FPGA Accelerated Tate Pairing Based Cryptosystems over Binary Fields

Chang Shu*, Soonhak Kwon**, and Kris Gaj*
George Mason University, Fairfax, Virginia, USA*
Sungkyunkwan University, Suwon, Korea**
cshu@gmu.edu, shkwon@skku.edu, kgaj@gmu.edu

Abstract

Though the implementation of the Tate pairing is commonly believed to be computationally more intensive than other cryptographic operations, such as ECC point multiplication, there has been a substantial progress in speeding up the Tate pairing computations. Because of their inherent parallelism, the existing Tate pairing algorithms are very suitable for hardware implementation aimed at achieving a high operation speed. Supersingular elliptic curves over binary fields are good candidates for hardware implementation due to their simple underlying algorithms and binary arithmetic. In this paper we propose efficient Tate pairing implementations over binary fields $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$ via FPGA. Though our field sizes are larger than those used in earlier architectures with the same security strength based on cubic elliptic curves or binary hyperelliptic curves, fewer multiplications in the underlying field are required, so that the computational latency for one pairing can be reduced. As a result, our pairing accelerators implemented via FPGA can run 15-to-25 times faster than other FPGA realizations at the same level of security strength, and at the same time achieve lower product of latency by area.

Keywords: Tate pairing, elliptic curve, FPGA.

1 Introduction

Pairing based cryptography is an important new branch of public key cryptography. This is because bilinear pairings allow identity-based cryptographic schemes that were not readily available using conventional techniques other than parings. The idea of identity-based cryptography is due to Shamir [15]. After the pioneering works of Boneh and Franklin [17] and Sakai et al. [18], using pairing techniques for identity based cryptography, numerous other works on pairing based protocols are proposed.

Though the implementation of the pairing based cryptography was commonly believed to be slow because of the heavy cost of Tate pairing computations, there has been much progress by the works of Galbraith et al. [12], Barreto et al. [8], Granger et al. [5, 6], and Duursma and Lee [7]. All their optimizations involve intricate techniques of deleting unnecessary operations from Miller's algorithm [13]. Especially the work of Duursma and Lee [7] promoted the study of efficient pairing computations of elliptic curves over $\mathbb{F}_{3^m}$, which has embedding degree 6. Subsequently their idea was applied to the case of binary fields by Kwon [11], and generalized to encompass different characteristics and also hyperelliptic cases by Barreto et al. [9] using eta pairing technique.

To the authors' knowledge, the first known hardware implementations of pairing computations are the works of Kerins et al. [1, 2] and of Grabher and Page [3], and they all considered Duursma-Lee algorithm for cubic fields. Recently Ronans et al. [4] proposed dedicated hardware for computing eta pairing of hyperelliptic curve based on the algorithm in [9]. Though the cubic elliptic and binary hyperelliptic cases have strong merits of having high embedding degrees such as 6 and 12, they also have some drawbacks for hardware implementations. First, the arithmetic circuits over $\mathbb{F}_{3^m}$ are more costly and complex than those for arithmetic over $\mathbb{F}_{2^m}$, and the binary fields can exploit the full power of gate capacity. Second, the binary hyperelliptic case [4] uses more complicated arithmetic operations than binary elliptic case and the resulting data path may not be so ideal for a hardware implementation.

In this paper, we propose a low-latency hardware accelerator for Tate pairing computations on supersingular elliptic curves over binary fields. Our implementation is based on the algorithms in [9] and [11]. Though the elliptic curves have rather low embedding degree 4, the arithmetic operations in the algorithms are simple and easy to parallelize. We have developed a compact design for the extension field multiplier by sharing XOR array among several multipliers in case that one of the two operands is the same for those field multiplications. The controller is realized by hardwired logic. The method to simplify the datapath for the final exponentiation is proposed. The optimal choices of parameters such as digit sizes of multipliers provide further optimization for the design. Consequently, our pairing accelerator can run 15-to-25 times as fast as the ones published in [1, 3, 4] at the same level of security strength with lower product of latency by area.

2 Overview of Tate Pairing Computations

Let E be an elliptic curve over a finite field $\mathbb{F}_q$, where q is a power of a prime. Let $l > 0$ be an integer relatively prime to q, and let k be the least positive integer satisfying $q^k \equiv 1 \pmod{l}$. Such k is called a security multiplier or embedding degree of E and l. Let $E(\mathbb{F}_q)[l] = \{P \in E(\mathbb{F}_q) | lP = O\}$.

A divisor D on E is a formal (finite) sum of the points P on the curve, $D = \sum n_p(P)$, $n_p \in \mathbb{Z}$. We call D a degree 0 divisor if $\sum n_p = 0$. A principal divisor is a divisor of the form $(f) = \sum n_p(P)$, where f is a rational function on E and P is a point of E with n_P the order of multiplicity of f at P. One can refer to [14] for elementary introduction to divisor theories. The (reduced) Tate pairing τ_l on the set $E[l]$ is defined as follows.

Definition 1. *Let $P \in E[l](\mathbb{F}_q)$ and $Q \in E[l](\mathbb{F}_{q^k})$. The Tate pairing is a map*

$$\tau_l : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_{q^k})[l] \longrightarrow \mathbb{F}_{q^k}^{\times}/(F_{q^k}^{\times})^l, \quad \text{with} \quad \tau_l(P, Q) = f_P(D_Q)^{\frac{q^k-1}{l}},$$

where f_P is a rational function satisfying $(f_P) = l(P) - l(O)$ and D_Q is a degree 0 divisor equivalent to $(Q) - (O)$ such that D_Q and (f_P) have disjoint supports.

It is well known that τ_l is a non-degenerate bilinear pairing [14]. An effective algorithm for finding a rational function f_P satisfying $(f_P) = l(P) - l(O)$ with $P \in E[l]$ is found by Miller [13]. This Miller's algorithm is further improved by the works of [5, 6, 7, 8, 9, 11, 12].

Let E be a supersingular elliptic curve over $\mathbb{F}_{2^m}$ with $gcd(m, 2) = 1$ defined by

$$E_b : Y^2 + Y = X^3 + X + b, \ b = 0, 1.$$

Then, it is well known that the corresponding elliptic curves have the embedding degrees $k = 4$, and have orders dividing $2^{2m} + 1$. More precisely we have

$$\begin{aligned} |E_b(\mathbb{F}_{2^m})| &= 2^m + 1 + (-1)^b 2^{\frac{m+1}{2}}, \quad \text{if} \quad m \equiv 1, 7 \pmod{8} \\ &= 2^m + 1 - (-1)^b 2^{\frac{m+1}{2}}, \quad \text{if} \quad m \equiv 3, 5 \pmod{8}. \end{aligned}$$

3 Algorithms for Pairing for Supersingular Elliptic Curves over Binary Fields

Inspired by the work of Duursma and Lee [7], a nice formula for the Tate paring computation on supersingular elliptic curve over binary field is given in [9, 11]. Moreover by introducing eta pairing technique, revised version of [9] contains an improved formula which reduces the number of iterations by half. The algorithms in [9, 11] use repeated product of the term $g_{2^iP}(\psi Q)$. Here P, Q are points on E_b and $g_P(X,Y)$ denotes the tangent line at P. That is if $P = (\alpha, \beta)$, then g_P is given by the equation $g_P(x,y) = (\alpha^2+1)x + \beta^2 + b + y$. Also ψ is a distortion map (automorphism) defined by

$$\psi : E_b \longrightarrow E_b, \quad \text{with} \quad \psi(x,y) = (x+s^2, y+sx+t),$$

where $s^2+s+1=0$ and $t^2+t+s=0$. For a point $P=(\alpha,\beta)$ on a supersingular curve, It is straightforward to verify that point doublings follow a nice formula $2^iP = \phi^i(\alpha^{(2i)}, \beta^{(2i)})$, where ϕ is defined as $\phi(x,y) = (x+1, y+x)$ and $\alpha^{(i)}$ denotes $\alpha^{(i)} = \alpha^{2^{i'}}$ with $i' \equiv i \pmod m$ and $i' \geq 0$. Inductively one can show that $\phi^i(x,y) = (x_i, y_i) = (x+i, y+ix+\epsilon_i)$, where

$$\epsilon_i = 0 \quad \text{if } i \equiv 0,1 \pmod 4, \quad \text{and} \quad \epsilon_i = 1 \quad \text{if } i \equiv 2,3 \pmod 4.$$

