FPGA Accelerated Tate Pairing Based Cryptosystems over Binary Fields

Chang Shu [#1], Soonhak Kwon [*2], Kris Gaj [#3]

[#] *Department of Electrical & Computer Engineering, George Mason University*
4400 University Drive, Fairfax, VA 22030-4444, USA
[1] cshu@gmu.edu
[3] kgaj@gmu.edu

[*] *Department of Mathematics, Sungkyunkwan University*
Suwon, Korea
[2] shkwon@skku.edu

***Abstract*— Tate pairing based cryptosystems have recently emerged as an alternative to traditional public key cryptosystems because of their ability to be used in multi-party identity-based key management schemes. Due to the inherent parallelism of the existing pairing algorithms, high performance can be achieved via hardware realizations. Three schemes for Tate pairing computations have been proposed in the literature: cubic elliptic, binary elliptic, and binary hyperelliptic. For our implementation we have chosen the binary elliptic case because of the simple underlying algorithms and efficient binary arithmetic. In this paper, we propose a new FPGA-based architecture of the Tate pairing-based computation over the binary fields $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$. Even though our field sizes are larger than in the architectures based on cubic elliptic curves or binary hyperelliptic curves with the same security strength, nevertheless fewer multiplications in the underlying field need to performed. As a result, the computational latency for a pairing computation has been reduced, and our implementation runs 10-to-20 times faster than the equivalent implementations of other pairing-based schemes at the same level of security strength. At the same time, an improvement in the product of latency by area by a factor between 12 and 46 for an equivalent type of implementation has been achieved.**

I. Introduction

Pairing based cryptography has become a subject of active research recently because it is the basis of a new class of public key cryptosystems called identity based cryptosystems. In these cryptosystems, a sender can derive a public key of a receiver directly from an ID of the receiver, without the need for any additional information. The basic idea of identity based cryptosystems was proposed by Shamir [14]. Applying pairing techniques to identity based cryptography was suggested first by Boneh and Franklin [16], and then extended by Sakai et al. [17].

Initially, the implementations of pairing based cryptosystems were commonly believed to be slow because of the heavy cost of underlying computations. A significant progress in this area has been accomplished by the works of Galbraith et al. [11], Barreto et al. [8], Granger et. al. [5], [6], and Duursma and Lee [7]. The optimizations introduced by these authors involved delicate techniques of deleting unnecessary operations from the Miller's algorithm [12]. In particular, the work of Duursma and Lee [7] promoted the study of efficient pairing computations for elliptic curves over Galois fields of characteristic three, $\mathbb{F}_{3^m}$. Subsequently, their idea was applied to the case of binary fields in [10], and generalized by Barreto et al. [9] to encompass different characteristics of the underlying field, as well as computations over hyperelliptic curves.

To the authors' knowledge, Kerins et al. [1], [2], and Grabher and Page [3] were the first to report hardware implementations of pairing based cryptosystems. Both publications considered Duursma-Lee algorithm for elliptic curves over cubic fields. Recently, Ronans et al. [4] proposed the dedicated hardware for computing pairing on hyperelliptic curves using the algorithm introduced in [9].

Though the cubic elliptic and binary hyperelliptic cases have strong merits with the high security they offer for relatively low operand sizes, they also have some drawbacks for hardware implementations. First, the arithmetic circuits over cubic fields, $\mathbb{F}_{3^m}$, are more complex and costly in terms of the area and power compared to the circuits for computations over $\mathbb{F}_{2^m}$. Secondly, the binary hyperelliptic case [4] involves more complicated higher-level operations than the binary elliptic case, and the data path may be very complex for hardware implementations.

In this paper, we propose a low latency hardware accelerator for the Tate pairing-based cryptosystems on supersingular elliptic curves over binary fields. We choose FPGAs as our target devices not only because they can serve as fast prototyping platforms, but also because of their reconfigurability. The reconfigurability is crucial in this application because of an early stage of development of the field, the lack of standards, and the constant progress in the cryptanalysis of pairing based cryptosystems, which may affect key sizes, and thus the sizes of all operands, already in the near future.

Our implementation is based on the algorithms presented in [9] and [10]. Even though the binary elliptic curves require relatively long operands (compared to the cubic elliptic and binary hyperelliptic cases), the arithmetic operations in the algorithms we apply are simple and easy to parallelize. We introduce a compact design for the extension field multiplier CA by sharing an XOR array in case that one of the two

operands is the same for the underlying field multiplications. Our controller is realized using hardwired logic. We derive the method to simplify the datapath for the final exponentiation. We also consider the optimal choices of parameters such as digit sizes of multipliers to further optimize the design. Consequently, our pairing accelerator can run 10-to-20 times faster than the ones published in [1], [3], [4], at the same level of security strength with the lower product of latency times area.

