier, a cryptosystem is the set of algorithms, all possible keys, all possible information, and respective encrypted information [4].

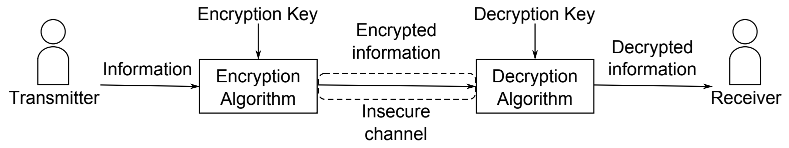


Figure 1.1: Encryption and decryption scheme.

In figure 1.1, a cryptosystem for encryption and decryption is shown viewed as a communication system. In this illustration, a transmitter sends some information through an insecure channel, so the information is encrypted by an encryption algorithm using an encryption key. The encrypted information can only be disclosed using the decryption algorithm and the right decryption key. In this sense, if a wrong decryption key is used, the output of the decryption algorithm will be "unreadable". Notice that the same key can be used for encryption and decryption preserving the security of the scheme [4].

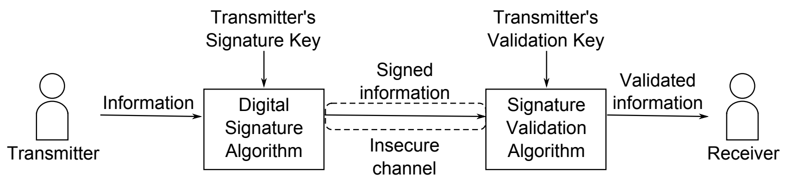

Transmitter's Signature Key
Transmitter's Validation Key
Signed information
Validated information
Information
Digital Signature Algorithm
Signature Validation Algorithm
Transmitter
Insecure channel
Receiver

Figure 1.2: Digital signature scheme.

Data integrity, authentication and non-repudiation are achieved by the use of digital signatures. This cryptographic scheme is depicted in figure 1.2 as a communication system. In this scenario, the transmitter signs the information using a signature key and a signature algorithm. The receiver then validates the signature using a validation key and a validation algorithm. If the signature is correctly verified, the receiver can be sure that the originator of the information is the transmitter (authentication) and so the transmitter cannot deny the information's origin (non-repudiation). Usually, the digital signature is generated using the information itself, if this information is modified the validation algorithm will not succeed, ensuring the truthfulness of the information (data integrity) [3].

The signature key is only known by the transmitter and must be kept secret, such that nobody else is able to sign using the transmitter's key. The validation key must be related to the signature key, such that validation key is easily computed from the signature key but computing the signature key from the validation key is not practical [4]. Notice that for digital signatures, the signature key and the validation key cannot be the same.

The cryptosystems for encryption/decryption and digital signature can be merge as shown in figure 1.3. The cryptosystem depicted in figure 1.3 is able to provide the four services of information security described above. Encryption/decryption modules are in charge of the confidentiality, the digital signature modules provide data integrity, authentication and non-repudiation.

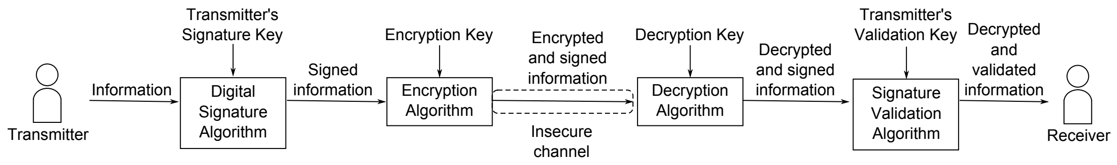

Transmitter's Signature Key
Encryption Key
Encrypted and signed information
Decryption Key
Decrypted and signed information
Transmitter's Validation Key
Decrypted and validated information
Transmitter
Information
Digital Signature Algorithm
Signed information
Encryption Algorithm
Insecure channel
Decryption Algorithm
Signature Validation Algorithm
Receiver

Figure 1.3: Cryptosystem that bring confidentiality, authentication, data integrity and non-repudiation services.

Several cryptographic protocols, schemes, algorithms and functions have been proposed as solution to provide security to information. In figure 1.4, a general view of modern cryptography is shown. At the top of this general view are the information security protocols like TLS/SSL and IPSec. A security protocol is a series of steps, that involve two or more parties, and is designed to accomplish a security task. Protocol steps have to be executed in restricted order. The protocol is directly related to the intended application, different applications have different needs, so several protocols have been proposed to cover different kind of applications. A protocol cannot be ambiguous and must specify every action for every possible situation. In the protocol, all the parameters regarding the security of the system are defined, like the size of the key, how keys are generated, transmitted and stored, among others. All protocols are based on different cryptographic schemes or cryptosystems, which can be of two kinds, symmetric or asymmetric [4].

Viewing the cryptosystem as a communication system as in previous examples, symmetric key cryptosystems use the same key in both transmitter and receiver sides. The encryption and decryption scheme shown in figure 1.1 can be implemented under this scenario, using the same key for encryption and decryption and keeping that key in secret. The digital signature cannot be accomplished by symmetric key cryptosystems [4].

There are two kinds of symmetric cryptosystems: block ciphers and stream ciphers. Block ciphers take as input a fixed size of information (a block) and encrypt that block using the secret key. Stream ciphers work in a continuous way, encrypting one digit of information at a time, usually a digit may be a single bit but not necessary. The aim of a cipher is to produce a “unreadable” output given the input information. Ciphers use substitutions and transpositions of bits or chunks of bits to produce statistic diffusion and confusion along information bits, with more diffusion and confusion more “unreadable” becomes the information.

Block ciphers produce more statistic diffusion and confusion along information bits than stream ciphers, but stream ciphers are considerably faster [4].

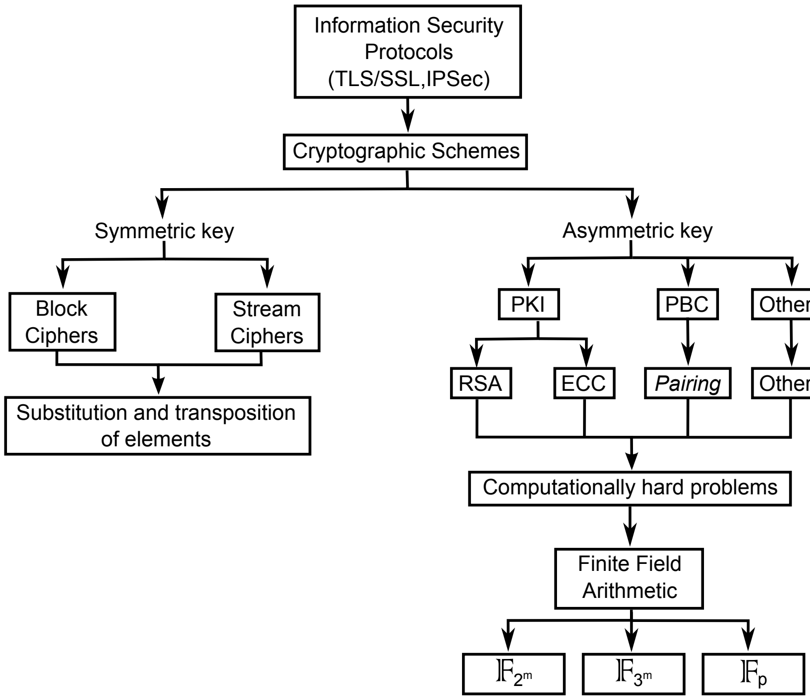


Figure 1.4: General view of modern cryptography.

On the other hand, asymmetric key cryptosystems use two different keys one at the transmitter's side and other at the receiver's side. In these cryptosystems each entity has two related keys: a private key and a public key. The private key is kept in secret while the public key is available to everyone. When encrypting any information, the transmitter uses the receiver's public key. When decrypting, the receiver uses its own private key. In contrast, for digital signatures the transmitter uses its private key for signing and receiver uses the transmitter's public key for signature validation. This type of cryptography is also called *Public Key Cryptography* (PKC) [4] being RSA and Elliptic Curve Cryptography (ECC) the most common PKC cryptosystems.

In asymmetric key cryptosystems, public and private keys are interrelated such that, the public key is easily derived from the private key but it is no feasible to derive the private key form the public one. Computationally hard problems are used in order to achieve this feature [4]. The next mathematical problems are typically used: big numbers factorization [5], discrete logarithm problem [6], and the discrete logarithm problem over elliptic curves [7, 8]. In this sense, RSA is based on the big numbers factorization, while ECC sustains its security on the discrete logarithm problem over elliptic curves.

Although the cryptosystems that work under the PKC scheme could bring the four information security services, there are some drawbacks with this scheme affecting directly the security. The first drawback is that keys have to be generated and distributed in a secure way, because there is the risk that an adversary generates and publishes a public key with the intention of supplanting the identity of an authorized entity. In this scenario, any entity has to be sure of the origin of any public key before validating or encrypting any information. Keys usually have a limited lifetime, even those keys whose lifetime have ended, must be treated properly in order to avoid confusion and missuses.

Additionally, once a pair of private and public keys is defined, those keys have to be used for a long period of time since it is impractical to change keys frequently. In fact, according with the recommendations of the National Institute of Standards and Technology (NIST), the lifetime of a key may extend to 3 years [3]. With that extended lifetime, an adversary has enough time to perform several attacks against the cryptosystem. In this sense, the size of the keys has to be quite large in order to make brute force attacks infeasible, especially for the public key. The NIST recommendation for key's size for 2010 to 2030 is published in [9], for a private key is at least 112 bits length, and for a public key is 2048 bits length when using RSA or 224 bits length when using ECC. Beyond 2030 the recommendation is at least 128 bits for a private key, 3072 bits for an RSA public key, and 256 for an ECC public key.

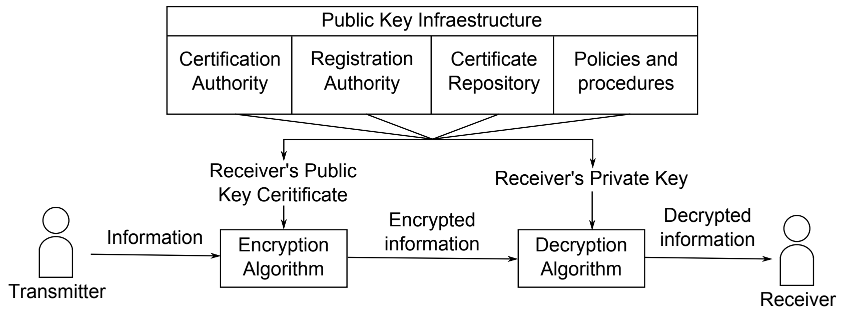


Figure 1.5: Infrastructure of public key cryptography.

Due to the drawbacks mentioned above, PKC requires a complex infrastructure in order to work properly [10]. In figure 1.5, it is illustrated the infrastructure for public key cryptography. Public and private keys are randomly generated and distributed as digital certificates [11]. A digital certificate is an electronic document that binds the identity of an entity with its respective key, as well as the key's lifetime. Digital certificates are generated by a trust authority called certification authority. A registration authority guarantees that the certificate is legitimately received by the corresponding entity, and also it validates the truthfulness of an existing certificate. All certificates are archived in a repository database to confirm the status of a specific certificate. Policies and procedures have to be defined to ensure the correct functionality of this infrastructure. Authorities are approved usually by governments and official authorities.

Due to the growing of communications systems in recent years, the management or distribution and validation of certificates has increased its complexity. Recent schemes have been proposed in order to mitigate the complications found in the certificate management infrastructure like Identity-based cryptography [12].

1.1.2 Identity-based cryptography

The Identity-based cryptography (IBE) was proposed by Shamir in 1985 [1]. The paradigm described by the IBE states that the public key used for encrypting any information, should be some information related to the identity of the receiver for example the e-mail or the phone number. In figure 1.6, it is depicted the IBE paradigm. In this scenario, the transmitter uses the receiver's e-mail as public key. The private key used for decrypting the information is then generated by a trusted third party from the receiver's e-mail. It is important to remark a third party is required for generating the private keys in order to keep the security of the system.

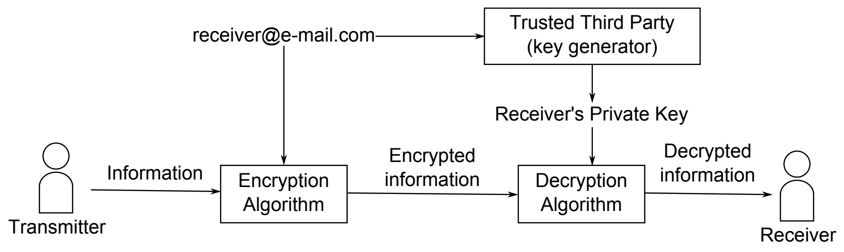


Figure 1.6: The Identity Based Cryptography paradigm.

There are many advantages of using identity information as public key [1]. The identity information does not need to be validated, the receiver's e-mail necessarily belongs to the receivers. The identity information does not need to be distributed by any special agent, the receiver distributes it in a natural way for example in the receiver's web page or business cards. In addition, the identity information can be enhanced with some extra information, for example, some expiration date for the information or some transmission policy. In this way, the key generator also acts as a policy enforcer.

Despite IBE was proposed in the 80's, the mathematical foundation that would ensure the safety of the system was unknown until 2001; when Boneh and Franklin introduce the first practical encryption scheme for IBE [2]. The Boneh and Franklin scheme, denoted as BF-IBE, base its security on the problem known as bilinear Diffie-Hellman problem, which is a reduction of the discrete logarithm problem over elliptic curves. The BF-IBE employs the concept of bilinear pairings over elliptic curves in order to work.

In figure 1.7, it is depicted the IB-IBE. In this scheme, r is Alice's private key, s is the Server's private Key, and P is a public parameter. Notice that Bob's public key Q_{Bob} is derived from Bob's e-mail. For encrypting a message, Alice uses the public keys of the Server and Bob and its own private key to generate a shared key. The shared key is computed using bilinear pairings. The message is then encrypted with the shared key. Alice sends the encrypted message to Bob as well as her public key rP. For decrypting the message, Bob requires his private key sQ_{Bob} which is computed by the Server. Bob is able to compute the shared key using bilinear pairings again and decrypt the message.

Several cryptographic schemes have been proposed under the paradigm of Identity-based cryptography: there are encryption schemes, signature schemes, key agreement schemes, and others. In [13], it is reported a hierarchical scheme for ID-Based encryption. In [14], it is presented a scheme called encryption with keyword search that allows to find a keyword inside an encrypted message. Paterson proposed the first IBE signature scheme in [15]. More recent works for IBE signature schemes are reported in [16, 17]. In works [18–20] there are presented different schemes for key agreement for IBE.

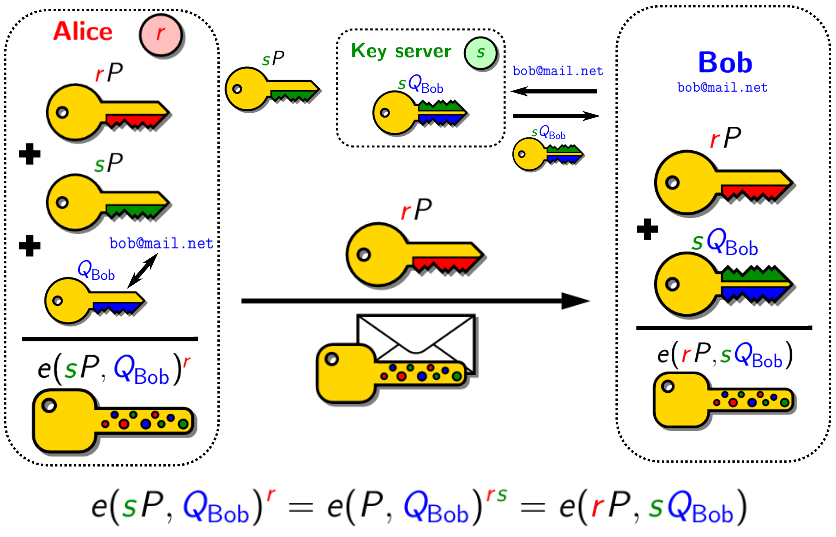


Figure 1.7: The Boneh and Franklin Identity-based encryption scheme.

1.1.3 Pairing-based cryptography

Besides Identity-based cryptography, bilinear pairings have been used for more cryptographic applications. The set of cryptography schemes that makes use of bilinear pairings to work is called Pairing-based cryptography (PBC). In this sense, Identity-based cryptography is a special case of PBC. A bilinear pairing is a function $e : G_1 \times G_1 \to G_T$ that maps two elements of an elliptic curve (G_1) to an element of a finite field (G_T). Chapter 2 covers a detailed explanation about bilinear pairings.

Boneh, *et al.* first proposed a PBC-based signature scheme in [21]. That signature scheme makes use of bilinear pairings to provide the data integrity, authentication and non-repudiation. That signature scheme is named *short signatures* and has the advantage of providing a signature of length 160 bits, which has the same security level as a 320-bits signature in DSA (Digital Signature Algorithm [22]) or 1024-bits length in RSA. The short signature scheme serves as the basis for other signatures schemes; for example, authors in [23] proposed a scheme for blind and ring signature. Boldyreva introduced the threshold signatures, multi-signature and blind signature schemes in [24]. Sakai and Kasahara proposed a signature scheme that reduces the number of pairing computations in [25]. A scheme where a user is able to add his/her signature without affecting other signatures called aggregate signature is reported in [26].

The first key agreement scheme based in bilinear pairing was proposed by Joux in [27]. That scheme called three-party one-round key agreement is based on the Diffie-Hellman problem [6], and it allows three different parties to agree a common shared key using individual secrets while minimizing the amount of sent messages among parties. That work was latter extended to multiple parties by Barua *et al.* in [28].

1.2 Problem description

As it has been shown in previous sections, a present problem with PKI is the complexity of their infrastructure. That infrastructure requires the distribution and management of certificates for all users that need secure communications. With the growing number of users and applications, keeping the PKI's infrastructure has become quite expensive. IBE then offers a solution to simplify the complexity of the PKI's infrastructure.

Nevertheless IBE functionality requires the computation of bilinear pairings over elliptic curves. Computing bilinear pairings requires about 75% of the processing time involving finite field arithmetic operations. Finite field arithmetic follows different rules that conventional arithmetic. Despite general purpose multiprocessors are very powerful, they do not bring support for finite field arithmetic. Therefore finite field arithmetic has been emulated with conventional arithmetic, becoming software implementations not appropriate for high speed applications like real-time communications; for example, a single multiplication over $\mathbb{F}_{2^{1223}}$ requires more than 240 AND operations and more than 480 XOR operations using a 64-bits general purpose multiprocessor, moreover for computing a bilinear pairing in than field are required more than 4,300 multiplications over $\mathbb{F}_{2^{1223}}$. In [29], authors report the fastest software implementation of the bilinear pairing named *eta* over binary fields.

The processing time of software implementations is overcame by specialized implementations in hardware. For example, the works [30–32] reported dedicated architectures for computing the *eta* pairing over binary fields. Other work that focuses on dedicated hardware implementations is [33], which proposes an architecture for the *ate* and optimal pairings over prime fields and ordinary curves. Some implementations are dedicated for ternary fields, like the ones reported in [32] and [34].

Moreover, the pairing algorithms are not yet standardized. There are a growing number of algorithms and improvements. In [35], Silverman explains the Weil and Tate pairings which are the first type of bilinear pairings proposed for cryptographic applications. Barreto, *et al.* introduced some improvements for the Tate pairing in [36] which allow a substantial reduction on the number of operations performed. Later, Barreto, *et al.* proposed the *eta* pairing in [37], which is a special case of the Tate pairing for supersingular curves. The authors of [38], proposed different versions of the *eta* pairing algorithm. On the other hand, the *ate* pairing presented in [39] by Hess, *et al.* generalizes the *eta* pairing for ordinary curves. More recently, in [40] the author introduces the concept of optimal pairings that improves the bilinear pairing computation over ordinary curves. In [41], an algorithm for the Optimal *eta* pairing in supersingular hyper-elliptic curves is reported. Besides the big variety of algorithms, the pairing algorithms have several parameters that can be configured independently of the security level. Those parameters are the elliptic curve, the tower field and the distortion map.