Thus,

$$g_{2^iP}(x,y) = (\alpha_i^{(2i+1)} + 1)x + \beta_i^{(2i+1)} + b + y \tag{1}$$

where $(\alpha_i^{(j)}, \beta_i^{(j)}) = \phi^i(\alpha^{(j)}, \beta^{(j)})$. Note that $\alpha_i^{(j)} = (\alpha_i)^{(j)} = (\alpha^{(j)})_i$ since the automorphism ϕ and the Frobenius map are commutative to each other. The results in [9, 11] says that we have the Tate pairing $\tau(P,Q)$ as

$$\tau(P,Q) = \left(\prod_{i=0}^{m-1} g_{2^iP}(\psi Q)^{2^{2m-i}} \right)^{2^{2m}-1}.$$

Based on Equation 1 and modifying original algorithm [9, 11] for parallel implementation we obtain Algorithm 1.

Algorithm 1 A modified algorithm from [9, 11] for parallel computation of Tate pairing.

```
Require: $P = (\alpha, \beta), Q = (x, y)$
Ensure: $C = \left(\prod_{i=0}^{m-1} g_{2^iP}(\psi Q)^{2^{2m-i}}\right)^{2^{2m}-1}$
1: $C \leftarrow 1$,
2: $\alpha \leftarrow \alpha^4$,   $\beta \leftarrow \beta^4$,   $u \leftarrow x^2 + y^2 + b + \frac{m-1}{2}$,   $v \leftarrow x^2 + 1$,   $\theta \leftarrow \alpha v$        {Initialize}
3: for $i = 0$ to $m-1$ do
4:     $A \leftarrow \beta + \theta + u + (\alpha + v)s + t$
5:     $C \leftarrow C^2$
6:     $C \leftarrow C \cdot A$
7:     $\alpha \leftarrow \alpha^4$,   $\beta \leftarrow \beta^4$,   $u \leftarrow u + v$,   $v \leftarrow v + 1$,   $\theta \leftarrow \alpha v$
8: end for
9: $C \leftarrow C^{2^{2m}-1}$        {Final exponentiation}
```

By refining eta pairing approach, Barreto et al. [9] successfully reduced the number of loops by half using Equation 2,

$$\tau(P,Q) = \left(\ell(\psi Q) \prod_{i=0}^{\frac{m-1}{2}} g_{2^i P}(\psi Q)^{2^{\frac{m-1}{2}-i}} \right)^{(2^{2m}-1)(2^m \mp 2^{\frac{m+1}{2}}+1)(2^{\frac{m+1}{2}}\pm 1)} \tag{2}$$

where $\ell(X,Y)$ is an equation of line passing $2^{\frac{m+1}{2}}P$ and ϵP with $\epsilon = (-1)^{b+\epsilon_{\frac{m+1}{2}}}$, and it is given as

$$\ell(X,Y) = Y + \beta + b + \epsilon_{\frac{m+1}{2}} + (\alpha + \frac{m-1}{2})(X+\alpha).$$

The corresponding algorithm in accordance with Equation 2 is shown below.

Algorithm 2 A modified algorithm from [9] for parallel computation of Tate pairing.

```
Require: $P = (\alpha,\beta), Q = (x,y)$
Ensure: $C = \left( \ell(\psi Q) \prod_{i=0}^{\frac{m-1}{2}} g_{2^i P}(\psi Q)^{2^{\frac{m-1}{2}-i}} \right)^{MT}$,
        where $MT = (2^{2m}-1)(2^m \mp 2^{\frac{m+1}{2}} + 1)(2^{\frac{m+1}{2}} \pm 1)$
 1: $C \leftarrow 1$
 2: $\alpha \leftarrow \alpha^2 + 1$,  $\beta \leftarrow \beta^2 + 1$,  $u \leftarrow y + b + 1$,  $v \leftarrow x + 1$,  $\theta \leftarrow \alpha v$    {Initialize}
 3: for $i = 0$ to $\frac{m-1}{2}$ do
 4:     $A \leftarrow \beta + \theta + u + (\alpha + v + 1)s + t$
 5:     $C \leftarrow C^2$
 6:     $C \leftarrow C \cdot A$
 7:     if $i < \frac{m-1}{2}$ then
 8:         $\alpha \leftarrow \alpha^4$,  $\beta \leftarrow \beta^4$,  $u \leftarrow u + v + 1$,  $v \leftarrow v + 1$,  $\theta \leftarrow \alpha v$
 9:     end if
10: end for
11: $A \leftarrow A + (\alpha^2 + v + 1) + s$    {Computing $\ell(\psi Q)$}
12: $C \leftarrow C \cdot A$    {Computing $\ell(\psi Q) \prod_{i=0}^{\frac{m-1}{2}} g_{2^i P}(\psi Q)^{2^{\frac{m-1}{2}-i}}$}
13: $C \leftarrow C^{MT}$    {Final exponentiation}
```

Here we computed the product $\ell(\psi Q) = \ell(x + s^2, y + sx + t)$ after the for-loop unlike in the original algorithm [9]. This is possible because the last element $g_{2^{\frac{m-1}{2}}P}(\psi Q)$ of the product in Equation 2 is related with $\ell(\psi Q)$ by Equation 3, where the values of α and β can be recovered after the accumulative multiplication stage without additional memories.

$$\ell(\psi Q) = g_{2^{\frac{m-1}{2}}P}(\psi Q) + \frac{m+1}{2} + \alpha^2 + x + s. \tag{3}$$

After reviewing previous works [1, 3, 4] on FPGA implementations of Tate pairing and analyzing comparable finite fields of equivalent security levels, we choose two finite fields $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$. Detailed reason for choosing these fields is explained in Section 4. Our modified Algorithms 1 and 2 have the following characteristics:

1. They are parallel in the sense that the two crucial operations $C \leftarrow C^2 A$ and $\theta \leftarrow \alpha v$ can be done in parallel.

2. We use a polynomial basis for our implementation of the above algorithms since a polynomial basis has an advantage over a normal basis for computing multiplications, even though a normal basis has a simpler squaring and square root operations.

3. We do not compute square root as in the original algorithms, because in the pentanomial case with $m = 283$, we found that square root operation in hardware is rather complicated unlike for the trinomial case, where square root operation is as fast as squaring [10, 16].