This paper is organized as follows. In Section 2, we introduce the mathematical background behind Tate pairing computations. In Section 3, we present two algorithms adopted in our implementations. In Section 4, we discuss the software implementation results for both algorithms and two binary fields selected based on the security analysis. In Section 5, we describe in detail our FPGA-based implementation. Finally, the conclusions from our research are drawn in Section 6.

II. Overview of Tate Pairing Computation

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. Let $E[l] = \{P | lP = O\}$ and $E(\mathbb{F}_q)[l] = \{P \in E(\mathbb{F}_q) | lP = O\}$.

A divisor D on E is a formal (finite) sum of the points P on the curve, $D = \sum n_p(P), \quad n_p \in \mathbb{Z}$. 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 [13] for elementary introduction of 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

$$\tau_l : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_{q^k})[l] \longrightarrow \mathbb{F}_{q^k}^{\times}/(F_{q^k}^{\times})^l$$

with $\tau_l(P, Q) = f_P(D_Q)^{\frac{q^k-1}{l}}$, 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 [13]. An effective algorithm for finding a rational function f_P satisfying $(f_P) = l(P) - l(O)$ with $P \in E[l]$ was proposed by Miller [12]. The Miller's algorithm was further improved by the works of [5], [6], [8], [9], [10], [11].

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

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

III. 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 of supersingular elliptic curve over binary field was proposed in [9], [10]. 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], [10] 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 $t^2 + t + s = 0$.

The results in [9], [10] imply that the Tate pairing $\tau(P, Q)$ is given by

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

Algorithm 1 A modified algorithm from [9], [10] for parallel computation of Tate pairing.

Require: $P = (\alpha, \beta), Q = (x, y)$
Ensure: $C = \tau(P, Q)$
1: $C \leftarrow 1$,
2: $\alpha \leftarrow \alpha^4$, $\beta \leftarrow \beta^4$, $v \leftarrow x^2 + 1$, $\theta \leftarrow \alpha v$,
$u \leftarrow x^2 + y^2 + b + \frac{m-1}{2}$ {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: $\alpha \leftarrow \alpha^4, \quad \beta \leftarrow \beta^4, \quad u \leftarrow u + v,$
$v \leftarrow v + 1, \quad \theta \leftarrow \alpha v$
8: **end for**
9: $C \leftarrow C^{2^{2m}-1}$ {Final exponentiation}

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 $i' \geq 0$. One can show inductively that $\phi^i(x, y) = (x + i, y + ix + \epsilon_i)$, where $\epsilon_i = 0$ if $i \equiv 0, 1 \pmod 4$, and $\epsilon_i = 1$ if $i \equiv 2, 3 \pmod 4$

Thus,

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

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.

By refining eta pairing approach, Barreto et al. [9] successfully reduced the number of loop iterations by half so that they showed

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

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

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

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

Require: $P=(\alpha,\beta), Q=(x,y)$
Ensure: $C = \tau(P,Q)$
1: $C \leftarrow 1$
2: $\alpha \leftarrow \alpha^2+1, \quad \beta \leftarrow \beta^2+1, \quad u \leftarrow y+b+1,$
$v \leftarrow x+1, \quad \theta \leftarrow \alpha v$ {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: $A \leftarrow A + (\alpha^2 + v + 1) + s$
12: $C \leftarrow C \cdot A$
13: $C \leftarrow C^{MT}, \quad MT = (2^{2m}-1)(2^m + 2^{\frac{m+1}{2}} + 1)(2^{\frac{m+1}{2}} - 1)$ {Final exponentiation}

Here we computed the product by $\ell(\psi Q) = \ell(x+s^2, y+sx+t)$ 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 to $\ell(\psi Q)$ by the relation

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

In Alg. 2, the values of α and β can be recovered after the accumulative multiplication stage without additional memories. 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 $\mathbb{F}_{2^{283}}$. Our modified Algorithms 1 and 2 have the following characteristics.

1) They are parallel algorithms in the sense that the two crucial operations $C \leftarrow C^2A$ 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 simple squaring and square root operation.
3) We do not compute square root as in the original algorithms, because in the pentanomial case where $m=283$, we found that square root operation in hardware is rather complicated unlike the trinomial case, where square root operation is as fast as squaring [15].