None of the works [30–34] are able to support different version of the pairing algorithm neither to support different parameters. Despite those implementations achieve fast and compact architectures, the flexibility on their functions is limited to just one option among a increasing number of choices. Some other works have tried to solve this problem by providing programmable implementations. The work in [42] reports a programmable architecture for prime fields, meanwhile the architecture reported in [38] is programmable for ternary fields. However, there has not been reported any programmable solution for binary fields. Arithmetic over binary fields is carry-free, therefore, it usually reports smaller and faster implementations compared to prime fields.

1.3 Bilinear pairings computation

So far an introduction to the context of Identity-based cryptography has been presented. In the previous section some works attempting to solve the tackled problem in this thesis have been mentioned. In this section more details of each work is provided. State of the art works have been divided in three categories: software, no-flexible hardware, and flexible hardware implementations.

1.3.1 Software implementations

Regarding software implementations, the work reported in [29] reaches the fastest implementation of pairing algorithms. 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 [43] proposed an alternative software implementation. In that work, authors make use of a Graphic Processor Unit (GPU) in order to accelerate the pairing computation. That work implemented the *eta* 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 computing 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 is specialized software libraries for pairing computations. Specialized software libraries exploit certain features of general purpose 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 [44] is a library specialized in Pairing-based cryptography, which is 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 on optimizing the memory consumption. For a security level of 66 bits, the library is able to compute the *eta* pairing in 5.32 ms.

Software implementations like [29] use very powerful general purpose microprocessors, but even with the latest technology they cannot achieve the performance reached with optimized architectures. According to [43], using GPUs to accelerate the pairing computation does not improve the processing time. [44] presented a library suitable for very constrained environments but it does not reach a high level of security.

1.3.2 Non-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 [30]. The architecture presented in that work is based on a Karatsuba-Ofman multiplier which uses a serial-parallel approach. A trade-off 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 [45] 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 [31] 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 device, 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 *eta* pairing for a 128-bit security level.

Beuchat *et al.* have reported several works improving the implementation of the *eta* pairing computation culminating with [32]. 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 device.

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 [46] and the lazy reduction technique [47] is used for reducing the complexity of the arithmetic operations in prime fields. Additionally an optimization at both architectural and algorithmic 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 7,032 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 requires an area of 4,755 slices and a time of 2.23 ms for a Xilinx Virtex 4.

Even [30–33] 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 best 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.

1.3.3 Flexible hardware implementations

The work reported in [42] 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 desired 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 [48] combined with the Montgomery Ladder technique [49]. 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 for the best case.

In [38] it is proposed a coprocessor for computing the *eta* 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 [41], 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 with a clock frequency of 2.53 GHz.

Finally, [50] reports 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 [50] focused on 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 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 *eta* 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], [42] and [50] 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. A flexible implementation for pairing computation over binary fields has not been reported. Authors in [41] show how a flexible architecture can be used for implementing new algorithms, nevertheless the processing time can be improved.

1.4 Hypothesis

If the arithmetic operators for computing the binary field arithmetic, the instruction set and the instruction format are optimized, a programmable solution for computing bilinear pairings over binary fields can be achieved being at least 75% faster than software implementations and as much the same size of custom architecture for computing bilinear pairings. The proposed solution, a programmable cryptoprocessor, should be able to support different versions of the pairing algorithm and different parameters such as the elliptic curve, the tower field and the distortion map.

1.5 Thesis objectives

1.5.1 General objective

The general objective of this research work is to design a flexible hardware architecture for Pairing-based cryptography with a minimum security level of 128 bits, through an efficient design that parallelizes the finite field arithmetic involved.

1.5.2 Specific objectives

The specific objectives defined in order to accomplish the goals of this thesis are:

1. Determine the requirements for a hardware architecture able to compute bilinear pairings with different algorithms and parameters.

2. Define a strategy for the hardware architecture that allows flexibility in the computation of bilinear pairings.

3. Propose a flexible hardware architecture for computing bilinear pairings with different algorithms and parameter, the proposed architecture must be competitive against similar architectures with similar security levels.

4. Evaluate the proposed hardware architecture under applications of Pairing-based cryptography.

1.6 Thesis outline

This thesis document is organized as follows: chapter 2 presents the theory related to the binary field arithmetic as well as the mathematical background required to understand the computations of bilinear pairings over elliptic curves. Also that chapter introduces some cryptographic schemes for Pairing-based cryptography. Chapter 3 describes the proposed flexible architecture for pairing computation, standing out the requirements considered for the design. The strategy followed for validating the functionality of the proposed architecture is described in chapter 4. Also this chapter presents the synthesis results and a comparison against related works. Finally, chapter 5 summarizes this work and the achieved contributions. The conclusions and future work are finally presented at the end of this chapter.

Chapter 2

Pairing-based cryptography

Identity-based cryptography proposed by Shamir in 1985 [1] makes use of bilinear pairings over elliptic curves as mathematical framework. Further, the new kind of cryptography named Pairing-based cryptography requires the computation of bilinear pairings. For cryptographic applications, elliptic curves are defined over finite fields. In this chapter the theoretical background for computing bilinear pairings over elliptic curves is discussed. First an introduction to finite fields and elliptic curves is presented. Then, a detailed definition of bilinear pairings, two types of them and the computation strategy are described. Finally, some basic schemes for Pairing-based cryptography are presented.

2.1 Binary finite fields and elliptic curves

2.1.1 Binary finite fields and arithmetic over $\mathbb{F}_{2^m}$

A finite field, represented as $\mathbb{F}_q$ 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's extension [51]. For cryptographic applications p is typically 2, 3 or a prime number [52]. The number of elements on the finite field is given by p^m. In polynomial basis, the set of elements in $\mathbb{F}_q$ is all the polynomials of degree at most $m-1$, where each polynomial is an element in $\mathbb{F}_q$ and each coefficient of the polynomial can only take its value from the set $\{0, ..., p-1\}$. Additionally, $\mathbb{F}_q$ 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 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 by a m-length bit vector.

Lets $\mathbb{F}_{2^m}$ 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* $A \oplus B$, is defined as a polynomial addition simply performed by a bitwise XOR gate among each coefficient with no carry propagation among coefficients. Due to the operation is bitwise, the resulting bit vector is at most leng m, which represents a polynomial with degree less than m. That polynomial belongs to $\mathbb{F}_{2^m}$.

A *multiplication* $A \otimes B$ can be seen as a two steps operation, see equation 2.1. First, a polynomial multiplication $C' = A \cdot B$, where C' is a polynomial of degree $2m - 2$, which do not belongs to $\mathbb{F}_{2^m}$. Therefore a modular reduction $C = C' \mod f(x)$ is performed in a second step in order to compute the congruent element with $A \otimes B$ which belongs to $\mathbb{F}_{2^m}$ [52].

$$\begin{aligned} C = A \cdot B \mod f(x) &= \sum_{i=0}^{m-1} a_i x^i \cdot \sum_{j=0}^{m-1} b_j x^j \mod f(x) \\ &= \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j x^{i+j} \mod f(x) \end{aligned} \tag{2.1}$$

Multiplication is a very expensive operation, so much work has been done to reduce its computational cost. There are several algorithms to compute the field multiplication, among them and widely known is the classical algorithm also named Schoolbook method consisting of a shift-and-add scheme [52]. Most of the proposed field multiplication algorithms are based on this method whose complexity is $O(n^2)$. In 1962, a multiplication algorithm was published by Karatsuba and Ofman [53] with $O(n^{\log_2 3})$ complexity. The KOA algorithm computes the first step of a field multiplication by using the divide and conquer technique. The polynomial multiplication is computed recursively using three field multiplications with low order operands. For simplicity, lets consider that $m = 2^t$ for $t \geq 0$, so that $m/2$ is always a power of 2.

KOA splits the multiplier and multiplicand as shown in equation 2.2.

$$A = (\underbrace{\alpha_{m-1}, \alpha_{m-2}, \cdots, \alpha_{m/2}}_{A^H}, \underbrace{\alpha_{m/2\,-1}, \cdots, \alpha_0}_{A^L}) \tag{2.2}$$

Thus, equation 2.3 holds:

$$A = A^H x^{m/2} + A^L \tag{2.3}$$

where A^H, A^L are $m/2$-order polynomials.

The operand B is processed in the same way as A, and $C' = A \cdot B$ is calculated as:

$$C' = (A^H x^{m/2} + A^L) \cdot (B^H x^{m/2} + B^L) = z_2 x^m + z_1 x^{m/2} + z_0 \tag{2.4}$$

where, $z_2 = A^H \cdot B^H$, $z_1 = A^H \cdot B^L + A^L \cdot B^H$ and $z_0 = A^L \cdot B^L$.

At this point, $A \cdot B$ requires four multiplications with operands that are half the size the initial ones. KOA can be used recursively to compute these new multiplications and reducing the number of multiplications to three at the cost of some more additions by redefining z_1, as it is shown in equation 2.5. In $\mathbb{F}_{2^m}$, additions and subtractions are the same operation, thus redefining z_1 has no substantial cost. The recursive Karatsuba-Ofman method for multiplying two polynomials $A, B \in \mathbb{F}_{2^m}$ is shown in algorithm 1.

$$z_1 = (A^H + A^L) \cdot (B^H + B^L) - z_2 - z_0 \tag{2.5}$$

The KOA algorithm receives as input the multiplier and multiplicand as well as their bit-length (n). In the first call $n = m$. At each recursive call, operands are divided resulting in $n/2$-bit vectors. The recursive KOA finishes when $n = 1$, returning as a result the bitwise AND of A and B. Steps 7-8 in algorithm 1 perform a recursive call to KOA and the resulting polynomials z_0, z_1 and z_2 are $(n-1)$-bit vectors. In step 9, the final multiplication $C'(x) = A(x) \cdot B(x)$ is calculated, resulting in a $(2n-1)$-order polynomial. When all recursive calls are finished, the final result is a $(2m-1)$-order polynomial. A graphical representation of $C'(x)$ operation is depicted in Fig. 2.

Up to this point, it is assumed that $m = 2^k$, but in many applications, such as cryptography, m is not a power of 2. One strategy is padding with 0s the bit vector representation of the input operands until reaching a power of 2 length, but with this strategy many hardware is unused. Thus, a modification to KOA called Binary Karatsuba Multiplier (BKM) was proposed in [52].

Algorithm 1 KOA[n,A,B]: Recursive Karatsuba-Ofman algorithm.

```
Require: $n = 2^t, t \geq 0, n \leq m, A, B$ being $n$-bit vectors
Ensure: $C' = A \cdot B$
 1: if $n = 1$ then
 2:     return $A \odot B$
 3: end if
 4: $A \leftarrow A^H x^{n/2} + A^L$
 5: $B \leftarrow B^H x^{n/2} + B^L$
 6: $z_2 \leftarrow$KOA[$n/2$ , $A^H, B^H$]
 7: $z_0 \leftarrow$KOA[$n/2$ , $A^L, B^L$]
 8: $z_1 \leftarrow$KOA[$n/2$ , $(A^L + A^H), (B^L + B^H)$] $+ z_2 + z_0$
 9: $C' \leftarrow z_2 x^n + z_1 x^{n/2} + z_0$
10: return $C'$
```

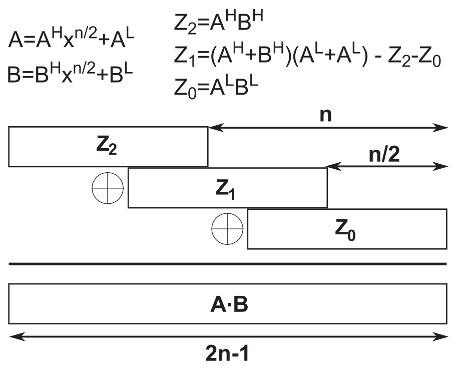


Figure 2.1: C' computation at step 9 of algortihtm 1.

Modular reduction. Algorithm 1 computes $C' = z_2 x^m + z_1 x^{m/2} + z_0$, where C' is a $(2m-1)$-bit vector that does not belong to $\mathbb{F}_{2^m}$ and needs to be reduced mod $f(x)$. For general irreducible polynomials $f(x)$, specialized reduction methods must be applied, such as the Barret reduction method [54] or the Montgomery method [55].

For special $f(x)$ classes, such as trinomials and pentanomials, the reduction step of KOA algorithm can be performed using a matrix of XOR gates [52]. The reduction technique is based on the fact that if $f(x) = x^m + g(x)$, where $g(x)$ is a low order ($< m$) polynomial, the equivalence $x^m \equiv g(x) \mod f(x)$ is sustained. Therefore, a trinomial $f(x)$ with form $f(x) = x^m + x^a + 1$ allows to express the polynomial C' in the following way:

$$\begin{aligned} C' &= \sum_{i=0}^{2m-2} c_i x^i = \sum_{i=0}^{m-1} c_i x^i + \sum_{i=m}^{2m-2} c_i (x^{a+i-m} + x^{i-m}) \\ &= \underbrace{\sum_{i=0}^{m-1} c_i x^i}_{(1)} + \underbrace{\sum_{i=0}^{m-1-a} c_{i+m} x^{a+i}}_{(2)} + \underbrace{\sum_{i=0}^{a-1} c_{2m-a+i} x^{a+i}}_{(3)} + \underbrace{\sum_{i=0}^{a-1} c_{2m-a+i} x^{i}}_{(4)} \\ &\quad + \underbrace{\sum_{i=0}^{m-1} c_{m+i} x^i}_{(5)} \end{aligned} \tag{2.6}$$

The last expression in equation 2.6 states that C can be formulated as a m-bit vector that results from adding five terms obtained from C', achieving the desired reduction $\mod f(x)$. Graphically, this reduction is shown in Fig. 2.2.

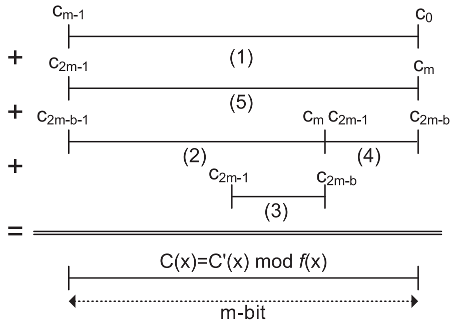


Figure 2.2: Computation of $C = C' \mod f(x)$, where $f(x)$ is a trinomial.

Squaring, $\bullet^2$ is a special case of multiplication when $A = B$ that also requires modular reduction [52].

$$\begin{aligned} C{=}A^2{=}A \otimes A \mod f(x) &= \sum_{i=0}^{m-1} a_i x^i \otimes \sum_{i=0}^{m-1} a_i x^i \mod f(x) \\ &= \sum_{i=0}^{m-1} a_i x^{2i} \mod f(x) \end{aligned} \tag{2.7}$$

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 [52].

An interesting identity for any two elements $A, B \in \mathbb{F}_{2^m}$ is depicted in equation 2.8, which states that squaring is distributive over addition. This identity will be useful for the proposed architecture design for pairing computation. It can be demonstrated that this identity also holds for squaring root. More properties about finite fields and other algebraic structures can be found in [51].

$$\begin{aligned} A^2 + B^2 &= \sum_{i=0}^{m-1} a_i x^{2i} \mod f(x) + \sum_{i=0}^{m-1} b_i x^{2i} \mod f(x) \\ &= \sum_{i=0}^{m-1} (a_i + b_i) x^{2i} \mod f(x) = (A+B)^2 \end{aligned} \tag{2.8}$$

Multiplicative inverse $(\bullet)^{-1}$ of an element A is defined as the unique element A^{-1}, such that $1 = A \otimes A^{-1} \mod f(x)$ holds. There exist several algorithms to compute A^{-1} given A. 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 $\mathbb{F}_{2^m}$ [52].

Extended finite fields

A field K_2 containing a field K_1 is called *extension field* of K_1, for example $\mathbb{F}_{2^m}$ is an extension field of $\mathbb{F}_2$. An irreducible polynomial $g(x)$ of degree k is necessary to define $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$ [56]. A sequence of field extensions is called *tower field* [56].

Here is an example of a tower field from $\mathbb{F}_q$ to $\mathbb{F}_{q^4}$: first it is defined an extension from $\mathbb{F}_q$ to $\mathbb{F}_{q^2}$ using the polynomial $g(u) = (u^2 + u + 1)$, this extension is denoted by the equation $\mathbb{F}_{q^2} = \mathbb{F}_q[u]/(u^2+u+1)$; next it is defined an extension from $\mathbb{F}_{q^2}$ to $\mathbb{F}_{q^4}$ using the polynomial $g(v) = (v^2 + v + u)$, denoted by $\mathbb{F}_{q^4} = \mathbb{F}_{q^2}[v]/(v^2+v+u)$.

A polynomial basis to represent elements in $\mathbb{F}_{q^k}$ with elements of $\mathbb{F}_q$ can be constructed from a tower field [56]. Using the previous example, 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 each coefficient $g_i \in \mathbb{F}_q$.

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 requiring only four additions over $\mathbb{F}_q$.

A multiplication also follows the polynomial multiplication rules. Using the Karatsuba-Ofman approach, it requires nine additions and several additions over $\mathbb{F}_q$. The exact amount of additions depends on the tower field. For the tower field defined above see algorithm 2 [57]. Notice that during the multiplication, it may occur cases like $g_1u \cdot g_1u = g_1^2u^2$. Due to the irreducible polynomials defining the tower field, it can be deduced that $u^2 = u + 1$ and $v^2 = u + v$. Therefore, $g_1u \cdot g_1u = g_1^2u^2 = g_1^2 + g_1^2u$ when using the tower field defined above.

Algorithm 2 Multiplication over $\mathbb{F}_{q^k}$

```
Require: $G = g_0 + g_1u + g_2v + g_3uv \in \mathbb{F}_{q^k}$ and $H = h_0 + h_1u + h_2v + h_3uv \in \mathbb{F}_{q^k}$
Ensure: $W = G \cdot H$
1: $a_0 \leftarrow g_0 + g_1$; $a_1 \leftarrow h_0 + h_1$; $a_2 \leftarrow g_0 + g_2$;
2: $a_3 \leftarrow h_0 + h_2$; $a_4 \leftarrow g_1 + g_3$; $a_5 \leftarrow h_1 + h_3$;
3: $a_5 \leftarrow g_2 + g_3$; $a_7 \leftarrow h_2 + h_3$; $a_8 \leftarrow a_0 + a_6$; $a_9 \leftarrow a_1 + a_7$;
4: $m_0 \leftarrow g_0 \cdot h_0$; $m_1 \leftarrow g_1 \cdot h_1$; $m_2 \leftarrow g_2 \cdot h_2$; $m_3 \leftarrow g_3 \cdot h_3$;
5: $m_4 \leftarrow a_0 \cdot a_1$; $m_5 \leftarrow a_2 \cdot a_3$; $m_6 \leftarrow a_4 \cdot a_5$; $m_7 \leftarrow a_6 \cdot a_7$; $m_8 \leftarrow a_8 \cdot a_9$;
6: $a_{10} \leftarrow m_0 + m_1$; $a_{11} \leftarrow m_0 + m_4$;
7: $w_0 \leftarrow a_{10} + m_2 + m_7$; $w_1 \leftarrow a_{11} + m_3 + m_7$;
8: $w_2 \leftarrow a_{10} + m_5 + m_6$; $w_3 \leftarrow a_{11} + m_5 + m_8$;
9: return $W = w_0 + w_1u + w_2v + w_3uv$;
```

However, there are cases where some coefficients are either 0 or 1, then multiplication is simplified. For example, consider $W = (g_0+g_1u+v)\cdot(h_0+h_1u+h_2v+h_3uv)$ where $g_2 = 1$ and $g_3 = 0$. This product only requires six multiplications and fourteen additions [58].