4 Choice of Underlying Fields

The following is the table for applicable curves (having large prime order subgroups) with corresponding MOV security levels. Cofactor means that the order of the given curve divided by the cofactor is a prime. Cubic elliptic and binary hyperelliptic cases in Table 1 are taken from [6, 9] and the binary elliptic cases are computed using MAPLE. In Table 1, the **bolded** numbers in MOV security are the closest security levels to $\mathbb{F}_q$ with $q \approx 2^{1024}$. At the current state of cryptographic standards, it is reasonable to choose a field for FPGA implementation whose MOV security is comparable to 1024-bit RSA. In the cubic elliptic case, it is the field $\mathbb{F}_{3^{163}}$ that gives the equivalent security level. Due to the nature of a cubic field, in which two bits are necessary to represent an element of $\mathbb{F}_3$, this field requires 326 bits to represent an element of the underlying field $\mathbb{F}_{3^{163}}$ and there is no known FPGA implementations for $\mathbb{F}_{3^{163}}$. Instead the implementation results for the cases $\mathbb{F}_{3^{79}}$ and $\mathbb{F}_{3^{97}}$ can be found in [1, 3]. For binary hyperelliptic case, the field which insures the security level of $\mathbb{F}_q$ with $q \approx 2^{1024}$ is $\mathbb{F}_{2^{103}}$, and its implementation can be found in [4].

Table 1: Applicable curves for cubic elliptic, binary hyperelliptic and binary elliptic cases.

	Fields	Curves	Co-Factors	MOV Security
Cubic Elliptic	$\mathbb{F}_{3^{79}}$	$Y^2 = X^3 - X - 1$	1	750
	$\mathbb{F}_{3^{97}}$	$Y^2 = X^3 - X + 1$	7	**922**
	$\mathbb{F}_{3^{163}}$	$Y^2 = X^3 - X - 1$	1	**1548**
	$\mathbb{F}_{3^{193}}$	$Y^2 = X^3 - X - 1$	1	1830
	$\mathbb{F}_{3^{239}}$	$Y^2 = X^3 - X - 1$	1	2268
	$\mathbb{F}_{3^{353}}$	$Y^2 = X^3 - X - 1$	1	3354
Binary Hyperelliptic	$\mathbb{F}_{2^{79}}$	$Y^2 + Y = X^5 + X^3 + 1$	151681	**948**
	$\mathbb{F}_{2^{103}}$	$Y^2 + Y = X^5 + X^3$	$13 \cdot 1237$	**1236**
	$\mathbb{F}_{2^{127}}$	$Y^2 + Y = X^5 + X^3 + 1$	198168459411337	1524
	$\mathbb{F}_{2^{199}}$	$Y^2 + Y = X^5 + X^3 + 1$	$2389 \cdot 121789$	2388
	$\mathbb{F}_{2^{239}}$	$Y^2 + Y = X^5 + X^3 + 1$	1	2868
	$\mathbb{F}_{2^{313}}$	$Y^2 + Y = X^5 + X^3 + 1$	1	3756
Binary Elliptic	$\mathbb{F}_{2^{239}}$	$Y^2 + Y = X^3 + X + 1$	1	**956**
	$\mathbb{F}_{2^{241}}$	$Y^2 + Y = X^3 + X + 1$	1	**964**
	$\mathbb{F}_{2^{283}}$	$Y^2 + Y = X^3 + X$	5	**1132**
	$\mathbb{F}_{2^{353}}$	$Y^2 + Y = X^3 + X + 1$	1	1412
	$\mathbb{F}_{2^{367}}$	$Y^2 + Y = X^3 + X + 1$	1	1468
	$\mathbb{F}_{2^{379}}$	$Y^2 + Y = X^3 + X + 1$	1	1516
	$\mathbb{F}_{2^{457}}$	$Y^2 + Y = X^3 + X + 1$	1	1828
	$\mathbb{F}_{2^{557}}$	$Y^2 + Y = X^3 + X$	5	2228

Although the arithmetic of paring computations on binary elliptic curves is very simple, there is no known FPGA implementation at this moment, and the reason might be low security multiplier (embedding degree). However since the hardware implementation of the cubic field is not so efficient compared with a binary field, and the binary hyperelliptic curve has complex arithmetic operations for point additions (which make the data path complicated for FPGAs), it is desirable to design an

FPGA circuit for binary elliptic case and compare it with existing architectures. From the previous results on cubic elliptic and binary hyperelliptic cases, we chose two fields $\mathbb{F}_{2^{239}}$ for the comparison with the cubic case [1, 3] and $\mathbb{F}_{2^{283}}$ for the comparison with the binary hyperelliptic case [4].

5 FPGA implementations

In this section, we focus on the FPGA implementations of Tate pairing on supersingular elliptic curves over binary fields. The underlying fields, $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$, are constructed via $f_{239}(x) = x^{239} + x^{36} + 1$ and $f_{283}(x) = x^{283} + x^{12} + x^7 + x^5 + 1$ accordingly. Both Alg. 1 and 2 contain mainly two stages, accumulative multiplication and final exponentiation, in both of which the operations in $\mathbb{F}_{2^{4m}}$ are involved. The best approach is to represent $\mathbb{F}_{2^{4m}}$ as an extension of $\mathbb{F}_{2^m}$ with a convenient basis [19], and work over the smaller field whenever possible. We use the basis $\{1, s, t, st\}$ for $\mathbb{F}_{2^{4m}}$ over $\mathbb{F}_{2^m}$, with $s \in \mathbb{F}_{2^{2m}}$, $t \in \mathbb{F}_{2^{4m}}$ satisfying:

$$s^2 + s + 1 = 0 \text{ and } t^2 + t + s = 0. \tag{4}$$

To obtain a high-efficiency pairing accelerator, one must consider issues such as: algorithm selection, top architecture, parallelism, resource sharing and efficient realization of the underlying field arithmetic.

Algorithm comparison: The most attractive advantage of using Alg. 2 instead of Alg. 1 is that it takes half iterations in the accumulative multiplication stage. However, a lower complexity of the data-path for final exponentiation can be gained when using Alg. 1 considering that its final exponentiation is much simpler than that of Alg. 2. Both algorithms are realized in our experiments.

Top architecture: There are basically two kinds of structures. The first one is the traditional stored-program machine (SPM), which contains three functional units: a processor, a controller, and memory [20]. The processor includes registers, datapaths, control lines and ALU. The controller should be capable of steering data to the proper destination according to the instruction. The memory is used to store instructions and data. To adopt such an architecture, the designer needs to develop ALU according to the operations necessary for pairing, and accordingly build the instruction set. Since the intermediate operands of pairing are data-dependent and most of the field operations can not be completed in a single or small number of clock cycles, it's not suitable to be pipelined. Moreover, in program-directed operations, instructions are synchronously fetched, decoded and executed, which will deter the operation speed of the accelerator of pairing because of the overhead of communication between memory and the processor. Alternatively, the pairing processor can be constructed via a main controller, interconnection networks, register files and ALU. The controller may be designed as a finite state machine (FSM), scheduling the computation tasks, i.e., it can generate the stimulate signals and select signals switching operands for ALU. The intermediate results will be stored in the register files in order to eliminate the overhead of communication between memory and ALU. We adopt the second architecture for our pairing processor, as shown in Fig. 1.

Parallelism and sharing resources: One advantage of hardware implementation is that it supports parallel computations and provides high operation speed as long as multiple operations can be performed at the same time. In the first stage of both algorithms, the computations $C \cdot A$ and $\alpha \cdot v$ can be completed simultaneously. Additionally, in the second stage of both algorithms, by multiplying the conjugates of the elements in extension fields $\mathbb{F}_{2^{4m}}$ and $\mathbb{F}_{2^{2m}}$, the inversion of extension fields can be transformed into one inversion in $\mathbb{F}_{2^m}$ and several multiplications in $\mathbb{F}_{2^m}$, $\mathbb{F}_{2^{2m}}$ and $\mathbb{F}_{2^{4m}}$. We use one special extension field multiplier, namely CA in Fig. 1, to perform the multiplications involved in both stages. The multiplier CA can be optimized to obtain a compact design by sharing some combinational circuits among several multipliers in $\mathbb{F}_{2^m}$ in case of the same

Figure 1: Top architecture of the accelerator of Tate pairing over binary fields.

operand. The multiplier computing $\alpha \cdot v$ in the first stage performs multiplications in $\mathbb{F}_{2^m}$ involved in final exponentiation as well.