IV. Software Implementation Results

We have implemented both Alg. 1 and 2 for Tate pairing over binary fields $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$ constructed via $f_{239}(x) = x^{239} + x^{36} + 1$ and $f_{283}(x) = x^{283} + x^{12} + x^7 + x^5 + 1$, respectively. The subfield arithmetic was realized via a public domain C++ library named LiDIA [20], and we developed the high level operations. All the codes were compiled with g++ 3.0.4 and simulations were performed on a Xeon station working at 2.8 GHz. With our software implementations, we can generate the bilinear test-vectors for our FPGA realizations of pairing. In Table I, we show that Alg. 2 is 1.5 times faster than Alg. 1 on average because half iterations in the accumulative multiplication stage are performed.

TABLE I
Timing of two algorithms of Tate pairing over binary fields, on a Xeon workstation at 2.8 GHz.

Finite Field $\mathbb{F}_{2^m}$	Latency of Alg. 1 (ms)	Latency of Alg. 2 (ms)	Speedup of Alg. 2 vs. Alg. 1
$\mathbb{F}_{2^{239}}$	10.8	7.5	1.44
$\mathbb{F}_{2^{283}}$	18.1	12.2	1.48

Although the arithmetic of paring computation of binary elliptic curve 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 binary field and the binary hyperelliptic curve has complex arithmetic operation for additions (which make the data path complicated for FPGA), it is desirable to design an FPGA accelerator 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 cubic case [1], [3] and $\mathbb{F}_{2^{283}}$ for the comparison with binary hyperelliptic case [4].

V. FPGA implementations

In this section, we focus on the FPGA implementations of Tate pairing on supersingular elliptic curves over binary fields, $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$. 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, 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:

$$s^2 + s + 1 = 0 \text{ and } t^2 + t + s = 0. \tag{3}$$

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 a half of 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.

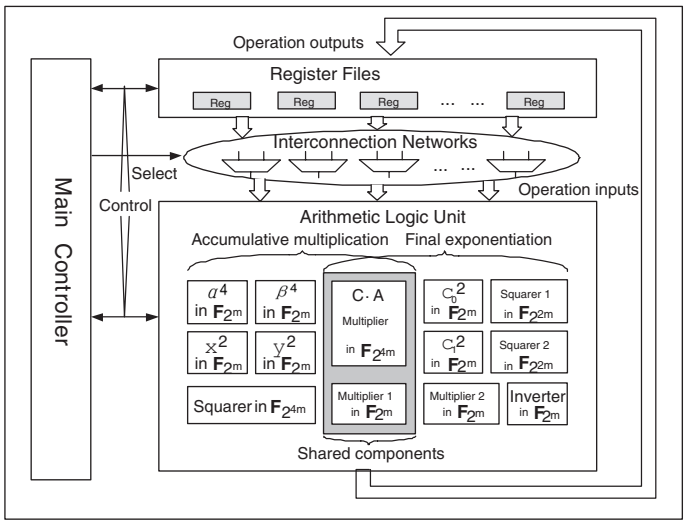

Fig. 1. Top architecture of the accelerator of Tate pairing over binary fields.

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. 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 cannot be completed in a single or small number of clock cycles, it is not suitable to be pipelined. Moreover, in program-directed operations, instructions are synchronously fetched, decoded and executed, which will reduce 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, 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 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 [21]. Since most intensive computations are performed in the first stage, 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.

A. 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.

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 given 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 for computing the coefficients of a^2. The circuit complexity in terms of gate count is proportional to m. Squaring over the extension fields $\mathbb{F}_{2^{2m}}$ and $\mathbb{F}_{2^{4m}}$ is relatively easy and can be decomposed into several squarings in $\mathbb{F}_{2^m}$.

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. It is claimed that the first algorithm is superior in term of low power [19]. Additionally, we find that fewer registers are needed when using the first algorithm because only partial product needs to be updated in each iteration apart from the shift-in digits. However, both partial product and one operand must be updated in each iteration when using the second one. Therefore