Squaring in $\mathbb{F}_{q^4}$ only requires four additions and four squaring over $\mathbb{F}_q$, considering the previous tower field [58]:

$$\begin{aligned} G^2 &= (g_0 + g_1u + g_2v + g_3uv)^2 \\ &= (g_0 + g_1 + g_3)^2 + (g_1 + g_2)^2u + (g_2 + g_3)^2v + g_3^2uv \end{aligned} \tag{2.9}$$

Raising an element to the q-th power is an operation easily computed using the tower field. For tower field defined in [37] this computation only requires five additions over $\mathbb{F}_q$:

$$\begin{aligned} G^q =& (g_0 + g_1u + g_2v + g_3uv)^q \\ =& (g_0 + g_1 + g_2) + (g_1 + g_2 + g_3)u + (g_2 + g_3)v + g_3uv \end{aligned} \tag{2.10}$$

2.1.2 Elliptic curves

Elliptic curves were first introduced in cryptosystems by Koblitz and Miller over 1985 [7, 8]. Since then, an entire area in cryptography has been developed called *Elliptic Curve Cryptography* (ECC). The advantage of using ECC over other kind of cryptosystems is that an equivalent level of security can be reached using shorter keys, because the fact that the *Discrete Logarithm Problem* is harder to solve when it is defined over elliptic curves [4]. Computing a bilinear pairing involves elliptic curve operations. This section presents a brief introduction of elliptic curves in cryptography.

An elliptic curve, denoted by E, is defined as a set of points (x, y) that satisfy the Weierstrass equation denoted by equation 2.11.

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.11}$$

Elliptic curves can be defined over real numbers but for cryptographic applications are defined over finite fields. In Figure 2.3, it is shown an example of an elliptic curve over real numbers 2.3(a) and over finite field of size 23 2.3(b). Notice that the same equation is used in both examples.

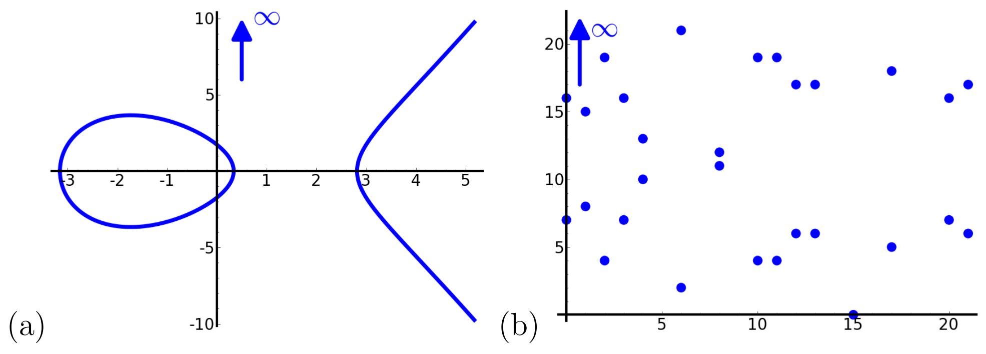


Figure 2.3: Elliptic curve defined by $y^2 = x^3 - 9x + 3$. 2.3(a) over real numbers. 2.3(b) over a finite field of size 23.

The number of points in an elliptic curve is very important because that number is close related to the security of ECC and Pairing-based cryptosystems. That number is given by $\#E(\mathbb{F}_q) = q + 1 - t$ where t is called the *trace of Frobenius* and due to the Hasse's theorem it is bounded by $|t| < 2\sqrt{q}$ [35]. When the field's characteristic p divides t, denoted by $p|t$, the curve is called *supersingular*, other way it is called *ordinary*. There are several types of bilinear pairings reported, some bilinear pairings are defined for supersingular curves while other for ordinary curves. so far in the literature, only supersingular curves are known when the curve is defined over binary finite fields [12].

As mention at the beginning of this section, bilinear pairing computation involves operation among elements of an elliptic curve. The basic operation for points in an elliptic curve is the addition, which is described by the *chord-and-tangent* rule. A geometric representation of the chord-and-tangent rule for real numbers is depicted in figure 2.4. For adding two different points $P, Q \in E$ over real numbers, figure 2.4(a), a line is drawn through P and Q which intersects the curve in a third point denominated as $-(P + Q)$; then a second line is drawn through the point $-(P + Q)$ and the point ∞, intersecting again the curve in the point which is the addition of $P + Q$. In figure 2.4(b), it is shown the geometric representation of the chord-and-tangent rule for finite fields. Notice that when the curve is defined over finite fields, the line through P and Q wraps the plot. The identity element in this additive group is named the point at infinity, denoted by ∞ [35]. The set of points in $E(\mathbb{F}_q)$ and a rule for adding two elements in $E(\mathbb{F}_q)$ called *chord-and-tangent* construct an algebraic structure called *cyclic additive group*.

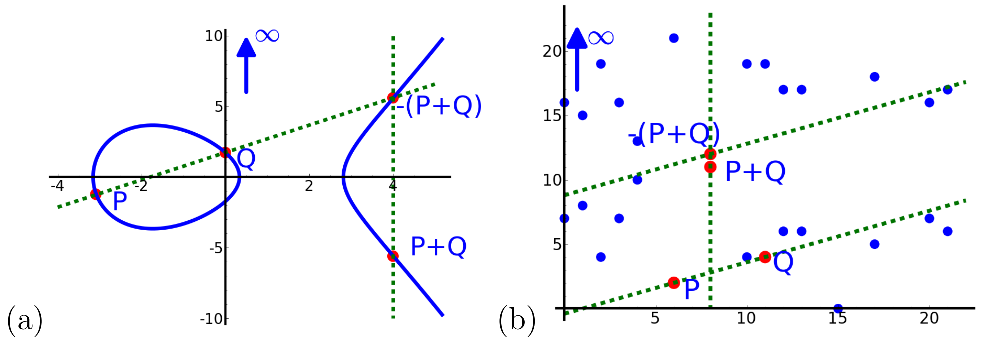


Figure 2.4: Rule for adding $P + Q$. 2.4(a) over real numbers. 2.4(b) over a finite field.

Other operation with points of an elliptic curve is the *scalar multiplication*.

The scalar multiplication is denoted as rP and defined by equation 2.12, where $r \in \mathbb{N}$, and $P \in E(\mathbb{F}_q)$, defined as r additions of the point P with itself.

$$rP = \underbrace{P + P + P + ... + P}_{r} \tag{2.12}$$

Other important concepts about elliptic curves, which are necessary for the understanding of bilinear pairings are order of a point, torsion subgroup, and embedding degree. 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 $k = 4$ [36].

The discrete logarithm problem and elliptic curves

As stated in section 1.1.1, computationally hard problems are used in order to bring support to several cryptosystems. These problems have to be easy to compute in one way but hard to solve in the inverse way. For example, given a natural number r and a point P over a elliptic curve $E(\mathbb{F}_q)$, computing $Q = rP$ is relatively easy, but given the points $Q, P \in E(\mathbb{F}_q)$ computing the natural number r such that $Q = rP$ is computationally infeasible. This problem is called the *Elliptic Curve Discrete Logarithm Problem* (ECDLP) [7, 8] and it is the base of an entire field in cryptography called Elliptic Curve Cryptography.

The best known algorithm for solving the ECDLP is the *Pollard's* ρ algorithm which has a exponential complexity [59]. However, the discrete logarithm problem could be easier to solve when it is defined over other kind of groups for example, when it is defined over multiplicative groups, the known index-calculus algorithms solve the problem in a subexponential time [60–62]. In the early 1990s, Menezes, Okamoto and Vanstone, proposed a method to reduce the ECDLP to the easier problem over finite fields using bilinear pairings [63]. Nevertheless in the early 2000s, constructive scheme based on bilinear pairings were proposed [2, 27, 64], showing that bilinear pairings allow the efficient implementation of a kind of cryptography called Pairing-based cryptography.

2.2 Bilinear pairings

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 bilinear pairing is a map:

$$\hat{e}: G_1 \times G_1 \to G_T \tag{2.13}$$

that satisfies the following conditions $\forall P, R, Q \in G_1$ [35]:

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

The following properties of bilinear pairings can be easily verified [35]:

$$\begin{aligned} & \hat{e}(P,\infty)=1 \quad \text{and} \quad \hat{e}(\infty,P)=1 \\ & \hat{e}(P,-Q)=\hat{e}(-P,Q)=\hat{e}(P,Q)^{-1} \\ & \hat{e}(sP,rQ)=\hat{e}(P,Q)^{sr} \quad \forall r,s \in \mathbb{N} \\ & \hat{e}(P,Q)=\hat{e}(Q,P) \end{aligned}$$

Under certain circumstances, bilinear pairings are defined over two different additive groups $\hat{e}{:}G_1 \times G_2 \to G_T$. This kind of pairing is called *asymmetric* pairing, while the former is called *symmetric*. Additional to the previous conditions, it is required that G_1 and G_2 be cyclic groups [35].

In the following subsection are introduced two of the most common bilinear pairings reported in the literature.

2.2.1 Tate pairing

The Tate pairing is an asymmetric bilinear pairing over elliptic curves defined in equation 2.14, where $E(\mathbb{F}_q)[r]$ and $E(\mathbb{F}_{q^k})[r]$ are additive groups formed by the set of point *r-Torsion* on $E(\mathbb{F}_q)$ and $E(\mathbb{F}_{q^k})$ respectively, and $\mathbb{F}_{q^k}^*$ is an multiplicative group formed by the elements in the extended field $\mathbb{F}_{q^k}$ except for the element 0.

$$\tau\colon E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k})[r] \to \mathbb{F}_{q^k}^* \tag{2.14}$$

As is shown in [35], the curves $E(\mathbb{F}_q)[r]$ and $E(\mathbb{F}_{q^k})[r]$ are isomorphic, that is both have the same algebraical structure; so the Tate pairing can be redefined as a symmetric pairing τ: $E(\mathbb{F}_q)[r] \times E(\mathbb{F}_q)[r] \rightarrow \mathbb{F}^*_{q^k}$.

A formula for computing the Tate pairing is depicted in equation 2.15, 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. The computation of equation 2.15 is divided in two stages: first $f_P(D_Q)$ is computed by the Miller's algorithm [65], second an exponentiation to the $(q^k-1)/r$-th power, called *final exponentiation* is required [36].

$$\tau(P,Q) = f_P(D_Q)^{(q^k-1)/r} \tag{2.15}$$

Miller's algorithm

A *divisor* is a formal sum of points on the curve $\mathfrak{D} = \sum_{P \in E} n_P(P)$, where n_P are integers which only a finite number are nonzero and (P) is the notation for a formal symbol of each point $P \in E(\mathbb{F}_q)$. The degree of a divisor is the sum of all integers n_P. Let $f : E(\mathbb{F}_q) \rightarrow \mathbb{F}_{q^k}$ be a function on the curve. The divisor of a function f is $div(f) \equiv \sum_{P \in E} ord_P(f)(P)$, where $ord_P(f)$ is the order of the zero or the pole of f at the point P. A divisor is called principal if $\mathfrak{D} = div(f)$ for some function f. Two divisors $\mathfrak{C}$ and $\mathfrak{D}$ are equivalent if their difference $\mathfrak{C} - \mathfrak{D}$ is a principal divisor. Let $P \in E(\mathbb{F}_q)[r]$ where r is coprime to q, an let $\mathfrak{D}_P$ be a divisor equivalent to $(P) - (\infty)$; under these circumstances the divisor $r\mathfrak{D}_P$ is principal, and hence there is a function f_P such that $div(f_P) = r\mathfrak{D}_P = r(P) - r(\infty)$. For more details in the definition of divisor the reader could refer to [35].

In order to f_P satisfy the necessary conditions of a bilinear pairing, the divisor of f_P must be, $div(f_P) = r(P) - r(\infty)$ where r is the order of the point P [65]. When evaluating the pairing, the function f_P must be evaluated on the point Q. The Miller's algorithm is a numeric method to construct the function f_P using the double-and-add rule for adding two points on the curve. In [35, 65] a detailed derivation of the Miller'a algorithm is presented. In algorithm 3, it is depicted the general form of Miller's algorithm.

Several improvements have been proposed to the original Miller's algorithm, the most significant are the elimination of denominators in lines 7 and 12 of algorithm 3 [37], computing the function f_P over points instead of divisor which actually is the algorithm 3 [37], and reducing the number of loops [66].

Algorithm 3 General form of Miller's algorithm

```
Require: $r$ and $P, Q \in E(\mathbb{F}_{2^m})[r]$.
Ensure: $f_P(Q)$ where $div(f_P) = r(P) - r(\infty)$.
 1: Let the binary representation of $r$ be $(r_t, ..., r_1, r_0)$
 2: Select a point $R \in E(\mathbb{F}_{2^m})[r] \setminus \{\infty, P, -Q, P-Q\}$
 3: Set $T = P$ and $f = 1$
 4: for $i = t-1$ to 0 do
 5:     Let $l$ be the tangent line through $T$
 6:     Let $v$ be the vertical line through $2T$
 7:     Set $f = f^2 \cdot \frac{l(Q+R)}{v(Q+R)} \cdot \frac{v(R)}{l(R)}$
 8:     Set $T = 2 \cdot T$
 9:     if $r_i = 1$ then
10:         Let $l$ be the line through $T$ and $P$
11:         Let $v$ be the vertical line through $T + P$
12:         Set $f = f \cdot \frac{l(Q+R)}{v(Q+R)} \cdot \frac{v(R)}{l(R)}$
13:         Set $T = T + P$
14:     end if
15: end for
16: return $f$
```

Final exponentiation

In order the pairing be a unique element of $\mathbb{F}_{q^k}$, the result of the Miller's algorithm has to be raised to the $(q^k - 1)/r$-th power [37]. This step in the pairing computation is called *final exponentiation*. Computing the final exponentiation require several operations over the extend field $\mathbb{F}_{q^k}$. Several improvements can be performed in order to reduce its cost, two examples can be found in references [36, 57].

2.2.2 *Eta* pairing

Several works have been proposed to optimize the Tate pairing computation at an algorithmic level [37, 39, 66]. An especial case of the original Tate pairing for supersingular curves is the *eta* pairing (η_T) presented in [37]. The η_T pairing reduces by the half the FOR-loop of the Mille's algorithm, being this pairing the most popular algorithm for bilinear pairings over binary fields. The η_T pairing requires a *distortion map* $\psi : E(\mathbb{F}_q) \rightarrow E(\mathbb{F}_q^k)$ for the point Q in order to $E(\mathbb{F}_q^k)$ be a cyclic group.

In algorithm 4, it is depicted the algorithm for computing the η_T over $\mathbb{F}_{2^m}$.

Several parameters depend on the elliptic curve and finite field [37]. Lets 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 [37], 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 $\delta = 0$. An 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 4, 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 [36, 57, 58].