Underlying field arithmetic: The underlying field $\mathbb{F}_{2^m}$ is constructed via the low Hamming weight irreducible polynomial, such as a trinomial or a pentanomial, by which reductions become simple. Squarer should not be shared since the multiplexers introduced are more expensive. Multiplier is the most significant component directly determining the performance of the accelerator, so it is imperative to implement it with high efficiency. Linear feedback shift register (LFSRs) structure is adopted in our MSD-serial multipliers. The multiplicative inversion in $\mathbb{F}_{2^m}$ is computed using Itoh-Tsujii algorithm. Since most intensive computations concentrate in the first stage, it requires that the multipliers inside the component CA and the multiplier performing $\alpha \cdot v$ should have large digit sizes to achieve high operation speed. On the other hand, in order to decrease the resource utilization without loss of performance, the multipliers working only at the final exponentiation stage can be relatively slow, with small digit size.

5.1 Design of Arithmetic Logic Unit

In the following section, we briefly review the traditional technique computing squaring over the underlying field. Our main interest is to compute the multiplications $C \cdot A$ efficiently. In particular, we propose a method optimizing individual subfield multiplier to obtain a compact design of the extension field multiplier CA. Furthermore, we present two schemes for the multiplier CA in which a different number of multipliers are used. These two schemes are ported to an FPGA device. The optimal choices are made based on the product of latency by area. Finally, the simplifying technique for the final exponentiation in both algorithms is explained.

5.1.1 Squarer

Squaring an element $a = \sum_{i=0}^{m-1} a_i x^i \in \mathbb{F}_{2^m}$, where $a_i \in \mathbb{F}_2$, is ruled by the equation $a^2 = \sum_{i=0}^{m-1} a_i x^{2i}$. Since the underlying fields are constructed via irreducible trinomials or pentanomials, by replacing x^m with $x^k + 1$ or $x^{k_3} + x^{k_2} + x^{k_1} + 1$, we can get the formulae computing the coefficients of a^2. The circuit complexity in terms of gate count is proportional to m. Further details can be found in [22]. Squaring over the extension fields $\mathbb{F}_{2^{2m}}$ and $\mathbb{F}_{2^{4m}}$ is relative easy and can be decomposed into several squarings in $\mathbb{F}_{2^m}$.

5.1.2 Multiplier

Digit serial multiplier, allowing the trade-off between timing and area, is more suitable for cryptographic applications with large operand sizes. There are two basic algorithms computing multiplications with polynomial basis representation in $\mathbb{F}_{2^m}$, left-to-right and right-to-left. Even though it is claimed in [24] that the second algorithm is superior in term of low power, we find that less registers are needed when using the first algorithm because only partial product needs to be updated in each iteration besides the shift-in digits. However, both partial product and one operand must be updated in each iteration when using the second one. Therefore we adopt the left-to-right algorithm to derive the digit-serial multiplier.

Let $a(x)$ and $b(x)$ be the two operands of the multiplication in $\mathbb{F}_{2^m}$ and let $c(x)$ be the product. Suppose that the shift D bits are from $a(x)$. Let $n = min\{l \mid l \in \mathbb{Z},\ l \geq m, \text{ and } D \mid l\}$. Let $a' = \sum_{i=0}^{n-1} a'_i x^i$, where $a'_i = a_i$ if $0 \leq i \leq m-1$, otherwise $a'_i = 0$. The multiplier contains mainly two parts, XOR-AND arrays for computing $s(x) = \sum_{i=0}^{D-1} a_{n-D+i} x^i b(x) \bmod f_m(x)$ and LFSRs for computing $c(x) \leftarrow c(x) + s(x)$. There are two candidates for the second part, see Fig. 2. The first approach is to compute $x^i b(x) \bmod f_m(x)$ separately and the partial sum is kept in m bits. For the second one, the partial sum is kept in $m + D$ bits before reduction. Even though less XOR gates are used in the second structure for an individual multiplier, these XOR-AND arrays can not be shared among different multipliers in $\mathbb{F}_{2^m}$. Additionally, the wire density is increased significantly if D is chosen large. On the contrary, the first approach is more suitable to construct the extension field multiplier CA in $\mathbb{F}_{2^{4m}}$ considering that the XOR arrays for $x^i b(x) \bmod f_m(x)$ can be shared among different multipliers in $\mathbb{F}_{2^m}$ in case they share one operand, see Equation 6 and Fig. 3. The second advantage of the first structure is its low wire density which makes it easy for placing and routing.

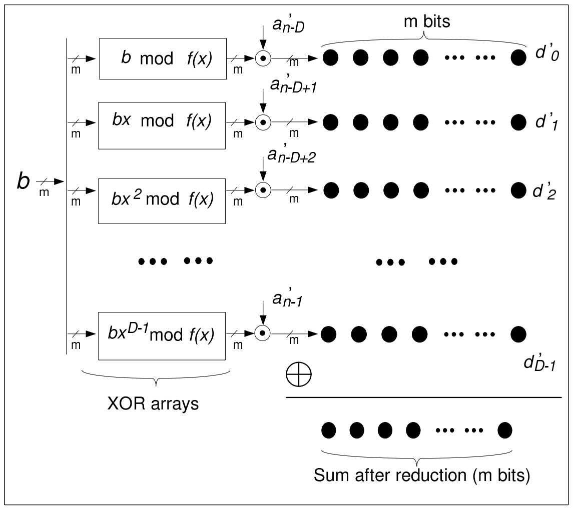
With the basis $\{1, s, t, st\}$ of $\mathbb{F}_{2^{4m}}$ over $\mathbb{F}_{2^m}$, we may write $A = w + zs + et$ where $w, z \in \mathbb{F}_{2^m}$ and $e \in \mathbb{F}_2$. We set $e = 1$ in the accumulative stage and $e = 0$ in the final exponentiation stage. Let $C = c_0 + c_1 s + c_2 t + c_3 st,\ c_i \in \mathbb{F}_{2^m}$, be the partial product of $C \leftarrow C \cdot A$. It it not suitable to apply Karatsuba-Ofman algorithm [23] directly to compute this extension field multiplication recursively since more underlying field multiplications would need to be calculated. However, we can use the same idea to simplify the computations of coefficients c'_1 and c'_3 (see Equation 6).

$$\begin{aligned} C \cdot (w + zs + et) &= (c_0 + c_1 s + c_2 t + c_3 st)(w + zs + et) \\ &= c'_0 + c'_1 s + c'_2 t + c'_3 st, \end{aligned} \tag{5}$$

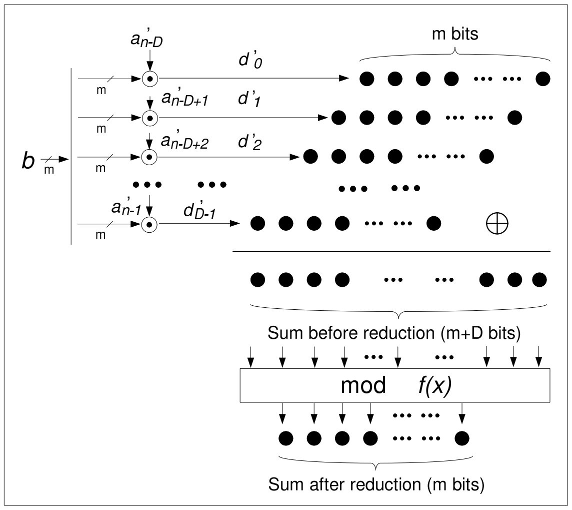
where

$$\begin{aligned} c'_0 &= c_0 w + c_1 z + e c_3 & c'_1 &= (c_0 + c_1)(w + z) + c_0 w + e(c_2 + c_3) \\ c'_2 &= c_2 w + c_3 z + e(c_0 + c_2) & c'_3 &= (c_2 + c_3)(w + z) + c_2 w + e(c_1 + c_3) \end{aligned} \tag{6}$$

Therefore, it takes only 6 $\mathbb{F}_{2^m}$-multiplications for the computation of $C \cdot A$, and all these 6 multiplications can be done simultaneously if 6 multipliers in $\mathbb{F}_{2^m}$ are adopted (See Fig. 4(a)).