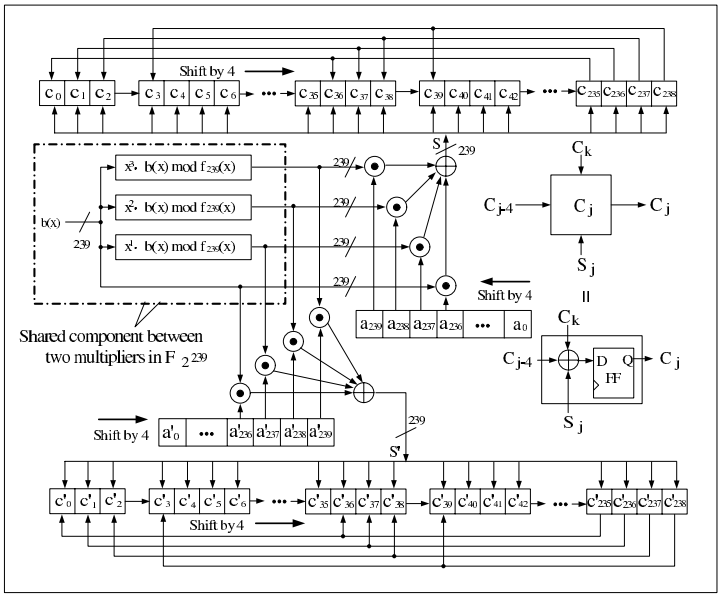

Fig. 2. Two digit serial multipliers in $\mathbb{F}_{2^{239}}$ with $D = 4$ sharing the component for $\sum_{i=0}^{3} x^i b(x) \bmod f_{239}(x)$.

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. Let $n = min\{l \mid l \in \mathbb{Z},\ l \geq m,$ 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. 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 fewer XOR gates are used in the second structure for an individual multiplier, these XOR-AND arrays cannot 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 5 and Fig. 2. The second advantage 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 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 5).

$$\begin{aligned} C \cdot (w + zs + et) &= (c_0 + c_1 s + c_2 t + c_3 st)(w + zs + et) \\ &= c'_0 + c'_1 s + c'_2 t + c'_3 st, \end{aligned} \tag{4}$$

where

$$\begin{aligned} c'_0 &= c_0 w + c_1 z + e c_3 \\ c'_1 &= (c_0 + c_1)(w + z) + c_0 w + e(c_2 + c_3) \\ c'_2 &= c_2 w + c_3 z + e(c_0 + c_2) \\ c'_3 &= (c_2 + c_3)(w + z) + c_2 w + e(c_1 + c_3) \end{aligned} \tag{5}$$

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. 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. To get the optimal choice in terms of low product of latency by area, both schemes of the special multipliers CA over $\mathbb{F}_{2^{4\times 239}}$ and $\mathbb{F}_{2^{4\times 283}}$ 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 II.

According to Table II, Scheme 1 is always superior in terms of low product of latency by area. Hence we adopt 6

TABLE II

FPGA IMPLEMENTATION RESULTS FOR THE MULTIPLIERS CA OVER $\mathbb{F}_{2^{4\times 239}}$ AND $\mathbb{F}_{2^{4\times 283}}$.

Scheme 1: 6 multipliers adopted for $\mathbb{F}_{2^{239}}$			
	$D = 4$	$D = 8$	$D = 16$
# FF	3664(4%)	3664(4%)	3664(4%)
# LUT	7799(8%)	13508(15%)	20744(23%))
# CLB slices	4268(9%)	7094(16%)	10722(24%))
Clock period (ns)	9.61	9.98	9.99
Latency (ns)	576.6	299.4	149.9
Scheme 2: 3 multipliers adopted for $\mathbb{F}_{2^{239}}$			
	$D = 4$	$D = 8$	$D = 16$
# FF	2675(3%)	2675(3%)	2675(3%)
# LUT	5580(6%)	8439(9%)	12075(13%)
# CLB slices	3617(8%)	5331(12%)	7536(17%)
Clock period (ns)	9.86	9.98	9.76
Latency (ns)	1182.8	598.8	292.8
Scheme 1: 6 multipliers adopted for $\mathbb{F}_{2^{283}}$			
	$D = 4$	$D = 8$	$D = 16$
# FF	4324(4%)	4348(4%)	4348(4%)
# LUT	9273(10%)	16060(18%)	24729(28%)
# CLB slices	5070(11%)	8416(19%)	12856(29%)
Clock period (ns)	9.84	9.98	9.99
Latency (ns)	698.6	359.3	179.8
Scheme 2: 3 multipliers adopted for $\mathbb{F}_{2^{283}}$			
	$D = 4$	$D = 8$	$D = 16$
# FF	3159(3%)	3171(3%)	3171(3%)
# LUT	6632(7%)	10031(11%)	14428(16%)
# CLB slices	3046(6%)	4483(10%)	6306(14%)
Clock period (ns)	8.89	9.99	9.98
Latency (ns)	1262.4	719.3	359.3

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.