Algorithm 4 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:     $s \leftarrow x_1 + \gamma$, $x_1 \leftarrow \sqrt{x_1}$,$y_1 \leftarrow \sqrt{y_1}$
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 $F^{(2^{2m}-1)\cdot(2^m+1-\epsilon 2^{(m+1)/2})}$
```

Nevertheless, an alternative algorithm for computing the η_T using different parameters for basis is presented in [67]. Besides, a version of the algorithm for computing η_T where no square roots are required is presented in [57]. The works [37, 57, 67] are able to compute bilinear pairings in very different ways. The necessity of a flexible solution able to manage this variety of parameters and algorithms emerge because development of algorithms and improvements are still in process and the lack of a standard for computing bilinear pairings in cryptographic applications.

2.3 Cryptographic schemes

So far in this chapter, it has been discussed a theoretical background about binary finite fields, elliptic curves and bilinear pairings. The application of these concepts in cryptography is presented in this section.

In the following cryptographic schemes, it is assumed the existence of a bilinear pairing as defined in section 2.2. In all cases, P is an element of the group G_1. $\mathbb{Z}_r$ denotes the set of integer numbers $\mod r$, where r is the order of G_1, and $\mathbb{Z}_r^* = \mathbb{Z}_r - \{0\}$.

2.3.1 Encryption

An Identity-based encryption scheme is an asymmetric key scheme (see figure 1.1), where the encryption key is derived from the identity of the receiver, for example the receiver's e-mail; and the decryption key is computed by a trusted third party or Private Key Generator (PKG) using also the receiver's e-mail.

Boneh and Franklin presented the first encryption scheme for IBE in [2] denoted by BF-IBE. The BF-IBE scheme requires an asymmetric pairing and it consists in four steps: setup, extract, encrypt and decrypt. The security of BF-IBE is based in the bilinear version of the Diffie-Hellman problem [68].

Setup: This step generates system parameters. The PKG chooses a random element $s \in \mathbb{Z}_r^*$, and sets $P_{pub} = sP$. Two cryptographic 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 the PKG is P_{pub}, private key of the PKG is s.

Extract: This step computes the receiver's keys, given a 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 the PKG as $S_{ID} = sQ_{ID}$.

Encrypt: This step is the encryption algorithm. The transmitter chooses a random integer t. Given a message $M \in \{0,1\}^n$, the encrypted message is the tuple $C = \langle tP, M \oplus H_2(g_{ID}^t)\rangle$, where $g_{ID} = e(Q_{ID}, P_{pub})$.

Decrypt: Given a encrypted message $C = \langle U, V\rangle$, the original message M is computed by $M = V \oplus H_2(e(S_{ID}, U))$.

The BF-IBE scheme holds due to the equation 2.16.

$$e(Q_{ID}, sP)^t = e(Q_{ID}, P)^{ts} = e(sQ_{ID}, tP) \tag{2.16}$$

Sakai and Kasahara proposed a signature scheme that reduces the number of pairing computations in [25]. The encryption scheme proposed in that work, additionally integrates the signature into the encryption scheme. In [13], it is reported a hierarchical scheme for ID-Based encryption. In that work, the PKG is not a single entity but a hierarchy of entities, such that the workload of private key generation is distributed. In [14], it is presented a scheme called encryption with keyword search. That scheme allows to find a keyword inside a encrypted message without disclosing the content of the encrypted message.

2.3.2 Signature

Boneh, Lynn and Shacham first proposed a signature scheme based on bilinear pairing called short signatures [21]. RSA uses a signature length of 1024 bits, while DSA uses a signature length of 320 bit. Short signatures uses a signature length of 160 bits. The signature scheme proposed by Boneh *et al.* consist in three steps: KeyGen, Sing, Validation.

KeyGen: Choose and publish a hash function $H : \{0,1\}^* \to G_1$. Signer chooses a secret key $x \in \mathbb{Z}_r^*$, and publishes 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 due to the equation 2.17.

$$e(P, xH(M)) = e(P, H(M))^x = e(xP, H(M)) \tag{2.17}$$

The short signature scheme proposed by Boneh *et al.*, serves as basis for several signature schemes. In [23] authors proposed a scheme for blind signature and ring signature. Blind signature is a scheme where a user cannot obtain more than one valid signature after one interaction with the signer. Blind signatures are typically employed in privacy-related protocols where the signer and message author are different parties, for example, in electronic voting and electronic payment systems. Ring signature refers a kind of signature where the transmitter belongs to a set of users (ring), each one with a pair of private and public keys. The ring signature verifies that the message is signed by a member of the ring but it cannot be disclosed who exact member performed the signature. Boldyreva introduced the multisignature and blind signature schemes in [24]. Multisignature is a scheme where a group of users can jointly sign the same message, such that a verifier is able to verify that each member of the group have participated in the signing process. In this scheme, the signature is rejected if not all members of the group participated in the signing process. A scheme for aggregate signature is reported in [26]. An aggregate signature is a scheme where any user can add its own signature to a message, the receiver is able to verify the presence of a specific signature over a particular message.

Paterson proposed a signature scheme for IBE in [15]. The Paterson's signature scheme requires three hash functions $H_1 : \{0,1\}^* \to G_1$, $H_2 : \{0,1\} \to \mathbb{Z}_r$, and $H_3 : G_1 \to \mathbb{Z}_r$. Given an identifier ID, the public key for signature verification is computed as $Q_{ID} = H_1(ID)$, the private key for signature generation is $D_{ID} = sQ_{ID}$, where $s \in \mathbb{Z}_r$ is the private key of the PKG. The public key of the PKG is $P_{pub} = sP$.

To sing a message M, the transmitter chooses a random $k \in Z_r$ and computes the signature as the pair $(R, S) \in G_1 \times G_1$, where $R = kP$ and $S = k^{-1}(H_2(M)P + H_3(R)D_{ID})$.

To verify a signature (U, V) on a message M, the receiver computes $e(U, V)$ and compares the result with the value $e(P, P)^{H_2(M)} \cdot e(P_{pub}, Q_{ID})^{H_3(U)}$.

The signature will be verified due to the equation 2.18.

$$\begin{aligned} e(R, S) &= e(kP, k^{-1}(H_2(M)P + H_3(R)D_{ID})) \\ &= e(P, (H_2(M)P + H_3(R)D_{ID}))^{k \cdot k^{-1}} \\ &= e(P, H_2(M)P + H_3(R)D_{ID}) \\ &= e(P, H_2(M)P) \cdot e(P, H_3(R)D_{ID}) \\ &= e(P, P)^{H_2(M)} \cdot e(P_{pub}, Q_{ID})^{H_3(R)} \end{aligned} \tag{2.18}$$

More recent works for IBE signature schemes are reported in [16, 17].

2.3.3 Key agreement

A key agreement scheme is a cryptographic primitive where two or more parties want to communicate securely, so all parties agree in a shared key from a secret of each party. The agreement has to be secure such that parties do not disclose their secrete to anyone. The first key agreement scheme based in bilinear pairings was proposed by Joux in [27]. This scheme consider three parties A, B, C, with secrete keys $a, b, c \in \mathbb{Z}_r$ respectively. 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 in the Diffie-Hellman problem for bilinear pairings [68]. That work was latter extended to multiple parties by Barua *et al.* in [28].

Chen *et al.* proposed an [20] a IBE key agreement scheme called ID-KEM (Identity-based Key Encapsulation Mechanism). The ID-KEM scheme follows the idea of hybrid encryption, where an asymmetric scheme is used for agreeing a shared key, then the shared key is used for encrypt/decrypt a message symmetrically. The ID-KEM scheme requires three hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_r$, $H_2 : G_T \to \{0,1\}^n$, and $H_3 : (\{0,1\}^*, P_{pub}) \to \mathbb{Z}_r$; where P_{pub} is PKG's public key. The ID-KEM scheme consists of four steps: master key generation, private key extraction, encapsulation, decapsulation.

Master key generation: The PKG chooses a random element $s \in \mathbb{Z}_r^*$ and sets its public key as $P_{pub} = sP$.

Private key extraction: The PKG computes the receiver's private key as $D_{ID} = \frac{1}{s+H_1(ID)}P$.

Encapsulation: The transmitter chooses a random encryption key to be shared (M). The transmitters encapsulates the shared key as the pair (U, V), where $U = t \cdot (P_{pub} + H_1(ID)P)$, $V = M \oplus H_2(e(P, P)^t)$, and $t = H_3(M, P_{pub})$.

Decapsulation: The receiver retrieves the shared key from the pair (U, V), by first computing $M = H_2(e(U, D_{ID})) \oplus V$ then $t = H_3(M, P_{pub})$. The shared key M is valid if $t \cdot (P_{pub} + H_1(ID)P) = U$.

The ID-KEM scheme holds due to the equation 2.19

$$\begin{aligned} e(U, D_{ID}) &= e(t \cdot (P_{pub} + H_1(ID)P), \frac{1}{s + H_1(ID)}P) \\ &= e(t \cdot (sP + H_1(ID)P), \frac{1}{s + H_1(ID)}P) \\ &= e(P, P)^t \end{aligned} \tag{2.19}$$

In works [18,19], they are reported other schemes for key agreement under the IBE paradigm.

Summary

Bilinear pairings is the basic operation for cryptography named Pairing-based cryptography. Bilinear pairings are defined over elliptic curves, which in turn are defined defined over finite fields. In order to design hardware architecture that computes efficiently a bilinear pairing, it is necessary to understand all concepts related with the bilinear pairing operation. Those concepts includes the special characteristics implied in the arithmetic over finite fields, in specific the binary field; the main concepts regarding the theory of elliptic curves applied to cryptography; and the definition, properties and algorithms for computing bilinear pairings. All these concepts have been introduced in this chapter.

This chapter also presents a brief introduction to several cryptographic schemes that belongs to the PBC. Among the presented schemes are encryption schemes like the one reported in [2], digital signature scheme like the one reported in [21], and key agreement schemes like the one reported in [27].

Next chapter describes the proposed flexible architecture for pairing computation, standing out the requirements considered for the design.

Chapter 3

Pairing cryptoprocessor design

Several algorithms have been proposed in the literature for computing pairings; however, there is no protocol or standard that defines a specific algorithm or parameters to be considered in practical Pairing-based cryptography. 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 main objective of the proposed pairing cryptoprocessor is to bring flexibility for the computation of bilinear pairings over binary fields, which is achieved in this thesis by designing a programmable coprocessor.

The main components of the proposed coprocessor for pairing computation are optimized modules for $\mathbb{F}_{2^m}$ arithmetic, whereas the control and flexibility of the coprocessor is achieved by the programmability of the architecture by means of an instruction set. The design of the coprocessor is guided by the goal of achieving a high flexibility with the minimum penalty in performance and area consumption. In order to reach a security level of 128 bits, the underlying finite field $q = \mathbb{F}_{2^m}$ to define the elliptic curves and the extended finite field has the order $m = 1223$ [58].

In this chapter, it is presented the hardware architecture of the proposed programmable cryptoprocessor, which is designed for computing bilinear pairing algorithms. First, the general design specifications are exposed, which are the basic rules considered during the design process. Then, the *Instruction Set Architecture* is exposed, which is formed by the set of instructions supported by the cryptoprocessor and the instruction format specified. After, the datapath that brings support to the instruction set architecture is described, giving a detailed explanation of each architectural module. At the end of the chapter, it is explained the main feature of the proposed cryptoprocessor, the programmability.

3.1 Design specifications

The design process was ruled by three general specifications:

- The architecture should only support arithmetic in $\mathbb{F}_{2^m}$.
- Only operations among registers are supported.
- The multiplication, squaring and square root are always preceded by an addition.

Each specification in the design process is explained and the impact of that specification in the coprocessor architecture is also detailed.

The first specification is that the architecture should only support arithmetic in $\mathbb{F}_{2^m}$. The pairing algorithms require as input points of an elliptic curve which is defined over $\mathbb{F}_{2^m}$, and so the elliptic curve points are represented as a pair (x, y) with coordinates in $\mathbb{F}_{2^m}$. All operations required by the pairing algorithm such as the Miller's one can be translated into arithmetic in $\mathbb{F}_{2^m}$. In the case of the final exponentiation, the required arithmetic operations are over the extended field $\mathbb{F}_{2^{km}}$. However, as it was shown in section 2.1.1, the arithmetic in extended fields can be also translated to simpler arithmetic operations in $\mathbb{F}_{2^m}$ independently of the tower field used.

A direct impact of the first specification is that 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. The number of registers required is a key factor to meet this specification. 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 are 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.

The second specification implies two considerations for the design. First, 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.

The repercussion of the third specification is a trade-off 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 programming 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.

3.2 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 2.1.1. 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 pairing algorithms reported in [32, 37, 57, 67] were analyzed. It was noted that Miller's algorithm performs operations where there are four main inputs, the coordinates of the input points of the elliptic curve. During Miller's algorithm computation, those points are mapped into an element on the extended field $\mathbb{F}_{2^{km}}$. Then during the final exponentiation, operations are performed over elements in $\mathbb{F}_{2^{km}}$.

A natural way to organize the working register is by grouping the registers 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 $k \leq 4$ [36]. Therefore, each bank comprises up to 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.

3.2.1 Instruction set

The complete instruction set comprises 11 instruction described in this section.

Addition$(D[], S[])$. It computes an addition up to four elements in $\mathbb{F}_{2^m}$ stored in a bank. 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 $\mathit{Addition}(D[], S[])$ is executed in one clock cycle. The source bank conserves its values unless the source and destination bank are the same. When only one register is indicated in the source bank, it implies just a movement of registers. For example, $\mathit{Addition}(G[0], F[0])$ implies that the content of register F_0 is just moved to register G_0. Examples:

$\mathit{Addition}(G[0], F[0,1,2])$:	Computes $F_0 + F_1 + F_2$ and stores the result in register G_0.
$\mathit{Addition}(G[0,3], F[1,3])$:	Computes $F_1 + F_2$ and stores the result in register G_0 and G_3.
$\mathit{Addition}(G[2], F[1])$:	Move the value of register $F1$ to the register $G2$.

$\mathit{Squaring}(D[], S[])$. It computes the squaring of the addition up to four elements in $\mathbb{F}_{2^m}$ stored in a bank. S indicates the source bank. D indicates the destination bank. Inside brackets are indicated which registers of S and D are accessed for being read or written respectively. The instruction $\mathit{Squaring}(D[], S[])$ is executed in one clock cycle. Examples:

$\mathit{Squaring}(G[0], F[0,1,2])$:	Computes $(F_0+F_1+F_2)^2$ and stores the result in register G_0.
$\mathit{Squaring}(G[0,3], F[1,3])$:	Computes $(F_1 + F_2)^2$ and stores the result in register G_0 and G_3.
$\mathit{Squaring}(G[1], F[3])$:	Computes $(F_3)^2$ and stores the result in register G_1.

$\mathit{SquareRoot}(D[], S[])$. It computes the root squaring of the addition up to four elements in $\mathbb{F}_{2^m}$ stored in a bank. S indicates the source bank. D indicates the destination bank. Inside brackets are indicated which registers of S and D are accessed for being read or written respectively. The instruction $\mathit{SquareRoot}(D[], S[])$ is executed in one clock cycle. Examples:

$\mathit{SquareRoot}(G[1], F[3])$:	Computes $\sqrt{(F_3)}$ and stores the result in register G_1.
$\mathit{SquareRoot}(G[0,3], F[1,3])$:	Computes $\sqrt{(F_1 + F_2)}$ and stores the result in register G_0 and G_3.

LoadMult$(S_2[], S_1[])$. It loads a new pair of operands and starts the multiplication of them. Each operand is the addition up to four elements in $\mathbb{F}_{2^m}$ stored in a bank. S_1 indicates the source bank for the first operand, while S_2 indicates the source bank for the second operand. Inside brackets are indicated which registers of S_1 and S_2 are read. Examples:

LoadMult$(F[0,1], G[2,3])$:	Begins the multiplication $(F_0+F_1)\cdot(G_2+G_3)$.
LoadMult$(F[2], G[0,1,2,3])$:	Begins the multiplication $(F_2)\cdot(G_0+G_1+G_2+G_3)$.

MoveBank(D, S). It copies the values from the bank S to the bank D. For the instruction *MoveBank*(D, S) the four registers of bank S are read and the four registers of bank D are written. The instruction *MoveBank*(D, S) is executed in one clock cycle. Example:

MoveBank(G, W):	Copies the content of the four registers in the bank W to the bank G, such that $G_0 = W_0$, $G_1 = W_1$, $G_2 = W_2$ and $G_3 = W_3$.

StoreMult$(D[])$. It stores the result of a multiplication operation in the bank D. Inside brackets is indicated which register in the bank D are written. Notice that more than one register of bank D can be written at the same time with the same result.

IncG0(). It increments the register G_0 by 1. The register G_0 was chosen to be attached with extra hardware in order to perform the operation $A \oplus 1$ implied by the second design specification.

Wait(n). It freezes the IP register for n clock cycles.

For(n). Hardware support to the instruction FOR-Loop n.

Jmp(n). Unconditional jump to the instruction at address n.

Jz(n). Conditional jump to the instruction at address n.

3.2.2 Instruction format

The arithmetic instructions were structured in order to preserve a unified format. For the addition, squaring and square root, the instructions have two operands, a source bank and a destination bank. The multiplication, different to the squaring and the square root, requires two operands and produces one output. The multiplication instruction was split into two instructions for preserving the same format as other instructions. The instruction *LoadMult*$(S2[], S1[])$, has two operands for loading and starting a new multiplication, while the instruction *StoreMult*$(D[])$ only has one operand for storing the multiplication result, for this case the second operand is just ignored during the decoding.

The case of the instruction *IncG0*() is also special. This instruction computes the operation $A = A \oplus 1$ requiring just a single NOT gate to be implemented. It was decided that this operation would be attached to a unique register in order to reduce the resource consumption, specifically the register G_0; otherwise, it was necessary to include additional hardware for integrating that single NOT gate into the architecture, which was not worthwhile. This instruction does not contradict the instruction format as both operands are ignored during the decoding.

The instruction *MoveBank*$(D[], S[])$ is used to copy the values stored in a Bank to another. It uses the same format as the previous instructions, two operands, a source bank and a destination bank. Besides, there are four control instructions, all with the same format. The four instructions have one single operand. The parameter n was used as a constant value in the four cases.

The instruction format is coded into a 16-bit word. Figure 3.1 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 specific registers within 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 $OP2$.

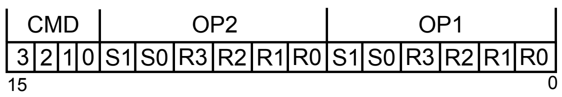


Figure 3.1: Proposed instruction format.

3.3 $\mathbb{F}_{2^m}$ arithmetic modules

In this section, it is explained the design of each architectural module, especially the $\mathbb{F}_{2^m}$ arithmetic modules. All modules are integrated into a final datapath, which along with the instruction set architecture, compose the proposed crypto-processor. Following the idea proposed in the hypothesis, each arithmetic module was designed in order to optimize the processing time and the amount of hardware resources consumed. The most complex arithmetic module is the multiplication, for which especial attention was dedicated to improve the hardware cost and processing time of the multiplication by proposing a modification in the field multiplication algorithm.

3.3.1 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:

$$\begin{aligned} A &= \alpha_{m-1}x^{m-1} + \alpha_{m-2}x^{m-2} + \cdots + \alpha_1 x + \alpha_0 \\ &= \sum_{i=0}^{m-1} \alpha_i x^i \end{aligned}$$

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.

Theoretical cost analysis

Addition is a very simple operation, area cost for a single field addition in $\mathbb{F}_{2^m}$ is m XOR gates and so the time delay is the one of an XOR gate, denoted by T_X.

3.3.2 Modular reduction

Modular reduction is required within multiplication, squaring and inversion. Representing the irreducible polynomial $f(x)$ as a bit-vector notice that $f_m = f_0 = 1$. The operation $xA(x)$ 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 $a_{m-1} = 0$, modular reduction is not necessary because $xA(x)$ is in the field $\mathbb{F}_{2^m}$. However, if $a_{m-1} = 1$, the operation $xA(x)$ is not in the field $\mathbb{F}_{2^m}$, so the resulting polynomial is reduced mod $f(x)$, following equation 3.1, which defines $xA(x) \bmod f(x)$ considering $f_m = f_0 = 1$, where $\oplus$ represents a bitwise XOR operation and $\odot$ represents a bitwise AND operation.

$$\begin{aligned} xA(x) \bmod f(x) \;&=\; (a_{m-2} \oplus [f_{m-1} \odot a_{m-1}], a_{m-3} \oplus [f_{m-2} \odot a_{m-1}], \ldots \\ &\quad , a_0 \oplus [f_1 \odot a_{m-1}], a_{m-1}) \end{aligned} \tag{3.1}$$

Equation 3.1 is well modeled by the Linear Feedback Shift Register (LFSR) shown in figure 3.2. The combinatorial logic denoted as CL-LFSR performs the required arithmetic to compute $xA(x) \bmod f(x)$. Therefore, d CL-LFSR blocks could be connected in a cascade fashion to implement a parallel LFSR (PLSFR) and to obtain $x^d A(x) \bmod f(x)$ in just one iteration. More details on the LFSR and the PLFSR are described in [69].

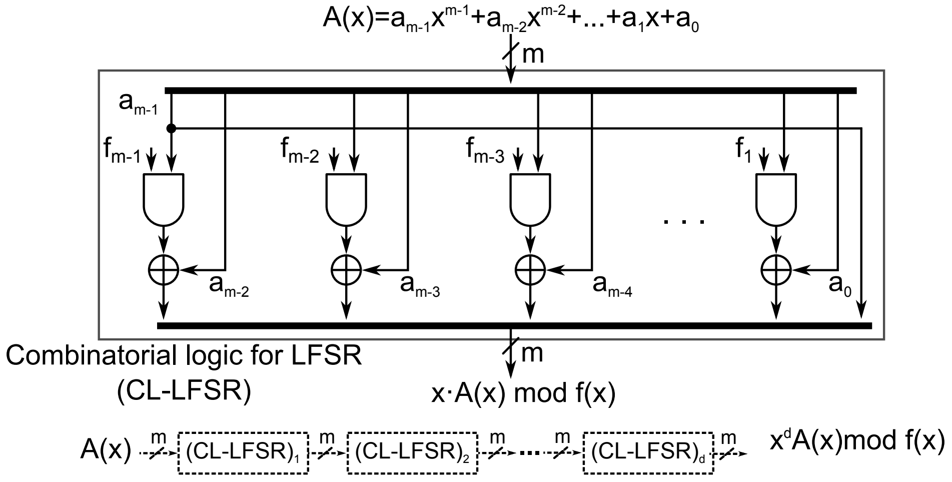


Figure 3.2: Modular reduction using Linear Feedback Shifts Registers.

Theoretical cost analysis

To calculate the PLFSR theoretical cost, only the special case of $f(x)$ as a trinomial of the form $f(x) = x^m + x^a + 1$ is considered. It is observed that only $f_m = f_{m-a} = f_0 = 1$, as a consequence most AND gates of a single LFSR in figure 3.2 have 0 at its outputs, and most XOR gates perform a simple right shift ($\alpha_{m-i} \leftarrow \alpha_{m-i-1}$), see equation 3.1. Simple shifts have no cost in hardware; therefore from all gates in a LFSR only one XOR gate is necessary to calculate $xA(x)$ when $f(x)$ is a trinomial. For computing $x^dA(x) \mod f(x)$, the number LFSR required for the PLFSR is multiplied by the number of XORs per LFSR. A total of d XOR gates are required for trinomials.

For computing a modular reduction using PLFSR, consider a polynomial $g(x)$ of degree $2m-1$ which is the output size of a multiplication or an squaring operation. The polynomial $g(x)$ can be written as equation 3.2.

$$\begin{aligned} g(x) &= (a_{2m-1}x^{m-1} + ... + a_m)x^m + a_{m-1}x^{m-1} + ... + a_0 \\ &= g_2(x)x^m + g_1(x) \end{aligned} \tag{3.2}$$

The polynomial $g_1(x)$ does not require modular reduction, but $g_2(x)x^m$ does and it can be implemented using PLFSR. The modular reduction of $g(x)$ is computed following the equation 3.3.

$$g(x) \mod f(x) = g_2(x)x^m \mod f(x) + g_1(x) \tag{3.3}$$

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 when $f(x)$ is a trinomial. This reduction technique is cheaper against the conventional method documented in [52] which requires a total of $2m + a$ XOR gates for trinomials.

Regarding the time delay, having PLFSR implies several LFSRs connected in cascade, however corresponding time delay is not equivalent to the number of LFSRs. The output of a XOR is also shifted and requires m shifts to return to the original position. Most gates are simple shifts when using trinomial as irreducible polynomial, with no hardware cost. For computing $x^dA(x) \mod f(x)$, d LFSRs are connected in cascade, if $d < m$ the time delay is the delay of one XOR gate, lets say T_X. If $m \leq d < 2m$, implies that at least two XOR gates are in cascade so the time delay is $2T_X$. Formally the time delay of the PLFSR is $(1 + (d \mod m))T_X$ for trinomials.

For computing the modular reduction of the polynomial $g(x)$ presented above, a total of m shifts are required which results in a delay of $2T_X$. An extra XOR gate delay is necessary for adding the least significant bits that not required modular reduction. The total time delay for the modular reduction is $3T_X$. The conventional method has the same time delay as reported in [52].

3.3.3 Novel KAO-LFSR multiplier

In order to optimize the cost of the field multiplication, a modification of the KOA algorithm was proposed in this thesis, which integrates the modular reduction step within the algorithm itself. Considering the step 9 of algorithm 1 presented in section 2.1.1, where z_2 is multiplied by x^n and z_1 by $x^{n/2}$, the proposed approach takes advantage of the modulo operation and integrates the modular reduction step within KOA algorithm through equation 3.4.

$$\begin{aligned} C &= C'(x) \mod f(x) \\ &= (z_2 x^n + z_1 x^{n/2} + z0) \mod f(x) \\ &= z_2 x^n \mod f(x) + z_1 x^{n/2} \mod f(x) + z_0 \end{aligned} \tag{3.4}$$

In the previous section, it was demonstrated that equation 3.4 can be solved using LFSR. Following this approach, two PLFSR are required to compute z_2x^n mod $f(x)$ and$z_1x^{n/2} \mod f(x)$. Therefore, the total number of shifts required are: n shifts for $z_2x^n \mod f(x)$ and $n/2$ shifts for $z_1x^{n/2} \mod f(x)$.

Different to the example shown in section 2.1.1, the field size is generalized to be of any size. The proposed strategy is similar to that used in [70] and [71]. It consists in splitting the input bit vectors by half using the function ceiling $\lceil \cdot \rceil$ to ensure an integer result since m is generally an odd number, see equation 3.5:

$$A(x) = (\underbrace{\alpha_{m-1}, \alpha_{m-2}, \cdots, \alpha_{\lceil m/2 \rceil}}_{A^H}, \underbrace{\alpha_{\lceil m/2 \rceil - 1}, \cdots, \alpha_0}_{A^L}) \tag{3.5}$$

where A^L size is $\lceil m/2 \rceil$ and A^H size is $m - \lceil m/2 \rceil$. The partition of a field element as shown in equation 3.5 is used in every recursive call of the KOA algorithm. To determine if a reduction is necessary, the first call is invoked with $n = m$, the input operands size is m and the result size is $2m - 1$, thus a reduction is necessary. In the next calls to KOA for computing z_0 and z_1, $n = \lceil m/2 \rceil$ and the result is size m so no modular reduction is required. The KOA call for computing

z_2 neither requires modular reduction since $n = m - \lceil m/2 \rceil$. In the subsequent recursive calls to KOA, the operands are smaller so no more modular reductions are required until the basic case of KOA algorithm, that makes all the KOA calls to return. In total, only one reduction at the first call is necessary following the proposed approach.

Algorithm 5 presents the proposed Karatsuba-Ofman algorithm based on Linear Feedback Shift Registers. It is worth to mention that the result C is already reduced $\bmod f(x)$. Steps 4 and 5 use the splitting strategy explained in equation 3.5 whereas steps 6-8 perform the recursive calls. Step 9 evaluates $n = m$ which is true only for the first call when using PLFSRs. For the rest of the calls $n < m$ and partial results sizes are smaller than m therefore a reduction is not needed.

Algorithm 5 KOA-LFSR[n,A,B]: Recursive Karatsuba-Ofman algorithm with reduction step integrated using LFSR.