(a) Structure 1



(b) Structure 2

Figure 2: Alternative structures for $\sum_{i=0}^{D-1} a_{n-D+i} x^i b(x) \bmod f_m(x)$.

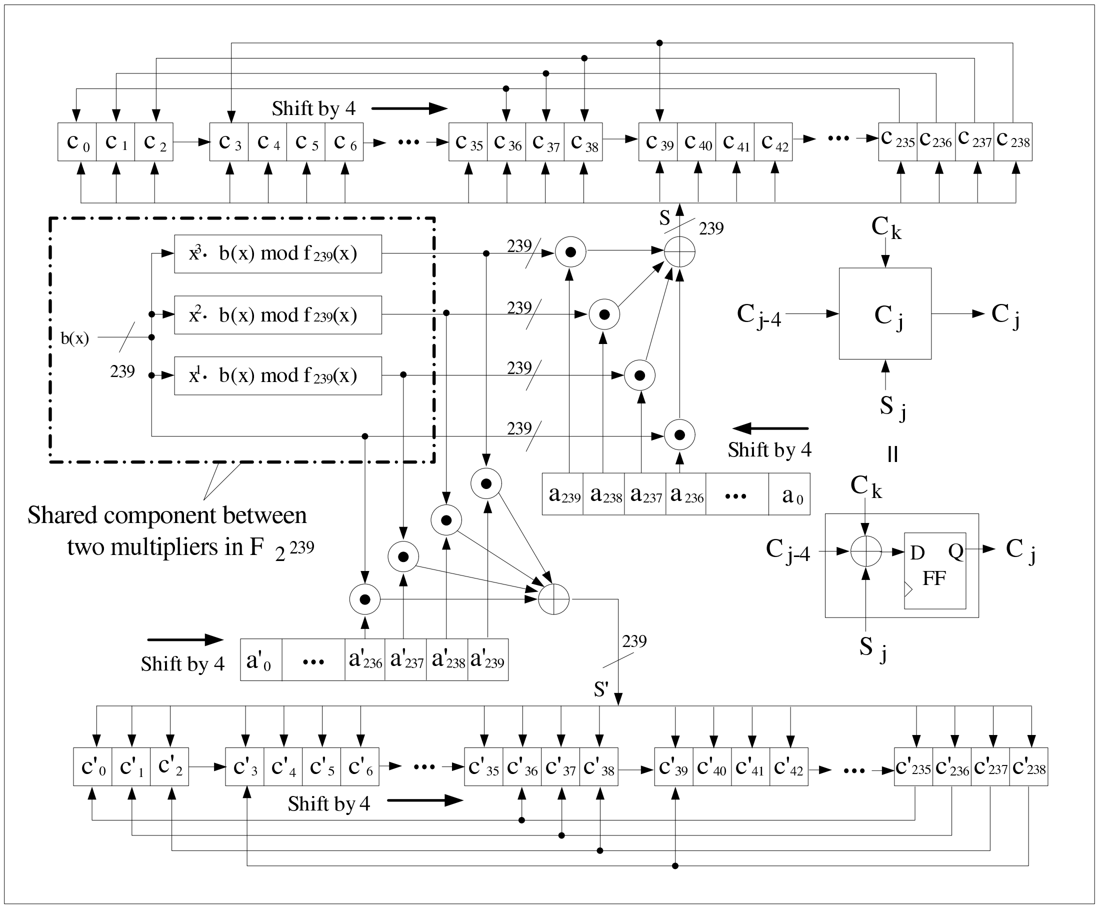


Figure 3: Two digit serial multipliers in $\mathbb{F}_{2^{239}}$ with $D = 4$ sharing the component for $\sum_{i=0}^{3} x^i b(x)$ mod $f_{239}(x)$.

(a) Scheme 1: 6 multipliers adopted

(b) Scheme 2: 3 multipliers adopted

Figure 4: Alternative schemes for CA.

Table 2: FPGA implementation results for the multipliers CA over $\mathbb{F}_{2^{4\times 239}}$ and $\mathbb{F}_{2^{4\times 283}}$.

	Scheme 1 for $\mathbb{F}_{2^{239}}$			Scheme 2 for $\mathbb{F}_{2^{239}}$		
	$D = 4$	$D = 8$	$D = 16$	$D = 4$	$D = 8$	$D = 16$
# FF	3664(4%)	3664(4%)	3664(4%)	2675(3%)	2675(3%)	2675(3%)
# LUT	7799(8%)	13508(15%)	20744(23%))	5580(6%)	8439(9%)	12075(13%)
# CLB slices	4268(9%)	7094(16%)	10722(24%))	3617(8%)	5331(12%)	7536(17%)
Clock period (ns)	9.61	9.98	9.99	9.86	9.98	9.76
Latency (ns)	576.6	299.4	149.9	1182.8	598.8	292.8

	Scheme 1 for $\mathbb{F}_{2^{283}}$			Scheme 2 for $\mathbb{F}_{2^{283}}$		
	$D = 4$	$D = 8$	$D = 16$	$D = 4$	$D = 8$	$D = 16$
# FF	4324(4%)	4348(4%)	4348(4%)	3159(3%)	3171(3%)	3171(3%)
# LUT	9273(10%)	16060(18%)	24729(28%)	6632(7%)	10031(11%)	14428(16%)
# CLB slices	5070(11%)	8416(19%)	12856(29%)	3046(6%)	4483(10%)	6306(14%)
Clock period (ns)	9.84	9.98	9.99	8.89	9.99	9.98
Latency (ns)	698.6	359.3	179.8	1262.4	719.3	359.3

Alternatively, if 3 multipliers are adopted in case of limited resources, the computation of $C \cdot A$ will take two multiplication rounds. In the first round, the first two coefficients of the product, namely c_0' and c_1', will be computed and stored in the registers. In the second round, the other two coefficients, namely c_2' and c_3', will be computed (See Fig. 4(b)). To get the optimal choice in terms of low product of latency by area, both schemes of the special multipliers CA over $\mathbb{F}_{2^{4\times239}}$ and $\mathbb{F}_{2^{4\times283}}$ are ported to the same FPGA device, Xilinx XC2VP100-6FF-1704. The optimization goal is speed instead of area. Second, the hierarchy is not kept so that registers and XOR arrays can be saved considering that several underlying field multiplications share the same operands. Comparisons in terms of timing and area are demonstrated in Table 2.

According to Table 2, Scheme 1 is always superior in terms of low product of latency by area. Hence we adopt 6 multipliers in the extension field multiplier CA for our pairing accelerator so that the multiplication $C \cdot A$ can be completed in one underlying field multiplication round.