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. V-B). Therefore, we only need to implement one inverter in $\mathbb{F}_{2^m}$. Itoh-Tsujii's algorithm is adopted in our realizations. It takes

$$\lfloor log_2(m-1) \rfloor + HW(m-1) - 1 \tag{6}$$

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

B. 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 $C_1^2 + C_1 C_2 + C_2^2 s = c_0 + c_1 s$ with $c_0, c_1 \in \mathbb{F}_{2^m}$, a straight calculation shows

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

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 $C_1 C_2$ and 1 $\mathbb{F}_{2^m}$-multiplication for $c_0 c_1$) for the denominator $c_0^2 + c_0 c_1 + c_1^2$, 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. 3 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. 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:

$$\begin{aligned} MT &= (2^{2m}-1)(2^m + 2^{\frac{m+1}{2}} + 1)(2^{\frac{m+1}{2}} - 1) \\ &= (2^{2m}-1)(2^m + 2^{\frac{m+1}{2}} + 1)2^m \end{aligned} \tag{7}$$

C. Parameter Choices for ALU

In Fig. 3, 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.

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. 3. Then we can estimate the latency for computing one Tate pairing using Alg. 1 as follows:

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

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 7. 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,

$$\begin{aligned} \Delta_2 &= \frac{m+5}{2}(T_1 + 2) + 3T_2 + T_3 + T_4 + 8; \\ T_4 &= \frac{m+1}{2} T_{clk} + 4(T_1 + 2) \end{aligned} \tag{9}$$

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.

D. 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. Our FPGA accelerators have been fully tested via the bilinear test-vectors generated by LiDIA. The results after placing and routing via Xilinx ISE-7.1 are summarized in Table III. Compared with software implementations, our FPGA accelerator can run 150-to-300 times faster.

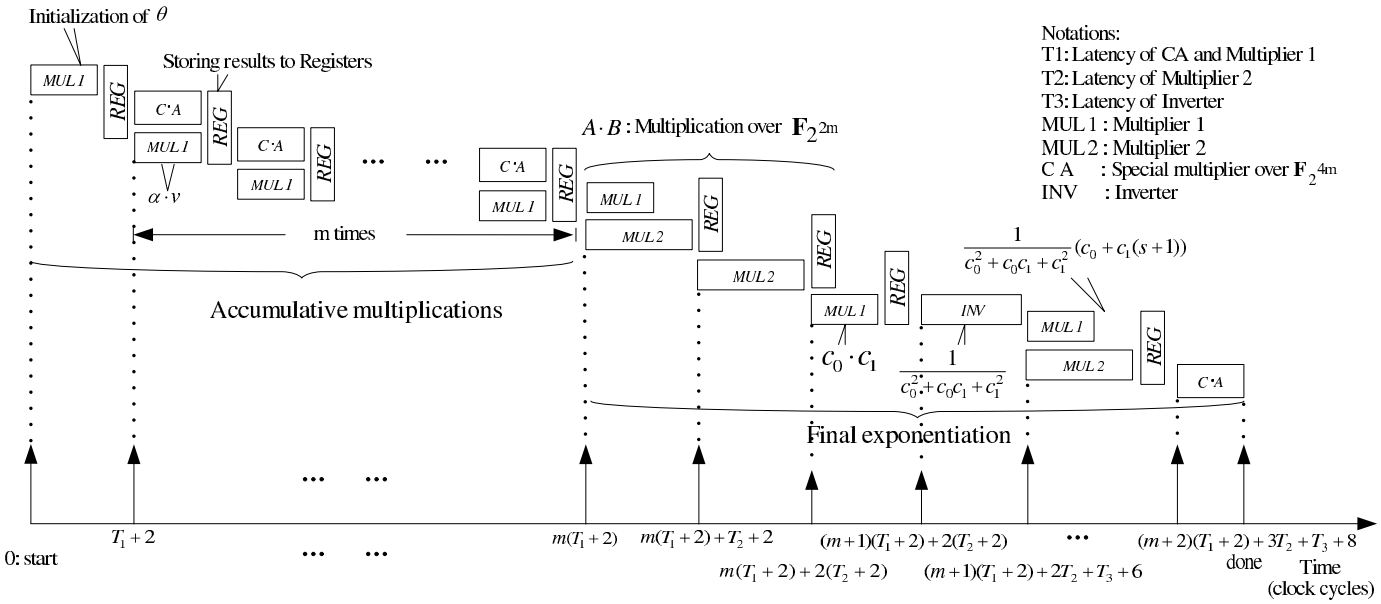

Fig. 3. Timing diagram of pairing computations according to Alg. 1.