```
Require: $n$ an integer smaller or equal to $m$; $A, B \in \mathbb{F}_{2^m}$
Ensure: $C = A \cdot B \mod f(x)$
 1: if $n = 1$ then
 2:     return $A \odot B$
 3: end if
 4: $A \leftarrow A^H x^{\lceil n/2 \rceil} + A^L$
 5: $B \leftarrow B^H x^{\lceil n/2 \rceil} + B^L$
 6: $z_2 \leftarrow$KOA-LFSR$[n - \lceil n/2 \rceil, A^H, B^H]$
 7: $z_0 \leftarrow$KOA-LFSR$[\lceil n/2 \rceil, A^L, B^L]$
 8: $z_1 \leftarrow$KOA-LFSR$[\lceil n/2 \rceil, (A^L + A^H), (B^L + B^H)] + z_2 + z_0$
 9: if $n = m$ then
10:     $C(x) \leftarrow z_2 x^{2\lceil n/2 \rceil} \mod f(x) + z_1 x^{\lceil n/2 \rceil} \mod f(x) + z_0$
11: else
12:     $C(x) \leftarrow z_2 x^{2\lceil n/2 \rceil} + z_1 x^{\lceil n/2 \rceil} + z_0$
13: end if
14: return $C$
```

As an example, consider the binary field $\mathbb{F}_{2^{163}}$, see figure 3.3. During the first call, the KOA-LFSR algorithm splits the input operands according to equation 3.5 and makes three recursive calls (steps 6-8) with $n \in \{95, 96, 96\}$ at a 2nd recursion level. In the second call $n = 95$, so the KOA-LFSR splits the inputs again and invokes three recursive calls with $n \in \{47, 48, 48\}$. For $n = 96$, recursive calls are with $n \in \{48, 48, 48\}$, these calls are at the third recursion level. Splits and recursive calls continue until the basic case when $n = 1$ and a single multiplication is carried out with a simple AND gate.

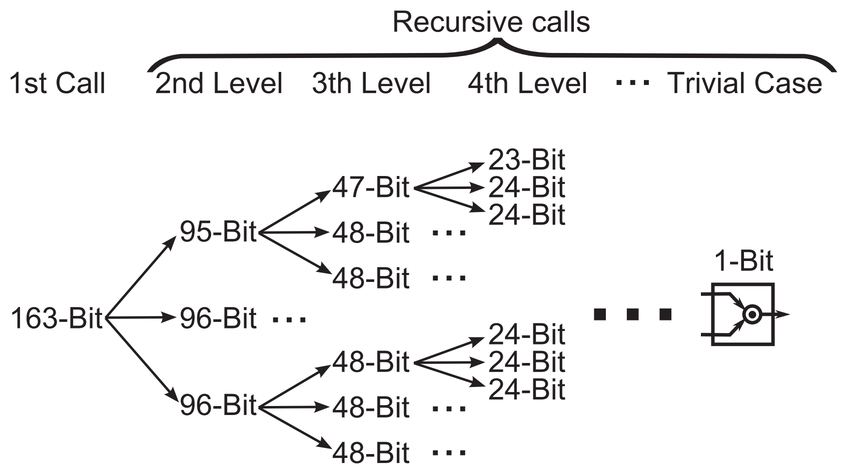


Figure 3.3: Recursive calls tree for the KOA-LFSR algorithm for $\mathbb{F}_{2^{163}}$.

Figure 3.4 shows a diagram with the fully parallel KOA algorithm based on the LFSR multiplier. Figure 3.4(a) shows the block diagram of the circuit for computing the first call of KOA, where modular reduction is required and it is implemented using PLFSR. Figure 3.4(b) is the block diagram of the circuit that computes all the subsequent call of KOA for the case $n < m$. In this case simple shifts are performed instead of PLFSR.

Theoretical cost analysis

The novel KOA-LFSR for finite field multiplication presented in the previous section leads to the next space and time complexity analysis. To simplify this analysis, only the special case of m being an even number is considered, that is:$\lceil m/2 \rceil = m/2$.

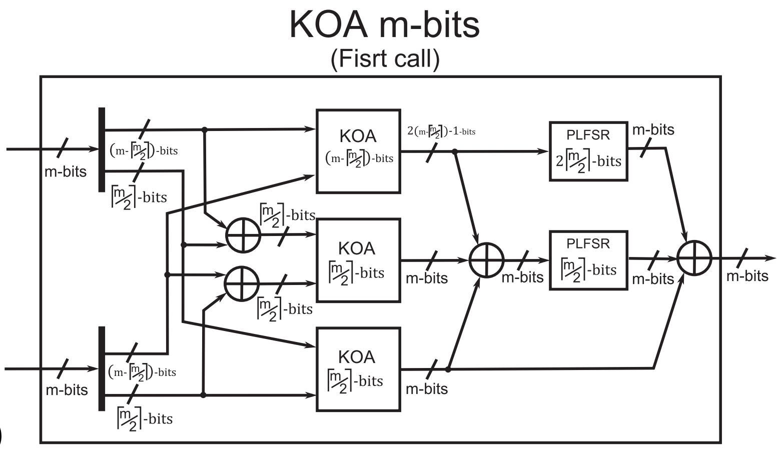
Let S be the cost in area of the KOA-LFSR algorithm hardware implementation. If $m = 1$, the total cost is only one 1-bit AND gate. If $m > 1$, the total cost is given by three KOA recursive calls with half size operands: $3 \cdot S_{m/2}$. In addition, according to algorithm 5 the following XOR gates are also needed when no reduction is required:

1. Two $(n/2)$ XOR gates to perform $(A^L + A^H)$ and $(B^L + B^H)$, step 8.

2. Two $(n-1)$ XOR gates to add three $(n-1)$-bit numbers, step 8.

3. One $(n-1)$-bit XOR for the addition in step 12.

(a)



(b)

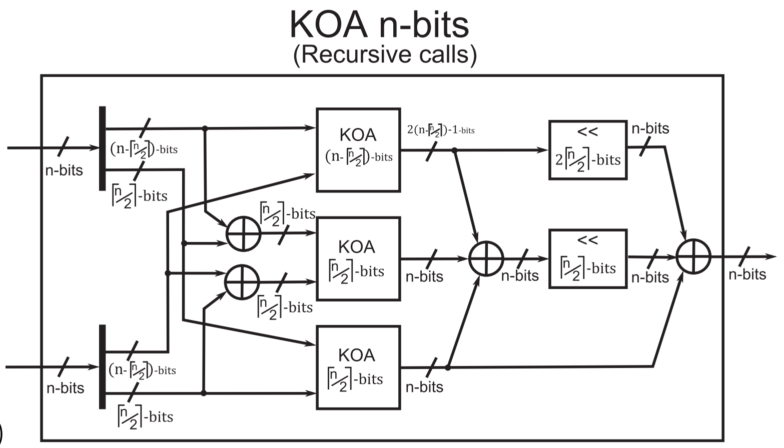


Figure 3.4: Fully Parallel Karatsuba-Ofman Multiplier based on LFSR for $\mathbb{F}_{2^m}$. 3.4(a) First call ($n = m$) using Parallel Linear Feedback Shift Registers. 3.4(b) Recursive calls ($n < m$) only use simple shift.

The total number of XOR gates is $4n - 3$.

It has been stated that PLFSRs are used only in the first call of KOA-LFSR algorithm. Thus, only in that first call two $(m/2)$ XOR gates are required to perform $(A^L + A^H)$ and $(B^L + B^H)$, see step 8 in algorithm 5. Also, two $(m-1)$ XOR gates are required to add z_0, z_2 and $(A^L+A^H)(B^L+B^H)$. Finally, two $(m-1)$ XOR gates are needed to compute the addition at step 10 in algorithm 5. The cost of modular reduction by PLFSR depends on the number of shifts required. This amount of shifts is fixed (m for $z_2x^m \bmod f(x)$ and $m/2$ for $z_1x^{m/2} \bmod f(x)$), the number of XOR needed for these reductions is the number of shifts multiplied by the number of XORs per shift. A total of $3m/2$ XOR gates are required for modular reduction using PLFSRs when $f(x)$ is a trinomial. Finally, Equation

3.6 summarizes the area cost of the hardware implementation of the KOA-LFSR algorithm when the irreducible polynomial is a trinomial, the hardware cost is expressed in the number of required AND gates and XOR gates.

$$S_n = \begin{cases} 1 \cdot AND & \text{if } n = 1 \\ 3S_{n/2} + 4n - 3 \cdot XOR & \text{if } n < m \\ 3S_{n/2} + 11n/2 - 3 \cdot XOR & \text{if } n = m \end{cases} \tag{3.6}$$

Consider T as the time delay required for hardware implementation. Thus, T_A and T_X are the delays for one AND and one XOR gate respectively. If $m = 1$, the total KOA-LFSR delay is T_A. If $m > 1$, the three KOA-LFSR calls in algorithm 5, steps 6-8 can be performed in parallel, each with $T_{m/2}$ time cost. However, KOA-LFSR call at step 8 requires two additions, $(A^H + A^L)$ and $(B^H + B^L)$. These two operations can be performed in parallel with one T_X delay. Once step 8 is completed, another two additions are required leading to a cost of $2T_X$. Finally, one addition is required at step 12, so the total cost is $T_{m/2} + 4T_X$.

For the first call, the use of PLFSRs adds some extra delay. Because at most there are m LFSRs connected inside a PLFSR, only 2 XOR gates are actually in cascade. For trinomials the PLFSR time delay is $2T_X$. Hence, the time delay for the first call is $T_{n/2} + 4T_X + 2T_X$. The overall time delay for the proposed multiplier considering trinomials is expressed by equation 3.7.

$$T_n = \begin{cases} T_A & \text{if } n = 1 \\ T_{n/2} + 4T_X & \text{if } n < m \\ T_{n/2} + 6T_X & \text{if } n = m \end{cases} \tag{3.7}$$

In Table 3.1, a theoretical cost comparison for the KOA algorithm with classical reduction and the proposed KOA-LFSR is presented considering the finite fields defined by trinomials. The theoretical cost for the KOA classic is taken from [72]. It is observed that the proposed KOA-LFSR algorithm achieves a reduction in hardware cost and in time delay required to implement the multiplier in hardware.

Parameter	KOA classic	KOA-LFSR	Case
Area	$1 \cdot AND$	$1 \cdot AND$	$n = 1$
(S_n)	$3S_{n/2} + 4n - 3 \cdot XOR$	$3S_{n/2} + 4n - 3 \cdot XOR$	$n \neq m$
	$3S_{n/2} + 6n + b - 3 \cdot XOR$	$3S_{n/2} + 11n/2 - 3 \cdot XOR$	$n = m$
Time	T_A	T_A	$n = 1$
delay	$T_{n/2} + 4T_X$	$T_{n/2} + 4T_X$	$n \neq m$
(T_n)	$T_{n/2} + 7T_X$	$T_{n/2} + 6T_X$	$n = m$

Table 3.1: Comparison of theoretical cost for the KOA classic vs the proposed KOA-LFSR.

3.3.4 Serial multiplier

A fully parallel Karatsuba-Ofman multiplier of 1223 bits results in a extremely huge design. Therefore, a serial-parallel approach of the Karatsuba-Ofman algorithm (KOA), similar to the one used in [30] was implemented. Inputs of size 1223 are split twice following the KOA principle 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 with KOA to complete the multiplication. Figure 3.5 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.

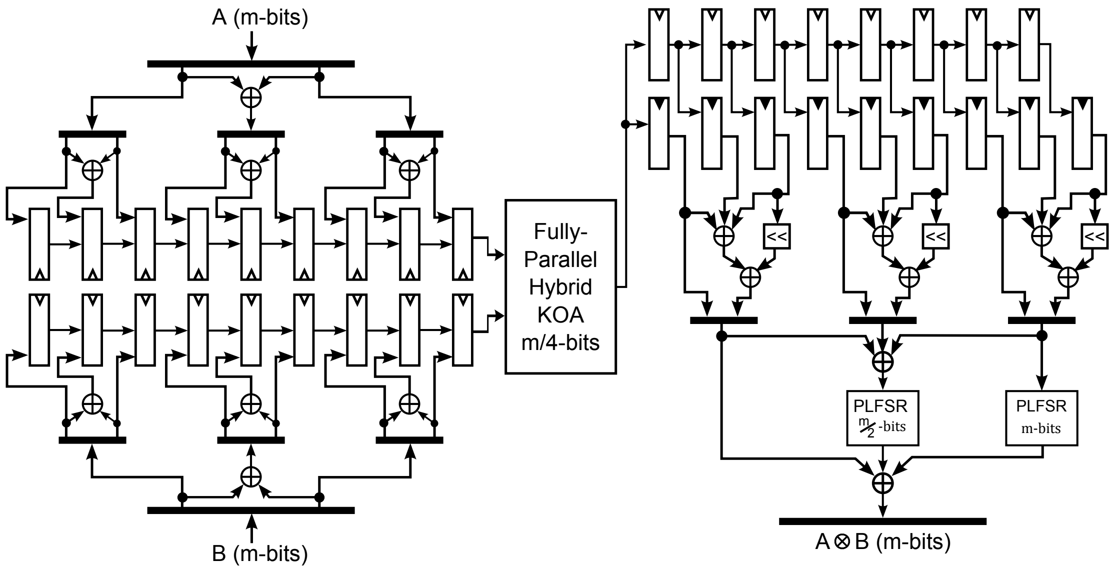


Figure 3.5: Serial-parallel multiplier based on the Karatsuba-Ofman algorithm with modular reduction by Parallel Linear Feedback Registers.

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. With this feature, the datapath does not require to store the multiplication result exactly at the ninth clock cycle but when it is better for the program. The design even allows to start a new multiplication without losing the previous result, in this case the previous result will remain available until the new result is computed. From figure 3.5 the bottom registers at the right of the fully-parallel KOA of $m/4$ bits are used with this purpose. These registers were placed prior to merging the partial results in order to reduce the critical path in the serial multiplier.

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 section 3.3.3. 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 [73] called overlap-free. The overlap-free technique split the input bits in odd indexes and even indexes instead of splitting in the most significant part and less significant part, saving one level of logic at each recursive call consuming the same hardware.

Additionally, Zhou *et al.* shown in [70] that for small inputs size, the schoolbook algorithm presents 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.

Theoretical cost analysis

The theoretical cost of the serial multiplier is divided into three parts: the cost of the fully-parallel hybrid KOA (fph-KOA) module, the cost before the fph-KOA module, and the cost after the fph-KOA module, see figure 3.5. To simplify this analysis, only the special case of m being an even number is considered, that is when $\lceil m/2 \rceil = m/2$.

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 fph-KOA module, the serial multiplier requires $5m/2$ XOR gates.

The output of the fph-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 addionally require. The serial multiplier requires $10m$ XOR gates after fph-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 fph-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 fph-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 3.8 expresses the amount of XOR gates required for s recursive calls of KOA.

$$\begin{aligned} S &= \underbrace{(4n-3)}_{\text{1st level}} + \underbrace{3\left(4\frac{n}{2}-3\right)}_{\text{2nd level}} + \underbrace{9\left(4\frac{n}{4}-3\right)}_{\text{3th level}} + \cdots + \underbrace{3^{s-1}\left(4\frac{n}{2^{s-1}}-3\right)}_{\text{s-th level}} \\ &= \sum_{i=1}^{s} 3^{i-1}\left(4\frac{n}{2^{i-1}}-3\right) \end{aligned} \tag{3.8}$$

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 [70]. 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 fph-KOA module is then expressed by equation 3.9.

$$S = \left[\sum_{i=1}^{s} 3^{i-1} \left(\frac{m}{2^{i-1}} - 3 \right) + 3 \left(\frac{m}{4} - 1 \right)^2 \right] \cdot XOR + \frac{3m^2}{8} \cdot AND \tag{3.9}$$

For the entire serial multiplier, the total amount of resources required is given in table 3.2. Cost is divided in the number of XOR gates, AND gates and registers.

Table 3.2: Theoretical cost of the serial KOA multiplier.

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

Theoretical critical time delay is now discussed. The critical time delay in this case is not the addition of all delays but the longest path between a pair of registers.

Before the fully-parallel hybrid KOA two levels of recursion are performed, each level has the delay of one XOR gate (T_X). After the fph-KOA module those two level of recursion are merged. The first merge is performed using the overlap-free technique with a delay of $2T_X$. The final merge is done by the KOA-LFSR, the delay of this merge is $6T_X$ as stated in section 3.3.3. The delay after the fph-KOA module is $8T_X$. For the fully-parallel hybrid KOA, each level of recursion adds a delay of $3T_X$. The schoolbook algorithm has a critical delay of $\lceil \log_2 w \rceil T_X + T_A$ [70]. Notice that the input size for the schoolbook algorithm depends on the number of recursions s defined before the truncation of KOA. So the delay of the fph-KOA module is $(3s + \lceil \log_2 m/(4^{s+1}) \rceil) T_X + T_A$.

From all previous path analyzed, the longest path between two registers is the path in the fully-parallel hybrid KOA, which is $(3s + \lceil \log_2 m/(4^{s+1}) \rceil) T_X + T_A$.

3.3.5 Squaring

As it is shown in equation 2.7, 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 3.6 illustrate the design.

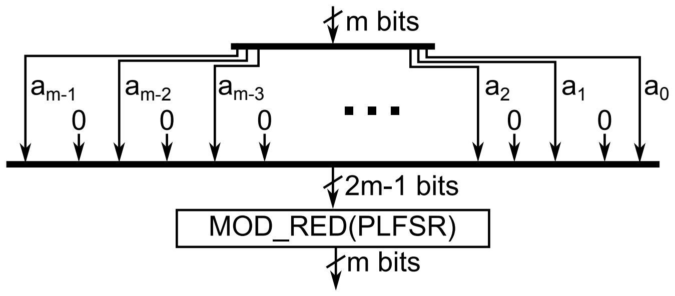


Figure 3.6: Squaring operation over $\mathbb{F}_{2^m}$.

Theoretical cost analysis

The cost in area and time for the squaring module is indeed only the cost of the modular reduction. This cost is because interleaving a '0' between each bit does not represent a hardware cost neither in area nor time.

3.3.6 Square root

Square root is also a cheap operation when using trinomials as irreducible polynomial. Following the algorithm described in section 2.1.1, 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 3.10 performs the computation of $D = \sum d_i x^i{}_{i=0}^{m-1}$, such that $D^2 = A \bmod f(x)$ [52].

$$d_i = \begin{cases} a_{2i} & i < (a+1)/2 \\ a_{2i} + a_{2i-a} & (a+1)/2 \leq i < (m+1)/2 \\ a_{2i-a} + a_{2i-m} & (m+1)/2 \leq i < (m+a)/2 \\ a_{2i-m} & (m+n)/2 \leq i < m \end{cases} \tag{3.10}$$

Theoretical cost analysis

Analyzing equation 3.10, 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. In the same way, the time delay is the delay of that addition, that is only one T_X.

3.4 Cryptoprocessor datapath

The final datapath integrates all architectural modules and connect them with the working registers. Figure 3.7 shows the proposed datapath. It contains six bank registers named F, G, H, I, V, W, each bank has 4 registers of m bits each.

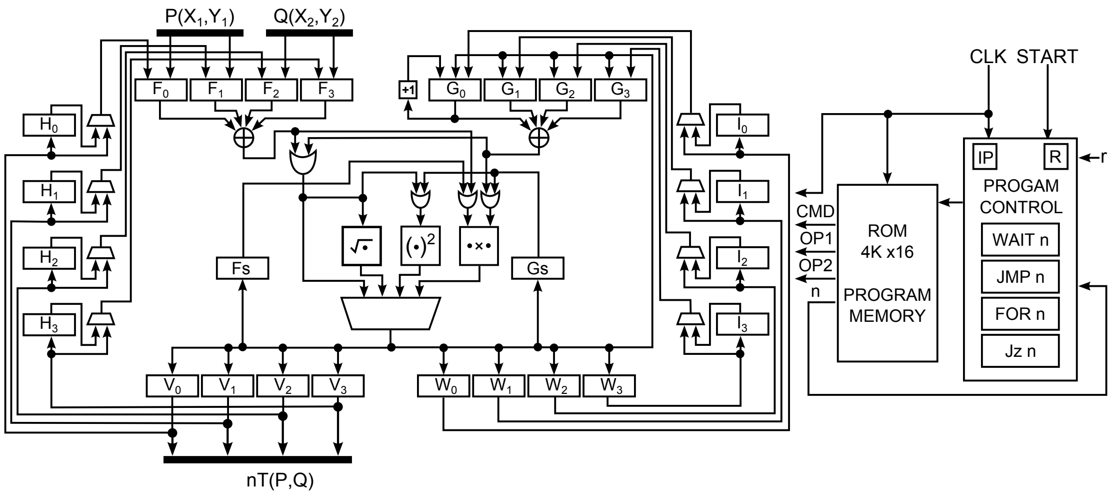


Figure 3.7: Proposed architecture for computing bilinear pairings over binary fields.

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, with the implicit repercussion of a 2-input multiplexer at the output of each register. Notice that banks F 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. For the bank F one input comes from banks V or H, the other comes from the input of the entire cryptoprocessor. For the bank G one input comes from the result of the arithmetic modules, the other comes from banks W or I.

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 using the *MoveBank* instruction, a multiplexer of 2 inputs is placed at the output of each register in bank H and I. The *MoveBank* instruction only supports certain movements: 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. Register Gs can be also used as input in the squaring module. 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 register G_0 is equipped with extra hardware for computing $G_0 = G_0 \oplus 1$.

Theoretical cost analysis

The theoretical cost in hardware of the complete cryptoprocessor is the addition of each module cost, plus the extra hardware required for the interconnection. 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 3.7, five OR gates of m bits are used to drive correctly the data. In total $5m$ OR gates are required.

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. At the output of banks H and I, there are eight multiplexers of 2 bits. At the input and output of banks F and G there are eight multiplexers of 2 bits per bank. Each multiplexer is of size m bits. 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. A total of $26m$ registers is required.

In table 3.3, 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.

Regarding the theoretical critical time delay, recall that the critical time delay in this case is the longest path between a pair of registers.

First lets analyze the path that follows the data that pass through the squaring module. The path begins at bank F or G, data go through a multiplexer before

Table 3.3: 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	-	-	$5m$	$27m$	$26m$
Total	$\sum_{i=1}^{s} 3^{i-1}\left(\frac{m}{2^{i-1}}-3\right)+3\left(\frac{m}{4}-1\right)^2+\frac{(40m-1)}{2}$	$(3m^2)/8$	$5m$	$27m$	$39m$

leaving the bank, then a 4-input XOR gate and an OR gate before entering the squaring module. The delay of the squaring is $3T_X$. After the squaring module, data travel along the 4-input multiplexer and reach a register. This path has a delay of $3T_M + T_O + 5T_X$, where T_M is the delay of a multiplexer of 2 inputs and T_O is the delay of an OR gate.

Now considering the square root, prior the square root module, the data pass through the same path that for the squaring plus an extra OR gate. The delay of the square root is in fact one T_X. Data also pass through a 4-input multiplexer and then finally reach a register. This path has a delay of $3T_M + 2T_O + 3T_X$. When the data comes from the register Gs the path is evidently shorter so it is not considered for analyzing the critical path.

The analysis of the multiplication module is divided into three parts because there are some registers within the serial multiplier. The data from bank F or G go through the multiplexer within the register, a 4-input XOR gate and an OR gate before the multiplier, then inside the multiplier the signal has a delay of $2T_X$ before it reaches a register. This path has a delay of $T_M + T_O + 4T_X$.

From section 3.3.4 it was known that the delay between registers within the serial multiplier modules is $(3s + \lceil\log_2 m/(4^{s+1})\rceil)T_X + T_A$, where s is the number of recursive calls of the fph-KOA module.

Before leaving the serial multiplier module, data have a delay of $8T_X$. Then data go across the multiplexer of 4 inputs and reach a register. This other path has a delay of $2T_M + 8T_X$.

From this five paths, the longest one is again the delay in the fully-parallel hybrid KOA module inside the serial multiplier. That is, the critical time delay is defined for the expression $(3s + \lceil \log_2 m/(4^{s+1}) \rceil)T_X + T_A$.

3.5 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.

Lets briefly introduce the control instructions: *Jmp*, *For*, *Wait* and *Jz*. Instruction *Jmp*(n) is used to perform an unconditional jump to the address specified by n. The instruction *Wait*(n) is used to freeze the IP register for n clock cycles. The instruction *For*(n) is used to support a For-Loop in hardware with exactly n iterations. At each iteration, if $n = 0$, the 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, the IP register increments by 2. Theses control instructions allow the cryptoprocessor to present a more versatile behavior and therefore computing more complex operations found in pairing algorithms.

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 [37]. 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.

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.

3.6 Programmability

The instructions set along with the datapath allow a lot of flexibility for pairing computing because of its programmability. The algorithm for computing the η_T presented in chapter 2 is rewritten in algorithm 6 for easier understanding.

Algorithm 6 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:     $s \leftarrow x_1 + \gamma$, $x_1 \leftarrow \sqrt{x_1}$,$y_1 \leftarrow \sqrt{y_1}$
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 $F^{(2^{2m}-1)\cdot(2^m+1-\epsilon 2^{(m+1)/2})}$
```