5.1.3 Inverter

The inversion in $\mathbb{F}_{2^{4m}}$ involved in the final exponentiation can be transformed into one inversion in $\mathbb{F}_{2^m}$ and several multiplications in $\mathbb{F}_{2^m}$ (the transforming procedure is explained in details in Sec. 5.2). Therefore, we only need to implement one inverter in $\mathbb{F}_{2^m}$. Itoh-Tsujii's algorithm [25] is adopted in our realizations. It takes

$$\lfloor log_2(m-1) \rfloor + HW(m-1) - 1 \tag{7}$$

multiplications in $\mathbb{F}_{2^m}$ to complete one inversion, where $HW(m-1)$ denotes the Hamming weight of $m-1$.

5.2 Simplifying the Data-path for the Final Exponentiation

Let $C = C_1 + C_2 t$ with $C_1, C_2 \in \mathbb{F}_{2^{2m}}$. Using subfield arithmetic and repeated norm calculations, we may compute $C^{2^{2m}-1}$. Letting $C_1^2 + C_1C_2 + C_2^2 s = c_0 + c_1 s$ with $c_0, c_1 \in \mathbb{F}_{2^m}$, a straight calculation shows

$$C^{2^{2m}-1} = \frac{1}{c_0^2 + c_0c_1 + c_1^2} \cdot (c_0 + c_1(s+1))(C_1^2 + C_2^2(1+s) + C_2^2 t),$$

where the total cost of the above computations is 1 $\mathbb{F}_{2^m}$-inversion plus 12 $\mathbb{F}_{2^m}$-multiplications. That is, we need 4 $\mathbb{F}_{2^m}$-multiplications (1 $\mathbb{F}_{2^{2m}}$-multiplication for C_1C_2 and 1 $\mathbb{F}_{2^m}$-multiplication for c_0c_1) for the denominator $c_0^2 + c_0c_1 + c_1^2$, 2 $\mathbb{F}_{2^m}$-multiplications for $\frac{1}{c_0^2+c_0c_1+c_1^2} \cdot (c_0 + c_1(s+1))$, and 6 $\mathbb{F}_{2^m}$-multiplications (2 $\mathbb{F}_{2^{2m}}$-multiplications) which come from the final multiplication by $C_1^2 + C_2^2(1+s) + C_2^2 t$. Therefore only six multiplication units (for parallel processing) are needed here and one can reuse the same arithmetic units as shown in Fig. 5 for multiplications, and thus the additional cost of the final exponentiation is the inversion circuit for computing $\frac{1}{c_0^2+c_0c_1+c_1^2}$.

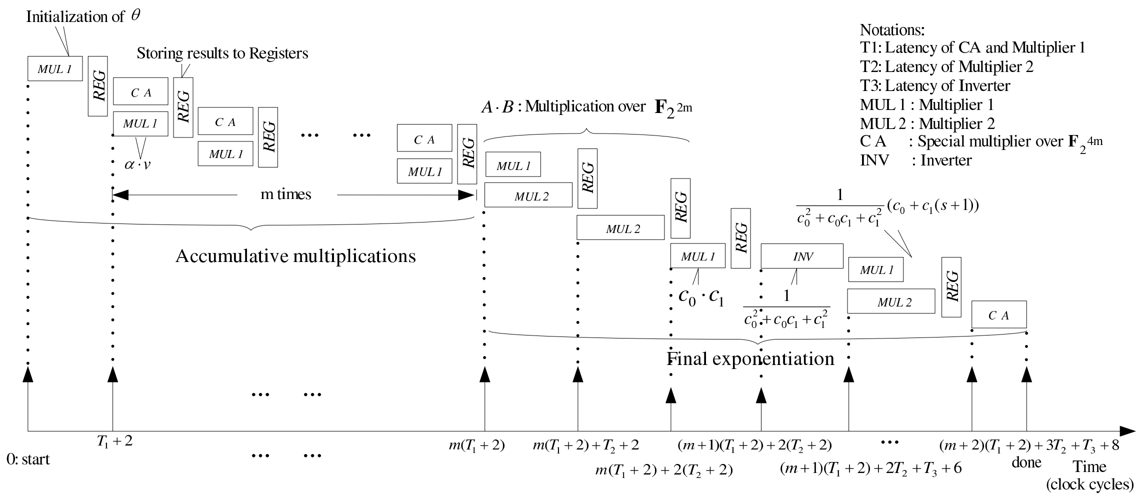
In Algorithm 2, we have more complicated final exponentiation, where $M = \frac{2^{4m}-1}{|E_b(\mathbb{F}_{2^m})|} = \frac{2^{4m}-1}{2^m \pm 2^{\frac{m+1}{2}}+1}$ $= (2^{2m}-1)(2^m \mp 2^{\frac{m+1}{2}} + 1)$ and $T = 2^{\frac{m+1}{2}} \pm 1$. Here we should be cautious about appropriate sign of T. The original definition of T [9] is $T = 2^m - N = \mp 2^{\frac{m+1}{2}} - 1$. However when $T = -2^{\frac{m+1}{2}} - 1$, one computes $(-T)\{(P) - (O))\}$ which gives an inverse of the rational function corresponding to $T\{(P) - (O))\}$, so the exponents should be changed to $-T$ in this case. When $m = 239$ and $b = 1$, we have $|E_b(\mathbb{F}_{2^m})| = 2^{239} - 2^{120} + 1$. In this case we get $MT = (2^{2m}-1)(2^m + 2^{\frac{m+1}{2}} + 1)(2^{\frac{m+1}{2}} - 1)$ with $m = 239$. Also when $m = 283$ and $b = 0$, we have $|E_b(\mathbb{F}_{2^m})| = 2^{283} - 2^{142} + 1$ and thus we have the same form of $MT = (2^{2m}-1)(2^m + 2^{\frac{m+1}{2}} + 1)(2^{\frac{m+1}{2}} - 1)$ with $m = 283$.

Suppose z is in $GF(2^{4m})$ such that $z = w^{2^{2m}-1}$ for some $w \in \mathbb{F}_{2^{4m}}$. Then $z^{2^{2m}+1} = w^{2^{4m}-1} = 1$ and thus we get $z^{-1} = z^{2^{2m}}$. Moreover from $1 = z^{2^{2m}+1} = z^{(2^m+1+2^{\frac{m+1}{2}})(2^m+1-2^{\frac{m+1}{2}})}$, we get $z^{(2^m+1+2^{\frac{m+1}{2}})(2^{\frac{m+1}{2}}-1)} = z^{(2^m+1+2^{\frac{m+1}{2}})2^m}$. Therefore the exponent MT can be interpreted as:

$$\begin{aligned} MT &= (2^{2m}-1)(2^m+2^{\frac{m+1}{2}}+1)(2^{\frac{m+1}{2}}-1) \\ &= (2^{2m}-1)(2^m+2^{\frac{m+1}{2}}+1)2^m \end{aligned} \tag{8}$$

5.3 Parameter Choices for ALU

In Fig. 5, we provide the timing diagram of scheduling computations for pairing according to Algorithm 1. Additions and squarings realized via combinational circuits need not to be taken into account for estimating the latency.



Initialization of θ
Storing results to Registers
MUL 1
REG
C A
$\alpha \cdot v$
MUL 2
INV
$A \cdot B$: Multiplication over $\mathbf{F}_{2^{2m}}$
m times
Accumulative multiplications
Final exponentiation
$c_0 \cdot c_1$
$\frac{1}{c_0^2 + c_0c_1 + c_1^2}$
$\frac{1}{c_0^2 + c_0c_1 + c_1^2}(c_0 + c_1(s+1))$
Notations:
T1: Latency of CA and Multiplier 1
T2: Latency of Multiplier 2
T3: Latency of Inverter
MUL 1 : Multiplier 1
MUL 2 : Multiplier 2
C A : Special multiplier over $\mathbf{F}_{2^{4m}}$
INV : Inverter
0: start
$T_1 + 2$
$m(T_1+2)$
$m(T_1+2)+T_2+2$
$m(T_1+2)+2(T_2+2)$
$(m+1)(T_1+2)+2(T_2+2)$
$(m+1)(T_1+2)+2T_2+T_3+6$
$(m+2)(T_1+2)+3T_2+T_3+8$
done
Time (clock cycles)

Figure 5: Timing diagram of scheduling computations according to Alg. 1.