TABLE III

PERFORMANCE AND COST OF TATE PAIRING ACCELERATORS ON ELLIPTIC CURVES OVER $\mathbb{F}_{2^{239}}$ AND $\mathbb{F}_{2^{283}}$.

		# FF	# LUT	# CLB slices	f (MHz)	Latency (μs)	Speedup over software
Alg. 1	$\mathbb{F}_{2^{239}}$, $D_1 = 16$	10,981 (12%)	34,499 (12%)	18,202 (41%)	100	55	196
	$\mathbb{F}_{2^{239}}$, $D_1 = 32$	11,077 (12%)	59,971 (68%)	31,719 (71%)	83	43	251
	$\mathbb{F}_{2^{283}}$, $D_1 = 16$	12,995 (14%)	42,997 (48%)	22,726 (51%)	84	87	208
	$\mathbb{F}_{2^{283}}$, $D_1 = 32$	13,007 (14%)	72,961 (82%)	37,803 (85%)	72	61	297
Alg. 2	$\mathbb{F}_{2^{239}}$, $D_1 = 16$	14,226 (16%)	48,895 (55%)	25,487 (57%)	84	41	183
	$\mathbb{F}_{2^{283}}$, $D_1 = 16$	16,563 (18%)	64,845 (73%)	33,252 (75%)	56	78	156

TABLE IV

COMPARISON WITH EARLIER FPGA IMPLEMENTATIONS.

	Curves	Underlying fields	MOV Security	Xilinx FPGA device	Controller	# CLB slices	Digit size D	f(MHz)	Latency (μs)
[1]	Elliptic	$\mathbb{F}_{3^{97}}$	922	XC2VP125	Hardwired logic	55,616	4	10	850
[3]	Elliptic	$\mathbb{F}_{3^{97}}$	922	XC2VP4FF672	Microprocessor	4,481	4	150	432
[4]	Hyperelliptic	$\mathbb{F}_{2^{103}}$	1236	XC2VP125	Hardwired logic	43,986	16	32	749
Alg. 2	Elliptic	$\mathbb{F}_{2^{239}}$	956	XC2VP100	Hardwired logic	25,287	16	84	41
Alg. 1	Elliptic	$\mathbb{F}_{2^{283}}$	1132	XC2VP100	Hardwired logic	37,803	32	72	61

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 cannot 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 10-to-20 times as fast as theirs on average with smaller product of latency by area. See Table IV, 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 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 cannot be large, so $D = 4$ is selected. 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 20 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 an FPGA-based co-processor, which is controlled by a general-purpose processor, i.e., the top architecture is a stored-program machine (SPM), as described previously. The 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 multiplication rounds are necessary for each iteration of the accumulative multiplication. In contrast, our processor supports 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 41 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 the same as the one used in [1]. A smaller underlying field $\mathbb{F}_{2^{103}}$ and a larger embedding degree $k = 12$ are selected. 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 computation of accumulative multiplications becomes much simpler. 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 43,986 slices are used. Our accelerator can run 12 times faster, whereas the resource utilization is only 37,803 slices, see Table IV.

VI. Conclusions

We investigated the FPGA implementations of the 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 chose 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 previously reported implementations, our pairing processors are more efficient in terms of the product of latency by area. Additionally, our accelerators can outperform earlier designs by a factor of 10-to-20 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*, LNCS 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* volume 3203 of *Lecture Notes in Computer Science*, pp. 74-83, Springer-Verlag 2004.

[3] P. Grabher and D. Page, "Hardware acceleration of the Tate pairing in charateristic three", *CHES 2005, Lecture Notes in Computer Science*, LNCS 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, *Cryptology ePrint Archive*, 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," preprint, *available at* http://eprint.iacr.org/ 2004/157.pdf, 2004.

[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. OhEigeartaigh, and M. Scott, "Efficient pairing computation on supersingular abelian varieties," preprint *available at* http://eprint.iacr.org/2004/375.pdf, 2004.

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

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

[12] V. Miller, "Short programs for functions on curves," *unpublished manuscript*, 1986.

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

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

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

[16] 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.

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

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

[19] 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.

[20] LiDIA, A C++ Library For Computational Number Theory, *available at* http://www.informatik.tu-darmstadt.de/TI/LiDIA.

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