Consider the algorithm depicted in algorithm 6 and assume that registers F_0 to F_3 contain the values x_1, y_1, x_2, y_2 as shown in figure 3.7. 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*$(W[2], G[0, 1, 2, 3])$. Notice that when only one register is accessed at the source bank, the instruction *Addition*$(D[], S[])$ is equivalent to just move one register to other, for example the instruction *Addition*$(G[0], F[2])$ is equivalent to $G_0 = F_2$.

Consider the operation $y_1 + y_2 + \frac{1-\beta}{2}$ in line 2 of algorithm 6, where the result depends on the value of β. $\frac{1-\beta}{2} = 0$ when $\beta = 1$, for computing $G_0 = y_1 + y_2$ the instruction *Addition*$(G[0], F[1, 3])$ is enough. Otherwise $\frac{1-\beta}{2} = 1$ when $\beta = -1$, the instruction *Addition*$(G[0], F[1, 3])$ followed by the instruction *IncG0*$()$ are required for computing $G_0 = y_1 + y_2 + 1$.

Now consider the operation $s \cdot (x_1 + x_2 + \gamma)$ in line 5 of algorithm 6, both operands depend on the value of γ, and previous that operation it is required to compute $x_1 = \sqrt{x_1}$ (line 4). Every case is supported by the proposed architecture. When $\gamma = 0$, consider the next sequence of instructions 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 $s \cdot (\sqrt{x_1} + x_2)$
```

When $\gamma = 1$, consider the next sequence of instructions.

```
1: $Addition\ (G[0], F[0])$:   move $x_1$ to $G_0$
2: $IncG0()$:                  compute $s = x_1 + 1$
3: $Addition(Fs, G[0])$:       move $s$ to $Fs$
4: $SquareRoot(G[0], F[0])$:   compute $G_0 = \sqrt{x_1}$
5: $IncG0()$:                  compute $G_0 = \sqrt{x_1} + 1$
6: $Addition(G[1], F[2])$:     move $x_2$ to $G_1$
7: $LoadMult(Fs, G[0,1])$:     begin $s \cdot (\sqrt{x_1} + x_2 + 1)$
```

Notice here that the program complexity is close related to 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 6, the operation $y_1 + y_2 + (1 + \gamma) \cdot x_1 + \delta$ 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 for computing 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 [52], so no extra hardware is required for computing this algorithm in the proposed architecture. The implemented Itoh-Tsujii algorithm is depicted in algorithm 7.

Arithmetic in the extended field is easily supported by the proposed cryptoprocessor independently of the tower field. Consider the tower field defined in [37]: $\mathbb{F}_{q^2} = \mathbb{F}_q[u]/(u^2+u+1)$, $\mathbb{F}_{q^4} = \mathbb{F}_{q^2}[v]/(v^2+v+u)$; which forms the basis for $\mathbb{F}_{2^{4m}}$ as $\{1, u, v, uv\}$, where $u^2 = u + 1$ and $v^2 = v + u$. With this tower field squaring and

Algorithm 7 Itoh-Tsujii algorithm for computing the multiplicative inverse operation.

```
Require: An element $a \in \mathbb{F}_{2^m}$, the irreducible polynomial $f(x)$ of degree $m$ which
    defines the finite field, an addition chain $U$ of length $t$ for $m-1$.
Ensure: $a^{-1} \in \mathbb{F}_{2^m}$.
1: $\beta_{u_0}(a) \leftarrow a$;
2: for $i = 1$ to $t$ do
3:     $\beta_{u_i}(a) \leftarrow [\beta_{u_{i_1}}(a)]^{2^{u_{i_2}}} \cdot \beta_{u_{i_2}}(a) \mod f(x)$;
4: end for
5: return $\beta^2_{u_t}(a) \mod f(x)$
```

raising an element to the q-th power are computed with the following equations:

$$G^2 = (g_0 + g_1 + g_3)^2 + (g_1 + g_2)^2 u + (g_2 + g_3)^2 v + g_3^2 uv$$
$$G^q = (g_0 + g_1 + g_2) + (g_1 + g_2 + g_3)u + (g_2 + g_3)v + g_3 uv$$

This operations can be easily computed by the proposed cryptoprocessor. Lets assume that the element $G \in \mathbb{F}_{2^{4m}}$ is stored in the bank G and the result will be stored in the bank W. The next sequence of instructions computes G^2:

```
1: Squaring(W[0], G[0, 1, 3]):   compute $W0 = (g_0 + g_1 + g_3)^2$
2: Squaring(W[1], G[1, 2]):      compute $W1 = (g_1 + g_2)^2$
3: Squaring(W[2], G[2, 3]):      compute $W2 = (g_2 + g_3)^2$
4: Squaring(W[3], G[3]):         compute $W3 = g_3^2$
```

The next sequence of instructions instead computes the element G^q:

```
1: Addition(W[0], G[0, 1, 2]):   compute $W0 = g_0 + g_1 + g_2$
2: Addition(W[1], G[1, 2, 3]):   compute $W1 = g_1 + g_2 + g_3$
3: Addition(W[2], G[2, 3]):      compute $W2 = g_2 + g_3$
4: Addition(W[3], G[3]):         compute $W3 = g_3$
```

Now consider the tower field defined in [67]: $\mathbb{F}_{q^4} = \mathbb{F}_q[u]/(u^4 + u + 1)$; which forms the following basis for $\mathbb{F}_{2^{4m}}$: $\{1, u, u^2, u^3\}$, where $u^4 = u + 1$. Squaring and raising an element to the q-th power are computed with the following equations:

$$G^2 = (g_0+g_2)^2 + g_2^2 u + (g_1+g_3)^2 u^2 + g_3^2 u^3$$
$$G^q = (g_0+g_2) + g_2 u + (g_1+g_3) u^2 + g_3 u^3$$

Instead of the previous sequences of instructions, the following one computes G^2:

1: *Squaring*$(W[0], G[0,2])$: compute $W0 = (g_0+g_2)^2$
2: *Squaring*$(W[1], G[2])$: compute $W1 = (g_2)^2$
3: *Squaring*$(W[2], G[1,3])$: compute $W2 = (g_1+g_3)^2$
4: *Squaring*$(W[3], G[3])$: compute $W3 = g_3^2$

And the next one computes G^q:

1: *Addition*$(W[0], G[0,2])$: compute $W0 = g_0 + g_2$
2: *Addition*$(W[1], G[2])$: compute $W1 = g_2 +$
3: *Addition*$(W[2], G[1,3])$: compute $W2 = g_1 + g_3$
4: *Addition*$(W[3], G[3])$: compute $W3 = g_3$

In these examples, notice how the programmability of the proposed cryptoprocessor is able to compute G^2 and G^q for two different tower fields. No architectural change is required for any case. Indeed, the amount of instructions is the same for both cases.

The distortion map is another parameter usually required by different pairing algorithms, which may be defined in different ways without affecting the security of the system. For example consider the distortion map proposed by Barreto in [37]: $\psi(x,y) = (x+u+1, y+xu+v)$ and the distortion map proposed for Ronan in [67]: $\psi(x,y) = (x+1+u+u^2, y+(x+1)u+u^2)$. In a very similar way as for the tower field, the proposed cryptoprocessor can compute the pairing algorithm using any distortion map.

Different versions of the pairing algorithm are also supported by the programmability. For example a first version, denoted as Barreto-Beuchat version, of the η_T algorithm is depicted in algorithm 8, which computes the Miller's algorithm and it makes uses of the extended field basis presented in [37], the final exponentiation is computed using the algorithm introduced in [57]. Conversely, algorithm 9 depicts a second version, denoted as Ronan version, of the η_T algorithm which was originally presented in [67]. The table 3.4 resumes the different parameters chosen in each version of the η_T algorithm. Notice that the elliptic curve, the tower field and the distortion map are different for the two versions.

Table 3.4: Parameters used in pairing algorithms.

Parameter	Barreto-Beuchat	Ronan
Security level	128 bits	
Finite field	$\mathbb{F}_{q=2^{1223}}$; $f(x) = x^{1223} + x^{255} + 1$	
Embedded degree	k=4	
Elliptic curve	$E : Y^2 + Y = X^3 + X$	$E : Y^2 + Y = X^3 + X + 1$
Tower field	$\mathbb{F}_{q^2} = \mathbb{F}_q[u]/(u^2 + u + 1)$ $\mathbb{F}_{q^4} = \mathbb{F}_{q^2}[v]/(v^2 + v + u)$	$\mathbb{F}_{q^4} = \mathbb{F}_q[u]/(u^4 + u + 1)$
Basis for $\mathbb{F}_{q^4}$	$\{1, u, v, uv\}$ $u^2 = u + 1$ $v^2 = v + u$	$\{1, u, u^2, u^3\}$ $s^4 = s + 1$
Distortion map	$\psi(x, y) = (x + u + 1,$ $y + xu + v)$	$\psi(x, y) = (x + 1 + u + u^2,$ $y + (x + 1)u + u^2)$

In algorithm 8, the line 8 is a multiplication in the extended field commonly known as *spare multiplication* because the second operand is always of the form $G = g_0 + g_1 u + v$, which substantially simplifies the multiplication, only requiring six multiplications over $\mathbb{F}_q$. The final exponentiation take advantages of the tower field in several ways, the partial results T_0 to T_6 and D are indeed elements of $\mathbb{F}_{q^2}$. One multiplication over $\mathbb{F}_{q^2}$ (lines 13 and 16) requires three multiplication and 4 additions over $\mathbb{F}_q$. The inversion of the element D requires three multiplications, two additions, one squaring and one inversion over $\mathbb{F}_q$. Only one multiplication over $\mathbb{F}_{q^4}$ is computed in the Barreto-Beuchat version of the final exponentiation.

Algorithm 9 uses an approach called unrolled loop during the Miller's algorithm. Notice that two elements G_0 and G_1 are computed and multiplied prior the multiplication $F \cdot G$. Due to the distortion map, G_0 and G_1 always have the form $G = g_0 + g_1 u + (g_1 + 1)u^2$, so the multiplication $G_0 \cdot G_1$ is more simplified that the spare multiplication of algorithm 8, it only requires three multiplication over $\mathbb{F}_q$. With the unrolled loop approach, notice that the FOR-loop of the Miller's algorithm only performs the half of iterations. The drawback here is that the line 12 is a full multiplication over $\mathbb{F}_{q^4}$, which requires nine multiplications over $\mathbb{F}_q$. The final exponentiation in this case performs all operations over the extended field. Five multiplications and one inversion over $\mathbb{F}_{q^4}$ are required in this scenario.

Summary

In this chapter, it was presented the proposed cryptoprocessor dedicated for computing bilinear pairings. The design was focused to provide enough flexibility to the cryptoprocessor in order to support different parameters as the distortion map, tower field and elliptic curve. The design targeted a finite field of order $m = 1223$ in order to reach a security level of 128 bits. The cryptoprocessor was ruled under three main specifications: the architecture only brings support to arithmetic operation over $\mathbb{F}_{2^m}$, the parameters of the operations only can be in registers, and the multiplication, squaring and square root are always preceded by an addition. An instruction set architecture, formed by the set of supported instructions and the instruction format, that bring support to all arithmetic operation required by pairing algorithms and program control was presented. Each architectural module was explained in detail. In this sense, a modification on the original Karatsuba-Ofman algorithm for field multiplication which integrates the modular reduction step into the polynomial multiplication was proposed. Finally, the programmability of the cryptoprocessor was explained through several examples.

Algorithm 8 Barreto-Beuchat version of the η_T pairing algorithm.