Let Δ_1 denote the latency for one computation of Tate pairing using Alg. 1 and let T_{clk} denote the clock period. Let D_1 denote the digit size of multipliers inside CA and Multiplier 1; D_2 denote the size of Multiplier 2; and D_3 denote the digit size of multiplier inside the inverter. The notations of T_1, T_2, T_3 are specified in Fig. 5. Then we can estimate the latency for computing one Tate pairing using Alg. 1 as follows:

$$\Delta_1 = (m+2)(T_1+2) + 3T_2 + T_3 + 8 \tag{9}$$

where $T_1 = \lceil m/D_1 \rceil T_{clk}$, $T_2 = \lceil m/D_2 \rceil T_{clk}$. Two extra clock cycles are required for the register read/write operations. In case of $\mathbb{F}_{2^{239}}$, it takes around 239 clock cycles for exponentiation and 12 multiplications in case of computing c^2 each cycle. However the latency for exponentiation needs to be shortened. We compute c^{2^4} in each cycle so that 60 cycles are necessary to complete the exponentiation in the inversion of $\mathbb{F}_{2^{239}}$. Then $T_3 \approx (60 + 12 \cdot \lceil 239/D_3 \rceil)T_{clk}$. Similarly $T_3 \approx (71 + 11 \cdot \lceil 283/D_3 \rceil)T_{clk}$ for $\mathbb{F}_{2^{283}}$. If we choose $D_1 = 16$, $D_2 = 4$ and $D_3 = 8$, the time spent on final exponentiation is 10.3% of accumulative multiplications. However if we choose $D_3 = 4$ and keep the same value of D_1 and D_2, the time for final exponentiation is 23.8% of accumulative multiplication. So we choose $D_1 : D_2 : D_3 = 4 : 1 : 2$ for both $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$.

For Alg. 2, the computation of C^{MT} needs to be additionally taken into account for latency estimation, see Equation 8. We use the component computing C^{2^4} each cycle for the extension field exponentiation, and one extension field multiplication with two operands in $\mathbb{F}_{2^{4m}}$ can be completed by CA in two underlying field multiplication rounds. Then, we use the following equation to estimate the latency of pairing for Alg. 2,

$$\Delta_2 = \frac{m+5}{2}(T_1+2) + 3T_2 + T_3 + T_4 + 8; \qquad T_4 = \frac{m+1}{2}T_{clk} + 4(T_1+2) \tag{10}$$

where T_4 denotes the latency for C^{MT}. We use the same ratio of D_1, D_2 and D_3 as Alg. 1.

5.4 Results and Comparisons with Previous Work

The target device for our implementations is Xilinx XC2VP100-6FF-1704. For Alg. 1, the digit size of multipliers inside CA is chosen as 16 and 32 for both $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$. For Alg. 2, due to the limitation of resources, the digit size D_1 is chosen as 16. The results after placing and routing via Xilinx ISE-7.1 are summarized in Table 3.

Table 3: Performance and cost of Tate pairing accelerators on elliptic curves over $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$ using both algorithms.

		# FF	# LUT	# CLB slices	f (MHz)	Latency (μs)
Alg. 1	$\mathbb{F}_{2^{239}}, D_1 = 16$	10,981 (12%)	34,499 (12%)	18,202 (41%)	100	47
	$\mathbb{F}_{2^{239}}, D_1 = 32$	11,077 (12%)	59,971 (68%)	31,719 (71%)	83	33
	$\mathbb{F}_{2^{283}}, D_1 = 16$	12,995 (14%)	42,997 (48%)	22,726 (51%)	84	76
	$\mathbb{F}_{2^{283}}, D_1 = 32$	13,007 (14%)	72,961 (82%)	37,803 (85%)	72	49
Alg. 2	$\mathbb{F}_{2^{239}}, D_1 = 16$	14,226 (16%)	48,895 (55%)	25,487 (57%)	84	34
	$\mathbb{F}_{2^{283}}, D_1 = 16$	16,563 (18%)	64,845 (73%)	33,252 (75%)	56	70

If the digit sizes are chosen the same for both algorithms, the latency of Alg. 2 is shorter than Alg. 1. However more resources are utilized for Alg. 2 because the multiple squarers in $\mathbb{F}_{2^{4m}}$ are adopted for C^{MT} and more complicated datapath in final exponentiation can not be avoided so that the critical path is longer than Algorithm 1.

Compared with other researchers' works [1, 3, 4], almost at the same level of security strength, our Tate pairing accelerators can run 15-to-25 times as fast as theirs on average. See Table 4, where D denotes the digit size of multipliers working in the accumulative stage. In [1], the pairing accelerator for the elliptic curve over $\mathbb{F}_{3^{97}}$ with $k = 6$ is realized via a larger FPGA device, Xilinx XC2VP125 with 55,616 slices. Karatsuba-Ofman's method [23] is used to construct the multiplier in $\mathbb{F}_{3^{6m}}$. To achieve the full power of parallel computation for such a large extension field multiplication, 18 multipliers in $\mathbb{F}_{3^m}$ are necessary. Due to the limitation of resources, the digit sizes of underlying field multipliers can not be large so they select $D = 4$. The cost is 60% of 55616 slices for the multiplier in $\mathbb{F}_{3^{6\times 97}}$. For comparison it costs only 10,722 slices for our multiplier CA in $\mathbb{F}_{2^{4\times 239}}$ where $D = 16$. No exact results for the whole accelerator of pairing are provided in [1], but it is claimed that 100% of resources are utilized. So the cost is around 55,616 CLB slices. The operation frequency is 10MHz and it takes 850 μs for one pairing. Our pairing accelerator on the elliptic curve over $\mathbb{F}_{2^{239}}$ can run 25 times faster.

In [3], a smaller device, Xilinx XC2VP4FF672-6 with 4928 slices is chosen as the target device. The cubic field arithmetic is realized as a kind of co-processor via FPGA, which is controlled by a more general purpose processor, i.e., the top architecture is a stored-program machine (SPM) as described previously. Their field arithmetic co-processor contains only one polynomial basis multiplier with digit size $D = 4$ so that multiplications are performed sequentially, i.e., at least 18 subfield

Table 4: Comparisons with Peer researchers' FPGA implementations with almost the same security strength.

	Curves	Underlying fields	MOV Security	Controller	# CLB slices	Digit size D	f(MHz)	Latency (μs)
[1]	Elliptic	$\mathbb{F}_{3^{97}}$	922	Hardwired logic	55,616	4	10	850
[3]	Elliptic	$\mathbb{F}_{3^{97}}$	922	Microprocessor	4,481	4	150	432
[4]	Hyperelliptic	$\mathbb{F}_{2^{103}}$	1236	Hardwired logic	43,986	16	32	749
Alg. 2	Elliptic	$\mathbb{F}_{2^{239}}$	956	Hardwired logic	25,287	16	84	34
Alg. 1	Elliptic	$\mathbb{F}_{2^{283}}$	1132	Hardwired logic	37,803	32	72	49

multiplication rounds are necessary for each iteration of the accumulative multiplication. In contrast, our processor support parallel computation of multiplications in $\mathbb{F}_{2^m}$ and the digit sizes we used are much larger than the ones in [3]. The latency for one pairing of our accelerator over $\mathbb{F}_{2^{239}}$ is only 34 μs. The latency of the design by Grabher et al. [3] is at least 432 μs. At the same time, our resource utilization is only 5 times larger as theirs.