```
Require: $P = (x_1, y_1), Q = (x_2, y_2) \in E(\mathbb{F}_{2^{1223}})$ and parameters of table 3.4
Ensure: $\eta_T(P, Q) \in \mathbb{F}_{2^{4(1223)}}$
 1: {Miller's algorithm}
 2: $s \leftarrow x_1 + 1$;
 3: $F \leftarrow s \cdot (x_1 + x_2 + 1) + y_1 + y_2 + (y_2 + s) \cdot u + v$;
 4: for $i = 1$ to $(m + 1)/2$ do
 5:     $s \leftarrow x_1$; $x_1 \leftarrow \sqrt{x_1}$; $y_1 \leftarrow \sqrt{y_1}$;
 6:     $G \leftarrow s \cdot (x_1 + x_2) + y_1 + y_2 + x_1 + 1 + (s + x_2) \cdot u + v$;
 7:     $x_2 \leftarrow x_2^2$; $y_2 \leftarrow y_2^2$;
 8:     $F \leftarrow F \cdot G$;
 9: end for
10: {Final exponentiation}
11: $m_0 \leftarrow f_0^2$; $m_1 \leftarrow f_1^2$; $m_2 \leftarrow f_2^2$; $m_3 \leftarrow f_3^2$
12: $T_0 \leftarrow (m_0 + m_1) + m_1 u$; $T_1 \leftarrow (m_2 + m_3) + m_3 u$;
13: $T_2 \leftarrow m_3 + m_2 u$; $T_3 \leftarrow (f_0 + f_1 u) \cdot (f_2 + f_3 u)$;
14: $T_4 \leftarrow T_0 + T_2$; $D \leftarrow T_3 + T_4$;
15: $D \leftarrow D^{-1}$;
16: $T_5 \leftarrow T_1 \cdot D$; $T_6 \leftarrow T_4 \cdot D$;
17: $V_0 \leftarrow T_5 + T_6$;
18: $V_1, W_1 \leftarrow T_5$;
19: $W_0 \leftarrow T_6$;
20: $V \leftarrow V_0 + V_1 v$; $W \leftarrow W_0 + W_1 v$;
21: $V \leftarrow V^{2^m+1}$;
22: for i = 1 to (m+1)/2 do
23:     $W \leftarrow W^2$;
24: end for
25: $F \leftarrow V \cdot W$;
26: $W \leftarrow W^{-1}$;
27: $F \leftarrow F \cdot W$;
28: return $F$
```

Algorithm 9 Ronan version of the η_T pairing algorithm.

```
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: {Miller's algorithm}
 2: $s \leftarrow x_1 + 1$;
 3: $F \leftarrow s \cdot (x_1 + x_2 + 1) + y_1 + y_2 + 1 + (y_2 + s + 1)u + (y_2 + s)u^2$;
 4: for $i = 1$ to $(m + 1)/4$ do
 5:     $s \leftarrow x_1$; $x_1 \leftarrow \sqrt{x_1}$; $y_1 \leftarrow \sqrt{y_1}$;
 6:     $G_0 \leftarrow s \cdot (x_1 + x_2) + y_1 + y_2 + x_1 + 1 + (s + x_2 + 1)u + (s + x_2)u^2$;
 7:     $x_2 \leftarrow x_2^2$; $y_2 \leftarrow y_2^2$;
 8:     $s \leftarrow x_1$; $x_1 \leftarrow \sqrt{x_1}$; $y_1 \leftarrow \sqrt{y_1}$;
 9:     $G_1 \leftarrow s \cdot (x_1 + x_2) + y_1 + y_2 + x_1 + 1 + (s + x_2 + 1)u + (s + x_2)u^2$;
10:     $x_2 \leftarrow x_2^2$; $y_2 \leftarrow y_2^2$;
11:     $G \leftarrow G_0 \cdot G_1$;
12:     $F \leftarrow F \cdot G$;
13: end for
14: {Final exponentiation}
15: $U, V, W, G \leftarrow F$;
16: for i = 1 to (m+1)/2 do
17:     $U \leftarrow U^2$;
18: end for
19: $U \leftarrow U^q$;
20: $W \leftarrow W^q$;
21: $V \leftarrow W$;
22: $W \leftarrow W^q$;
23: $F \leftarrow W$;
24: $W \leftarrow W^q$;
25: $W \leftarrow W \cdot U$;
26: $W \leftarrow W \cdot G$;
27: $F \leftarrow F \cdot V$;
28: $U \leftarrow U^q$;
29: $U \leftarrow U^q$;
30: $F \leftarrow F \cdot U$;
31: return $F$
```

Chapter 4

Implementation results

This chapter describes the experiments carried out in order to validate each architectural module and the entire cryptoprocessor. The design was implemented in a FPGA device, synthesis results are presented and compared with state-of-art works.

4.1 Validation strategy

The hardware designs were modeled using VHDL as a description language. Several experiments were performed in order to validate the correct behavior of each architectural module and the complete architecture. A total of 1000 test data vectors were created randomly in order to validate each hardware arithmetic module; C/C++ routines based on the library Miracl[1] were used to generate test data vectors. 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.

Special attention was dedicated to the multiplication operation due to this thesis presents a novel Karatsuba-Ofman multiplier. In this case, the validation was performed for the different binary fields proposed for cryptography applications; some fields are recommended by the NIST [22], while others are proposed by CERTICOM as a challenge[2]. Table 4.1 summarizes the irreducible polynomials

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

[2]http://www.certicom.com/index.php/curves-list

used to validate the proposes KOA-LFSR multiplier. Notice that only trinomials and pentanomials were considered.

Table 4.1: Irreducible polynomials used to validate the KOA-LFSR multiplier.

Trinomials $f(x) =$	Recommended by	Pentanomials $f(x) =$	Recommended by
$x^{167} + x^{6} + 1$	Other	$x^{131} + x^{13} + x^{2} + x + 1$	Certicom
$x^{191} + x^{9} + 1$	Certicom	$x^{163} + x^{7} + x^{6} + x^{3} + 1$	NIST
$x^{233} + x^{74} + 1$	NIST	$x^{277} + x^{12} + x^{6} + x^{3} + 1$	Other
$x^{239} + x^{36} + 1$	Certicom	$x^{283} + x^{12} + x^{7} + x^{5} + 1$	NIST
$x^{359} + x^{68} + 1$	NIST	$x^{571} + x^{10} + x^{5} + x^{2} + 1$	NIST
$x^{409} + x^{87} + 1$	NIST		

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, both versions were presented in algorithms 8 and 9 in section 3.6.

Finally, in order to validate the cryptoprocessor under some Pairing-based cryptography application, the Identity-based Key Encapsulation Mechanism (ID-KEM) introduced in section 2.3.3 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.

4.1.1 Metrics of performance

Several metrics are used for evaluating hardware architectures, including the amount of area resources, processing time, efficiency, etc. Their definitions are:

Area is a parameter used to measure the amount of hardware resources required by some architecture. The unit used for measure the area depends on the implementation technology used, for an FPGA implementation typically the number of Slices, or the number Look-Up Tables (LUTs) are reported. A Slice is a configurable unit within an FPGA, a slice usually contains a few Look-Up Tables and some registers depending on the FPGA technology. A LUT is logic cell with x inputs used for implementing an arbitrary Boolean function of x inputs. The amount of registers is another metric used for measure the area consumed by some architecture.

Minimum period and *maximum frequency* are parameters used to measure the maximum speed operation of some architecture. The minimum period is the inverse of the maximum frequency. Minimum period is measured in seconds and maximum frequency is measured in hertz. Minimum period is preferred for combinational designs as it represents the maximum path delay. For sequential designs, maximum frequency represents the maximum clock frequency the architecture can operate with.

Clock cycles is a parameter used to measure the amount of clock cycles required by some architecture to complete a determined computation.

Latency is a parameter used to measure the processing time of hardware designs. It is obtained by multiplying the number of clock cycles by the minimum period. The smaller the latency of the design, the better its processing time.

$A \cdot T$ product is a parameter used in hardware architectures comparisons to bring a balanced comparison between the area consumed and the processing time. The $A \cdot T$ product is defined as the area consumed times the processing time, in this case the units used are $Slices \times Seconds$. The smaller the $A \cdot T$ product of the design, the better.

Program memory is a parameter used to measure the amount of RAM or ROM memory required by the hardware implementation for storing the control program. It is measured in bits.

4.2 Implementation of $\mathbb{F}_{2^m}$ arithmetic modules

4.2.1 Implementation results of the KOA-LFSR multiplier

This section discusses the implementation results of the KOA-LFSR multiplier. The proposed multiplier was implemented using VHDL as a description language. For synthesis, Xilinx ISE 13.2 was used targeting the Xilinx Virtex-6 (xc6vlx240t) device and using default synthesis flags. For comparison purposes, the fully-parallel Binary Karatsuba Multiplier (BKM) using the classical reduction presented in [52] was implemented and synthesized. Those results are also presented in this section.

In figure 4.1, it is shown the synthesis results for trinomials whereas in figure 4.2, it is shown the results for pentanomials. These graphs show the tendency of LUTs used and the minimum clock period achieved by each architecture. These results include the total hardware usage necessary for the multiplication and the reduction step. These figures show that the proposed KOA-LFSR algorithm improves the resources consumption and processing time when compared to the BKM with classical reduction.

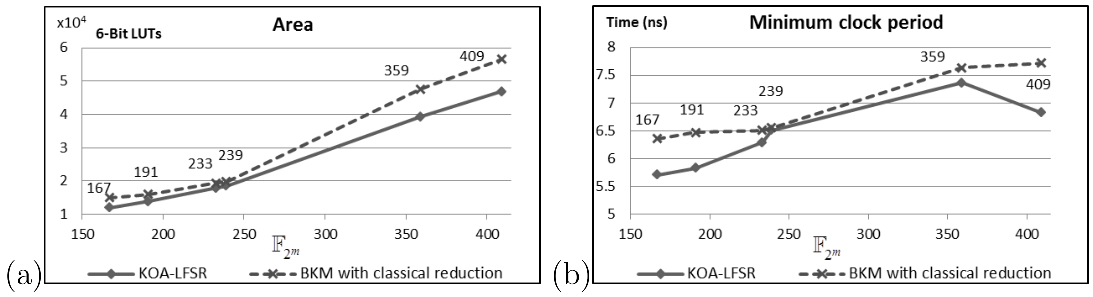


Figure 4.1: Implementation results of KOA-LSFR for trinomials, where $\mathbb{F}_{2^m}$ is the underlying finite field.

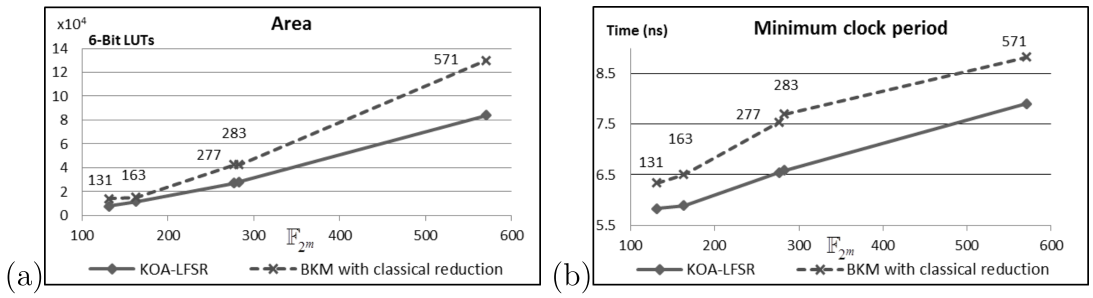


Figure 4.2: Implementation results of KOA-LSFR for pentanomials, where $\mathbb{F}_{2^m}$ is the underlying finite field.

These results not only confirm the theoretical improvement shown in table 3.1, but also demonstrate that the proposed multiplier helps the synthesis tool to optimize the FPGAs resources usage. The KOA-LFSR algorithm has a very regular structure from which the synthesis tool takes advantage and optimizes the result. In figures 4.1 and 4.2, it is observed the area and time tendency when the field size increases. For trinomials it is shown an average improvement of 13.63%

in area and 6.58% in processing time for the proposed KOA-LFSR multiplier; whereas for pentanomials, the average improvement is 35% in area and 11.12% in processing time for the proposed KOA-LFSR multiplier.

In table 4.2, the proposed multiplier is compared to different KOA based multipliers. A brief discussion of other multipliers is discussed below. The KOA-LFSR was re-synthesized for other devices to provide a comparison as fair as possible, a direct comparison is difficult because other works do not always consider the cost of the KOA multiplication and the modular reduction step together. Some authors only work on the polynomial multiplier, others focus on the reduction for general polynomials. Some works do not consider a fully-parallel approach.

Table 4.2: Comparative between the proposed KOA-LFSR and other multipliers.

Ref.	Device	m	Fully parallel?	Includes reduction?	Area	Latency (ns)
[74]	Virtex E	191	yes	yes	6,265 Slices	45.89
KOA-LFSR	Virtex E	191	yes	yes	7,093 Slices	19.31
[70]	Virtex 5	163	yes	no	7,488 LUTs	N/R
KOA-LFSR	Virtex 5	163	yes	yes	7,786 LUTs	5.47
[75]	Virtex 5	128	yes	yes	6,941 LUTs	5.49
KOA-LFSR	Virtex 5	131	yes	yes	6,162 LUTs	5.75
[76]	Virtex II	128	no	no	2,473 Slices	378.00
[76]	Virtex II	128	no	no	3,978 Slices	153.00
KOA-LFSR	Virtex II	131	yes	yes	4,147 Slices	10.12
[76]	Virtex II	240	no	no	4,839 Slices	290.00
KOA-LFSR	Virtex II	239	yes	yes	10,510 Slices	10.71
[77]	Spartan 3	128	yes	no	10,172 Slices	59.52
[77]	Spartan 3	128	no	no	2,528 Slices	515.64
KOA-LFSR	Spartan 3	131	yes	yes	4,205 Slices	13.45
[77]	Spartan 3	256	yes	no	N/R	69.77
[77]	Spartan 3	256	no	no	8,276 Slices	569.04
KOA-LFSR	Spartan 3	239	yes	yes	13,620 Slices	15.05

In [74], a multiplier based on the BKM technique is presented, which truncates the recursion at a predefined number of bits and then uses a more efficient multiplier. The idea in that work is that for small bit-length operands there are better multipliers than the KOA approach. Thus, in that work the recursion is truncated at different levels. Experimental tests with $n \in \{4, 8, 16\}$ are presented with best results when $n = 8$. That work reports the number of slices used and the time required. The modular reduction step is performed with the classic method explained in section 2.1.1.

In [70], authors perform a detailed analysis of several KOA-based multipliers implemented in FPGAs and ASICs. That work considers multipliers that are an hybrid of the KOA and the schoolbook algorithms. First, it analyzes separately both approaches and realizes that the schoolbook method is better for small fields. Then, it implements a KOA multiplier that truncates recursive calls and executes small multipliers with the schoolbook method. The KOA multiplier used on that work uses a splitting strategy very similar to the one used in the KOA-LFSR. In [70], experiments with several multipliers are carried out. In that work the schoolbook method implementation is carried out manually so optimization in area and time do not depends in a synthesis tool. Their results show the number of LUTs required in their design.

In [75], a pipelined KOA based multiplier is presented. That multiplier truncates KOA's recursive calls after some steps and thereafter the schoolbook method is used. Pipeline registers are placed between every KOA recursive call. That design is assessed considering several pipeline stages in order to find the best compromise between area and time. The modular reduction strategy is not explicitly mentioned.

In [76], several combinations of parallel and serial multipliers are provided. Results for a sequential 240-bit multiplier are presented, being comparable against the results of the proposed KOA-LFSR approach for the finite field $\mathbb{F}_{2^{239}}$.

In the work reported in [77], different architectures of Karatsuba multipliers are explored; some of them are fully-parallel while others are a hybrid of parallel and serial multipliers. The fastest (fully-parallel) and the smallest architectures are presented. The reduction step is not considered in that research. Since that work only considers m as a power of 2, fields with order closer to 128 and 256 are chosen for comparison. The number of slices is used for comparing area consumption.

4.2.2 Serial multiplier implementation results

As mentioned in section 3.3.4, for implementing the fully-parallel hybrid KOA module inside the serial multiplier module, the KOA algorithm was truncated after s recursive calls. In [70] 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 4.3: 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 [30], a results comparison against that work is performed. In table 4.4 the best implementation achieved for the serial multiplier is compared against [30]. Notice that the proposed serial multiplier includes the modular reduction inside the multiplier, different to the multiplier reported in [30] which additionally requires a modular reduction module.

Table 4.4: Comparative of $\mathbb{F}_{2^m}$ serial multipliers.

Design	LUTs	FFs	Minimum period	Clock cycles	Latency	$A \cdot T$	Requires reduction?
This	28,518	17,507	3.144 ns	9	28.296 ns	0.80	yes
[30]	30,148	N/R	4 ns	10	40ns	1.21	no

4.3 Cryptoprocessor implementation results

The proposed architecture was implemented using VHDL as a description language. Program memory was implemented with Xilinx's Block Memory Generator LogiCORE. 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 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.

Table 4.5 shows the implementation results of the synthesis process regarding FPGA resource consumption. The required area is 16,451 slices for a Virtex 6 device. From this area, about 43% is used by the field multiplier; being this module the biggest individual one from all arithmetic modules as expected. Nevertheless, the multiplexers inside each bank register consume a great amount of resources, a total of 51%. The remaining 6% of the area is used for the rest of the arithmetic modules. The number of FPGA registers used is also reported in table 4.5, about 33% of the total amount of registers is required by the serial multiplier while the rest is used by the bank registers.

Table 4.5: FPGA resource consumption per architectural module.

Architectural Module	Area (Slices)	Registers
Full Cryptoprocessor	16,451	50,882
Serial Multiplier	7,130	17,507
Datapath	8,403	33,375
Additions	612	0
Squaring	153	0
Square root	153	0

In table 4.6, it is presented the synthesis results for Virtex 4 and Virtex 6 devices. Notice how the area is closely related to the technology, for a Virtex 4 a total of 46,879 are used, considerably more slices than the slices used for a Virtex 6. 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. About the maximum clock frequency, it depends on the longest path delay among two registers. For the proposed architecture this path is inside the serial multiplier with seven levels of logic. The maximum frequency the proposed cryptoprocessor can operate with is 188.9 MHz for a Virtex 6 when using the flag *register_balancing* in the synthesis process. In the same way, time is closely related to 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.

In table 4.7, it is compared the computation of two versions of the η_T algorithm using the proposed cryptoprocessor. Both versions were introduced in algorithm 8 and 9 of section 3.6. The first version computes the Miller's algorithm and it makes use of the extended field basis presented in [37], the final exponentiation is

Table 4.6: Synthesis results of the proposed architecture for two different devices.

Device	Area (Slices)	Maximum Frequency (MHz)
Virtex 6	16,451	188.9
Virtex 4	46,879	85.6

computed using the algorithm introduced in [57]. First version is named Barreto-Beuchat. The second version computes the η_T algorithm as reported in [67]. Second version is named Ronan version.

The column clock cycles shows the total of cycles required to compute the complete pairing. The number of cycles depends directly on the algorithm. Using the proposed cryptoprocessor, the pairing algorithms can be fairly compared because they are implemented using the same platform. Notice also that the amount of program memory required by the Ronan version is almost double than the one required by the Barreto-Beuchat version. That is because the operations in the Ronan version are more dependent, so more instructions $\mathit{MoveBank}(D[], S[])$ are required. Additionally, the final exponentiation in the Ronan version requires a total of five multiplications over $\mathbb{F}_{q^k}$, while final exponentiation in the Barreto-Beuchat version only performs one multiplication over $\mathbb{F}_{q^k}$, so less code is needed.

Table 4.7: Processing time of the proposed architecture for two different version of η_T algorithm.

η_T algorithm	Device	Program memory (kbits)	Clock cycles ($\times 10^3$)	Maximum frequency (MHz)	Latency (us)
Barreto-Beuchat	Virtex 6	5.3	51.5	188.9	273
	Virtex 4	5.3	51.5	85.6	601
Ronan	Virtex 6	10.3	57.6	188.9	305
	Virtex 4	10.3	57.6	85.6	673

4.4 Comparisons

In this section, the implementation results of the proposed cryptoprocessor are compared against state-of-art works.

A comparison against state-of-art custom implementations of the η_T pairing for binary fields is presented in table 4.8. All works reported in this table reach a security level of 128 bits except for [32], which achieves 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 [32] 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 [30] and [31]. The $A \cdot T$ product reached in this work is 4.49, which is just 1.56x bigger than [30] and 2.64x bigger than [31]. Compared to [32] the $A \cdot T$ product is 3.18x bigger, but notice that [32] only reaches a security level of 105 bits, if the results presented in that work are extrapolated the area estimated is more than 130,000 Slices and the estimated processing time is about 33 us. With these estimations the $A \cdot T$ product is 4.29, essentially the same as the $A \cdot T$ of the proposed cryptoprocessor. These results show that custom architectures are slightly faster/smaller than the proposed design, however the proposed architecture reaches a great flexibility which allows the computation of bilinear pairing with different parameters and different versions of the pairing algorithm. The given flexibility justifies the fact that the proposed design does not improve the area and processing time of custom architectures.

Table 4.8: 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	16,451	188.9	51.5	273	4.49
	Virtex 4	46,879	85.6	51.5	601	28.17
[30]	Virtex 6	15,167	250.0	47.6	190	2.88
[31]	Virtex 6	16,403	267.0	27.3	102	1.70
[32]*	Virtex 4	78,874	130.0	2.4	18.8	1.41
*That work targeted a security level of 105 bits.						

Table 4.9 compares the proposed cryptoprocessor against works in the literature that exhibit some degree of flexibility. Notice that in the literature there is no flexible solution for binary fields reported. In table 4.9, the results compared are from the Virtex 4 as works [38,42] used the same device. For the case of work [50],

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 computation time than the proposed design, that work only achieve a security level of 66 bits whereas the proposed architecture achieves a security level of 128 bits, roughly extrapolating the results of [38], the underlying finite field required for reach a security level of 128 bits requires to be 5 times bigger, the area usually grows in quadratic fashion and the time in lineal fashion, so the estimated area is 46,275 Slices and the estimated processing time is 685 us, with a $A \cdot T$ of 31.7, these estimations are essentially the same as the proposed architecture, but the one proposed is for binary fields. Comparing area consumption and processing time with the work reported in [42], the proposed cryptoprocessor outperforms that work in both parameters. Comparison with the ASIP reported in [50] is harder because the target devices of this thesis is a FPGA, not an ASIC; anyway it can be noticed that the proposed architecture is able to execute a pairing algorithm 23x faster.

Table 4.9: 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	$\mathbb{F}_{2^m}$	46,879	85.6	51.5	601	28.2
Ronan	$\mathbb{F}_{2^m}$	46,879	85.6	57.6	673	31.5
[38]*	$\mathbb{F}_{3^m}$	1,851	203	27.8	137	0.25
[42]	$\mathbb{F}_p$	52,000	50	1,729	34,600	1,799
[50]**	$\mathbb{F}_p$	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.

A comparative with state-of-art in software implementations for computing bilinear pairings is presented in table 4.10. Software implementations are in fact flexible implementations that use general purpose 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 purpose 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 of the computations of any Pairing-based protocol to be executed by the general purpose microprocessor.

Table 4.10: Comparative of the proposed architecture with software implementations.

Ref.	Device	Field	Maximum frequency (MHz)	Latency (us)
Barreto-Beuchat	Virtex 6	$\mathbb{F}_{2^m}$	188.9	273
Ronan	Virtex 6	$\mathbb{F}_{2^m}$	188.9	305
[78]	Intel Core i7	$\mathbb{F}_{2^m}$	2,000	517
[78]	Intel Core i7	$\mathbb{F}_{2^m}$	2,000	3,228
[43]	NVidia GTX 480	$\mathbb{F}_{3^m}$	1,401	3,010
[44]	MICAz	$\mathbb{F}_{3^m}$	7.383	2.45×10^6

A parameter almost never reported is the total of memory required by the control of the hardware architectures, because it is not considered a crucial parameter as the area consumption and processing time. In fact, memory optimization was not a target of the proposed design. However, in mobile environments computing resources are very constrained, and for example the amount of memory consumed is critical. In this sense, the proposed instruction set architecture leads to very compact programs. As noted in table 4.11, the work reported in [44] consumes less memory than any other related work. It is observed in table 4.11 also that the memory required by the proposed design in this thesis is half the memory required by [44].

Summary

This chapter presented and discussed the implementation results achieved by the architectural design of this thesis work. First, it was introduced the validation strategy followed in this thesis to ensure the correct functionality of every architectural module. Details in the implementation of the Karatsuba-Ofman multiplier based on linear feedback shift registers was provided because this architecture was

Table 4.11: Comparison of memory consumption of the proposed architecture against related works.

Ref.	Implementation Type	Field	Program Memory (kbits)
Barreto-Beuchat	Flexible	$\mathbb{F}_{2^m}$	5.3
Ronan	Flexible	$\mathbb{F}_{2^m}$	10.3
[38]	Flexible	$\mathbb{F}_{3^m}$	28.1
[34]	Custom	$\mathbb{F}_{3^m}$	24
[50]	ASIP	$\mathbb{F}_p$	32
[44]	Software	$\mathbb{F}_{3^m}$	21.7

a novelty introduced in this thesis work. In order to select the best serial multiplier, the design of the serial multiplier was implemented for different recursion levels in the KOA algorithm. The best implementation results for the serial multiplier obtained in this thesis is compared with the one reported in [30]. Finally, the complete cryptoprocessor was implemented for the FPGA devices Virtex 6 and Virtex 4. Then a comparison with works from the state-of-art was performed.

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.

Chapter 5

Conclusions

This chapter presents final remarks about this thesis work. A brief summary about the work performed is presented, emphasizing the most important points of the architectural design. Then a recapitulation about the main research contributions is depicted. Finally, some guidelines to improve the presented work and research ideas are introduced.

5.1 Objectives review

This thesis work has introduced a novel programmable cryptoprocessor for computing bilinear pairings over elliptic curves defined over binary fields. Bilinear pairings are the mathematical background that brings support to Pairing-based schemes such as Identity-based encryption, short signatures and key agreement schemes. Different to other hardware architectures designed for binary fields, the one presented in this work is able to compute different versions of the pairing algorithm considering different elliptic curves, tower fields, and distortion maps, all these using the same hardware. The proposed cryptoprocessor is the first programmable solution for binary fields implemented in hardware, which additionally is able to compute pairing algorithms 1.9x faster than fastest software implementations reported.

The proposed design followed three main specifications in the design: *i*) the architecture should only support arithmetic in $\mathbb{F}_{2^m}$, *ii*) only operations among registers are supported, and *iii*) an addition is always performed before a multiplication, squaring and square root. Considering these specifications, an instruction set architecture was proposed. This instruction set architecture consists in a group of instructions and the instruction format.

Optimized architectural modules were designed in order to achieve the best possible implementation results. Special attention was given to the field multiplication module because this is the most complex module in terms of area consumption and processing time. A modification to the Karatsuba-Ofman algorithm was proposed, called KOA-LFSR. All architectural modules were integrated into a datapath. A theoretical cost of each module and the full datapath was discussed. A detailed discussion about the programmability was presented.

Architectural modules and the full cryptoprocessor were validated using Xilinx ISim as simulation environment. The cryptoprocessor was specially validated using values taken from a Identity-Based Cryptographic scheme, resulting in a successful validation. The implementation of the KOA-LFSR targeted several finite fields used in cryptographic applications. Results of the KOA-LFSR confirm the theoretical improvement of this modification to the KOA algorithm. Implementation results of the cryptoprocessor show that the proposed design requires a competitive amount of resources compared with related works, requiring in average the 90% of the hardware required by related works. In addition, the processing time is in average 28x shorter than the one achieved by flexible architectures and almost as good as the custom architectures of the state-of-art, in average 2.5x slower. The compact instruction format allows smaller programs than related works, therefore it consumes less than the half of memory required by other works.

The thesis objectives stated in section 1.5 were thoroughly accomplished. At the end of this thesis a programmable architecture for computing bilinear pairings was designed and implemented. It is worth to remark that bilinear pairings are the mathematical background for Identity-based cryptography. The design focused a binary finite field of size $m = 1223$, which is the size required to achieve a security level of 128 bits. The design methodology was the optimization of arithmetic modules and the integration of them into a programmable datapath.

The first specific objective was achieved through literature review and theoretical analysis. As a result, it was observed that there is no standard or protocol for Pairing-based Cryptographic schemes, in fact the development and improvement of current schemes is still under development. Parameters like the elliptic curve, the tower field, and the distortion map may vary across different implementation without affecting the security of the system; so a flexible solution able to support such changes is desired. The lack of a flexible solution for binary fields motivated the design and implementation of an efficient and flexible cryptoprocessor for Pairing-based cryptography.

The second specific objective was completed through the definition of a programmable cryptoprocessor. A programmable architecture was chosen because it fits better to the desired flexibility of a software along with the efficiency of a custom architecture. As evidence, this thesis proposed a set of architectural specifications and the Instruction Set Architecture for a programmable solution.

For the third specific objective was also completed. In this sense, this thesis design and implemented a set of arithmetic modules, among them a novel finite field multiplier was proposed. A final datapath was designed and validated, which is able to compute bilinear pairings with different parameters and using different versions of the pairing algorithm.

Finally, the last specific objective was achieved in the validation stage. A pairing-based scheme was implemented in software, and all pairing values were used to verify the correct functionality of the proposed cryptoprocessor.

5.2 Summary of contributions

This thesis works has two main research contributions:

First, a programmable architecture for computing bilinear pairings, the most time consuming and core operation for Identity-based cryptography, has been proposed. This architecture is the first of this kind which has targeted binary fields. This architecture computes bilinear pairings faster than state-of-art software implementations, and also has a very competitive area/time compared with custom and fixed architectures. In this way, the architecture preserves the speed up of specialized architectures and the flexibility of software implementations. Preliminary results of this contribution were reported in a conference paper, submitted and accepted to the 8th International Workshop on Reconfigurable Communication-centric Systems-on-Chip[1].

- Eduardo Cuevas-Farfán, Miguel Morales-Sandoval, René Cumplido, Claudia Feregrino-Uribe, Ignacio Algredo-Badillo, "A programmable FPGA-based

[1]http://www.recosoc.org/

cryptoprocessor for bilinear pairings over $\mathbb{F}_{2^m}$," Presented at 8th International Workshop on Reconfigurable Communication-centric Systems-on-Chip, Darmstadt, Germany, July 2013.

Second, this thesis proposed a novel $\mathbb{F}_{2^m}$ multiplier called KOA-LFSR. The proposed approach integrates the modular reduction within the polynomial multiplication step of the Karatsuba-Ofman algorithm. An array of Linear Feedback Shift Registers (LFSR) connected in cascade are used to implement the reduction step. This modification leads to a theoretical and practical improvement of the original Karatsuba-Ofman algorithm. The design and implementation results of this novel KOA-LFSR multiplier were reported in a journal article, which is now published:

- E. Cuevas-Farfan, 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. 310, 2013.

5.3 Future work

Further research looking for architectural improvements can be pursued. A thorough architectural optimization process may be implemented in order to improve the maximum clock frequency, being the pipelining technique one approach to achieve that goal. An architectural change may be done, following the multiplication strategy proposed in [31]. Resource consumption should be analyzed in order to reach a smaller design, especially in those architectural modules where more area consumption is identified. In this sense, a different approach for organizing the registers may be proposed in order to use less multiplexers. The cryptoprocessor could be considered within a whole system for computing not just pairing algorithms but Pairing-based schemes. A communication interface to send/receive the operands should implemented. For this purpose an option 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. Finally, in order to accelerate the codification for the cryptoprocessor, a compiler could be created in order to translate faster instructions to its binary representation.

Additionally, there are some research ideas that may be explored in more detail in future works. These ideas include the study of a programmable and field independent coprocessor, such that the architecture reaches total flexibility. Research in other cryptographic modules, like Random Number Generators and Hash Functions, focused for Pairing-based schemes may be driven. Finally, research on architectures resistant to side-channel attacks could be pursued in order to obtain safer and relivable architectures.

List of Figures

List of Tables

Bibliography

[1] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," *Advances in Cryptology*, vol. 196, pp. 47–53, 1985.

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

[3] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for Key Management Part 1 : General," *Special Publication (SP) 800-57, NIST*, no. July, 2012. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf

[4] B. Schneier, *Applied Cryptography*, 2nd ed. Wiley, 1996.

[5] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

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

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

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

[9] E. Barker and A. Roginsky, "Transitions : Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths," *Special Publication (SP) 800-131A, NIST*, no. January, 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf

[10] K. G. Paterson and G. Price, “A comparison between traditional public key infrastructures and identity-based cryptography,” *Information Security Technical Report*, vol. 8, no. 3, pp. 57–72, Jul. 2003.

[11] D. R. Kuhn, V. C. Hu, W. T. Polk, and S.-J. Chang, “Introduction to Public key Technolgy and the Federal PKI Infraestructure,” *Special Publication (SP) 800-32, NIST*, no. February, 2001. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-32/sp800-32.pdf

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

[13] C. Gentry and A. Silverberg, “Hierarchical ID-Based Cryptography,” in *Advances in Cryptology - ASIACRYPT 2002*, Y. Zheng, Ed. Springer Berlin / Heidelberg, 2002, pp. 548–566.

[14] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, “Public Key Encryption with Keyword Search,” in *Advances in Cryptology - EUROCRYPT 2004*. Springer Berlin / Heidelberg, 2004, pp. 506–522.

[15] K. Paterson, “ID-based signatures from pairings on elliptic curves,” *Electronics Letters*, vol. 38, no. 18, p. 1025, 2002.

[16] S. Mishra, R. A. Sahu, S. Padhye, and R. S. Yadav, “Efficient ID-Based Multi-proxy Signature Scheme from Bilinear Pairing Based on k-plus Problem,” in *Integrated Computing Technology.* Springer Berlin / Heidelberg, 2011, pp. 113–122.

[17] R. A. Sahu and S. Padhye, “New ID-Based Proxy Multi-signature from Pairings,” in *Informatics Engineering and Information Science.* Springer Berlin / Heidelberg, 2011, pp. 174–184.

[18] K. Y. Choi, J. Y. Hwang, and D. H. Lee, “Efficient ID-based Group Key Agreement with Bilinear Maps,” in *Public Key Cryptography - PKC 2004*. Springer Berlin / Heidelberg, 2004, pp. 130–144.

[19] H. M. Lee, K. J. Ha, and K. M. Ku, “ID-based Multi-party Authenticated Key Agreement Protocols from Multilinear Forms,” in *Information Security - ISC 2005.* Springer Berlin / Heidelberg, 2005, pp. 104–117.

[20] 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.

[21] 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.

[22] "Digital Signature Standard (DSS)," *Federal Information Processing Standars Publication (FIPS) 186-3, NIST*, 2009. [Online]. Available: http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[23] 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.

[24] 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.

[25] 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

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[36] 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.

[37] 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.

[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] 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.

[40] F. Vercauteren, "Optimal Pairings," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 455–461, Jan. 2010.

[41] 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.

[42] 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.

[43] 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

[44] 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.

[45] V. S. Miller, "Short Programs for functions on Curves," *UNPUBLISHED*, pp. 1–7, 1986. [Online]. Available: http://crypto.stanford.edu/miller/miller.pdf

[46] S. Duquesne, "RNS arithmetic in $\mathbb{F}_{p^k}$ and application to fast pairing computation," *Journal of Mathematical Cryptology*, vol. 5, no. 1, Jan. 2011.

[47] J. C. Bajard, S. Duquesne, M. Ercegovac, and N. Meloni, "Residue systems efficiency for modular products summation: application to elliptic curves cryptography," in *Proc. SPIE 6313, Advanced Signal Processing Algorithms, Architectures, and Implementations XVI*, F. T. Luk, Ed. SPIE, Aug. 2006.

[48] 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.

[49] 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.

[50] 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.

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

[52] 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.

[53] A. Karatsuba and Y. Ofman, “Multiplication of Multidigit Numbers on Automata,” *Soviet Physics-Doklady*, vol. 7, no. 7, pp. 595—-596, 1963.

[54] M. M. Knezevic, F. Vercauteren, and I. Verbauwhede, “Faster Interleaved Modular Multiplication Based on Barrett and Montgomery Reduction Methods,” *IEEE Transactions on Computers*, vol. 59, no. 12, pp. 1715–1721, Dec. 2010.

[55] C. K. Koc, “Montgomery reduction with even modulus,” *IEE Proceedings of Computers and Digital Techniques*, vol. 141, no. 2, pp. 314–316, 2010.

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

[57] 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.

[58] 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.

[59] J. M. Pollard, “Monte Carlo Methods for Index Computation ($\mod p$),” *Mathematics of Computation*, vol. 32, no. 143, pp. 918–924, 1978.

[60] D. Coppersmith, “Fast evaluation of logarithms in fields of characteristic two,” *IEEE Transactions on Information Theory*, vol. 30, no. 4, pp. 587–594, Jul. 1984.

[61] D. M. Gordon, “Discrete Logarithms in $GF(P)$ Using the Number Field Sieve,” *SIAM Journal on Discrete Mathematics*, vol. 6, no. 1, pp. 124–138, Feb. 1993.

[62] L. M. Adleman and M.-D. a. Huang, “Function Field Sieve Method for Discrete Logarithms over Finite Fields,” *Information and Computation*, vol. 151, no. 1-2, pp. 5–16, May 1999.

[63] 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.

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

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

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

[67] 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.

[68] 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

[69] 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.

[70] 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.

[71] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba Algorithm for Efficient Implementations," *Cryptology ePrint Archive*, vol. 2006/224, 2006.

[72] 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.

[73] 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

[74] A. B. El-sisi, S. M. Shohdy, and N. Ismail, "Reconfigurable Implementation of Karatsuba Multiplier for Galois Field in Elliptic Curves," *Novel Algorithms and Techniques in Telecommunications and Networking*, pp. 97–92, 2010.

[75] G. Zhou, H. Michalik, and L. Hinsenkamp, "Improving Throughput of AES-GCM with Pipelined Karatsuba Multipliers on FPGAs," *Reconfigurable Computing: Architectures, Tools and Applications*, vol. 5453, pp. 193–203, 2009.

[76] M. Machhout, M. Zeghid, W. El Hadj Youssef, B. Bouallegue, A. Baganne, and R. Tourki, "Efficient Large Numbers Karatsuba-Ofman Multiplier Designs for Embedded Systems," in *Conference of the World Academy of Science Engineering and Technology 28*. WASET, 2009, pp. 992–1001.

[77] W. El hadj youssef, M. Machhout, M. Zeghid, B. Bouallegue, and R. Tourki, "Efficient hardware architecture of recursive Karatsuba-Ofman multiplier," in *2008 3rd International Conference on Design and Technology of Integrated Systems in Nanoscale Era*. IEEE, Mar. 2008, pp. 1–6.

[78] 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.