Ronan et al. [4] have implemented Tate pairing accelerator on the hyperelliptic curve over binary field using the device Xilinx XC2VP125 which is tha same as the one used in [1]. They chose a smaller underlying field $\mathbb{F}_{2^{103}}$ and a larger embedding degree $k = 12$. However, in each iteration of accumulative multiplication stage, 16 multiplications in $\mathbb{F}_{2^m}$ and 1 multiplication in $\mathbb{F}_{2^{12m}}$ need to be computed. These 16 subfield multiplications must be completed before the accumulative multiplication in $\mathbb{F}_{2^{12m}}$. Note that, in [4], one multiplication in $\mathbb{F}_{2^{12m}}$ is realized as 54 multiplications in $\mathbb{F}_{2^m}$ using Karatsuba's method. In our case, we choose a larger underlying field $\mathbb{F}_{2^{283}}$ and smaller embedding degree $k = 4$ so that the computations of accumulative multiplications becomes much simpler and only 7 multiplications in $\mathbb{F}_{2^m}$ are involved each round and these multiplications can be performed in parallel. The shortest latency for one pairing of Ronan et al's is 749 μs and 50,893 slices are used. Our accelerator can run 15.6 times as fast as theirs whereas the resource utilization is only 37,803 slices, see Table 4.

6 Conclusions

We investigated the FPGA implementations of Tate pairing on supersingular elliptic curves over binary fields with embedding degree $k = 4$. We adopted two top algorithms computing pairing by which full power of parallel computations can be exploited with less resource utilization than in designs of other researchers. We performed security analysis for different curves over binary and cubic fields by which we can choose two binary fields $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$ for our experiments to achieve almost the same security strength as others. Besides the superiority of top algorithms, we also proposed an optimization method to obtain a compact design of the extension field multiplier CA by sharing some combinational circuits among several individual subfield multipliers in $\mathbb{F}_{2^m}$ in case of one shared operand. Furthermore we compared two schemes with a different number of multipliers in $\mathbb{F}_{2^m}$ for CA to get the optimal choice. The controller is implemented via hardwired logic to eliminate the overhead of instruction fetching and decoding, particularly for the simple operations such as additions and squarings. The technique simplifying the final exponentiations for both algorithms are addressed. Finally the implementations are further optimized by optimal parameter choices for ALU. Compared with other researchers' work, our pairing processors are more efficient in terms of the product of latency by area. In particular, our accelerators can outperform earlier designs by a factor of 15-to-25 in terms of the total execution time.

References

[1] T. Kerins, W.P. Marnane, E.M. Popovici, and P.S.L.M. Barreto, “Efficient hardware for the Tate pairing calculation in charateristic three”, *CHES 2005, Lecture Notes in Computer Science*, vol. 3659, pp. 412–426, 2005.

[2] T. Kerins, E.M. Popovici, and W.P. Marnane, “Algorithms and architectures for use in FPGA implementations of Identity Based Encryption Schemes”, In *Field Programmable Logic and Applications - FPL 2004, Lecture Notes in Computer Science*, vol. 3203, pp. 74-83, 2004.

[3] P. Grabher and D. Page, “Hardware acceleration of the Tate pairing in charateristic three”, *CHES 2005, Lecture Notes in Computer Science*, vol. 3659, pp. 398 - 411.

[4] R. Ronan, C.O. Eigeartaigh, C. Murphy, M. Scott, T. Kerins, and W.P. Marnane, “A dedicated processor for the Eta pairing”, preprint *available at* http://eprint.iacr.org/2005/330.pdf.

[5] R. Granger, D. Page, and M. Stam, “Hardware and software normal basis arithmetic for pairing based cryptography in characteristic three”, *IEEE Trans. on Computers,* vol. 54, pp. 852–860, 2005.

[6] R. Granger, D. Page, and M. Stam, “On small characteristic algebraic tori in pairing based cryptography,” preprint *available at* http://eprint.iacr.org/2004/132.pdf, 2004.

[7] I. Duursma and H. Lee, “Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$”, *Asiacrypt 2003, Lecture Notes in Computer Science*, vol. 2894, pp. 111–123, 2003.

[8] P. Barreto, H. Kim, B. Lynn, and M. Scott, “Efficient algorithms for pairing based cryptosystems”, *Crypto 2002, Lecture Notes in Computer Science*, vol. 2442, pp. 354–368, 2002.

[9] P. Barreto, S. Galbraith, C. O’hEigeartaigh, and M. Scott, “Efficient pairing computation on supersingular abelian varieties,” preprint *available at* http://eprint.iacr.org/2004/375.pdf, 2004.

[10] P. Barreto, “A note on efficient computation of cube roots in characteristic 3,” preprint *available at* http://eprint.iacr.org/2004/305.pdf, 2004.

[11] S. Kwon, “Efficient Tate pairing computation for supersingular elliptic curves over binary fields,” preprint *available at* http://eprint.iacr.org/2004/303.pdf, 2004.

[12] S. Galbraith, K. Harrison, and D. Soldera, “Implementing the Tate pairing,” *ANTS 2002, Lecture Notes in Computer Science*, vol. 2369, pp. 324–337, 2002.

[13] V. Miller, “Short programs for functions on curves,” *unpublished manuscript*, 1986.

[14] A.J. Menezes, Elliptic Curve Public Key Cryptosystems, Kluwer Academic Publisher, 1993.

[15] A. Shamir, “Identity-based cryptosystems and signature schemes,” *Crypto 1985, Lecture Notes in Computer Science*, vol. 196, pp. 47–53, 1985.

[16] K. Fong, D. Hankerson, J. López, and A. Menezes, “Field inversion and point halving revisited,” *Technical Report CORR 2003-18*, Univ. of Waterloo, 2003.

[17] D. Boneh and M. Franklin, “Identity based encryption from the Weil pairing,” *Crypto 2001, Lecture Notes in Computer Science*, vol. 2139, pp. 213–229, 2001.

[18] R. Sakai, K. Ohgishi, and M. Kasahara, “Cryptosystems based on pairing,” *SICS 2000, Symposium on Cryptography and Information Security*, pp. 26–28, 2000.

[19] I.F. Blake, G. Seroussi, and N.G. Smart, Advances in elliptic curve cryptography, Cambridge University Press, 2005.

[20] M.D. Ciletti, Advanced digital design with the Verilog HDL, Prentice Hall, 2004.

[21] T. Wollinger, “Software and Hardware Implementation of Hyperelliptic Curve Cryptography”, Ph. D Thesis, Bochum : Europäischer University, 2004.

[22] G. Orlando and C. Paar, “A high-performance reconfigurable elliptic curve processor for $GF(2^m)$”, *CHES 2000*, vol. 1965, pp. 41-56, 2000.

[23] A. Karatsuba and Y. Ofman, “Multiplication on many-digit numbers by automatic computers”, *Translation in Physics-Doklady*, vol 7, pp. 595-596, 1963.

[24] L. Song and K.K. Parhi, “Efficient Finite Field Serial/Parallel Multiplication”, *Proceedings of International Conference on Application Specific Systems, Achitectures and Processors - ASAP’96*, pp. 72-82, 1996.

[25] T. Itoh and S. Tsujii, “A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases”, *Information and Computation*, vol. 78, pp. 171-177, 1988.