Reconfigurable Computing Approach for Tate Pairing Cryptosystems over Binary Fields

Chang Shu, Soonhak Kwon, and Kris Gaj

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. In this paper, we propose a new FPGA-based architecture of the Tate pairing-based computation over binary fields. 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 be performed. As a result, the computational latency for a pairing computation has been reduced, and our implementation runs 2-to-20 times faster than the equivalent implementations of other pairing-based schemes at the same level of security strength. Furthermore, we ported our pairing designs for 8 field sizes ranging from 239 to 557 bits to the reconfigurable computer, SGI Altix-4700 supported by Silicon Graphics, Inc., and performance and cost are demonstrated.

Index Terms

Tate pairing, elliptic curve, reconfigurable computing, Field Programmable Gate Arrays (FPGAs), finite field.

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 [22]. Applying pairing techniques to identity based cryptography was suggested first by Boneh and Franklin [24], and independently by Sakai et al. [25].

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. [15], Barreto et al. [12], Granger et. al. [9], [10], and Duursma and Lee [11]. The optimizations introduced by these authors involved delicate techniques of deleting unnecessary operations from the Miller’s algorithm [16]. In particular, the work of Duursma and Lee [11] promoted the study of efficient pairing computations for elliptic curves over Galois fields of characteristic three, $F_{3^m}$. Subsequently, their idea was applied to the case of binary fields in [14], and generalized by Barreto et al. [13] to encompass different characteristics.
of the underlying field for the case of supersingular elliptic curves. Recently, Hess, Smart, and Vercauteren [17] introduced the Ate pairing by generalizing the previous Eta pairing proposed in [13], so this new pairing can be applied to more general class of curves including both supersingular and ordinary elliptic 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. Ronan et al. [4] proposed the dedicated hardware for computing pairing on hyperelliptic curves using the algorithm introduced in [13]. Recently, Komurcu et al. [5] and Beuchat et al. [6]–[8] have improved implementations of pairing over cubic elliptic curves. Especially, resource utilization can be reduced considerably in Beuchat et al.’s design.

In public key cryptosystems, such as RSA, elliptic curve, and pairing, latency instead of throughput is frequently used to measure the performance, as the applications of these algorithms are generally key management and digital signature generation or verification. For these applications, latency, defined as the time of computations for one set of data, is important, as it determines the total response time. In this paper, we propose a low latency hardware accelerator for the Tate pairing-based cryptosystems on supersingular elliptic curves over binary fields. In order to compare our pairing schemes with others at the same level of security strength, we select the underlying fields and curves based on the analysis via MAPLE [31]. 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 as well as other operand sizes.

Our implementation is based on the algorithms presented in [13] and [14]. Even though the binary elliptic curves require relatively long operands (compared to the cubic elliptic and binary hyperelliptic curves), 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 2-to-20 times faster than the ones published in [1], [3]–[8], at the same level of security strength.

Furthermore, we choose a Reconfigurable computer, SGI Altix-4700 provided by Silicon Graphics, Inc., as the end-to-end system for our pairing computations over 8 selected binary fields ranging from $\mathbb{F}_2^{239}$ to $\mathbb{F}_2^{557}$. Our paper, as one of the first in the literature, demonstrates the practical implementation of completing pairing computations on the high-performance reconfigurable computing platform, which is capable of supporting key management and digital signature generation/verification for high end servers. Our results demonstrate that using pairing based cryptography in place of RSA in this kind of applications is fully feasible, without sacrificing security or speed. We also demonstrate the development of a generic library of IP cores for pairing-based schemes with a wide range of key sizes providing various levels of security strength. In the investigated programming environment, these hardware
IP cores can be then called from a high level language, such as C, and the security level can be changed at runtime.

This paper is organized as follows. In Section II, we introduce the mathematical background behind Tate pairing computations. In Section III, we present two algorithms adopted in our implementations. The issues of choosing the underlying binary fields for pairing friendly elliptic curves are also addressed. So far, the underlying fields have been reported primarily for the case of pairing friendly elliptic curves over cubic fields and hyperelliptic curves over binary fields. We have developed a MAPLE program in order to generate multiple pairing friendly elliptic curves over binary fields. Fields generated by this program are reported in Table I, and the entire approach for choosing appropriate field sizes and the corresponding curves is described in Section IV, Choice of Underlying Fields. In Section V, comparison between two proposed algorithms for pairing computations in software is performed. Software results for all eight fields, reported in Table III, are then later compared with the corresponding hardware results in Table VII, demonstrating a very substantial speed-up offered by our FPGA implementations over the state-of-the-art software implementations based on the well known LiDIA [28] library. In Section VI, we describe in detail our FPGA-based implementations, which includes top architecture of pairing accelerator, optimization technique for both underlying field and extension field multipliers, and simplifying datapath for the final exponentiation. In Section VII, details of porting hardware IP cores of pairing to SGI Altix-4700 are demonstrated. Finally, the conclusions from our research are drawn in Section VIII.

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, $D = \sum n_P(P), \ n_P \in \mathbb{Z}$. 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 [18] for elementary introduction of divisor theories, and [19]–[21] for more knowledge of Tate pairing. The (reduced) Tate pairing $\tau_l$ on the set $E[l]$ is defined as follows.

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

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

with $\tau_l(P, Q) = f_P(D_Q)^{\frac{1}{l-1}}$, 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 $\tau_l$ is a non-degenerate bilinear pairing [18]. 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 [16]. The Miller’s algorithm was further improved by the works of [9], [10], [12]–[15].

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

$$E_b : Y^2 + Y = X^3 + X + b, \ b = 0, 1.$$ (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{equation}
|E_b(F_{2^m})| = 2^m + 1 + (-1)^b 2^{m+1}, \text{ if } m \equiv 1, 7 \pmod{8} \quad (2) \\
= 2^m + 1 - (-1)^b 2^{m+1}, \text{ if } m \equiv 3, 5 \pmod{8}.
\end{equation}

III. ALGORITHMS FOR PAIRING FOR SUPERSINGULAR ELLIPTIC CURVES OVER BINARY FIELDS

Inspired by the work of Duursma and Lee [11], a nice formula for the Tate pairing computation of supersingular elliptic curve over binary field was proposed in [13], [14]. Moreover by introducing the Eta pairing technique, revised version of [13] contains an improved formula which reduces the number of iterations by half. The algorithms in [13], [14] use repeated product of the term $g_{2iP}(\psi Q)$. 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 $\psi$ 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 [13], [14] imply that the Tate pairing $\tau(P, Q)$ is given by
\[
\tau(P, Q) = \left( \prod_{i=0}^{m-1} g_{2iP}(\psi Q)^{2^{2m-1}} \right)^{2^{2m-1}}.
\]

**Algorithm 1** A modified algorithm from [13], [14] for parallel computation of Tate pairing.

**Require:** $P = (\alpha, \beta), Q = (x, y)$

**Ensure:** $C = \tau(P, Q)$

1. $C \leftarrow 1$
2. $\alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4, v \leftarrow x^2 + 1, \theta \leftarrow \alpha v, u \leftarrow x^2 + y^2 + b + m - 1$ \{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. $C \leftarrow C \cdot A$ \{Computations of Steps 6 and 7 can be performed in parallel.\}
7. $\alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4, u \leftarrow u + v, \quad v \leftarrow v + 1, \quad \theta \leftarrow \alpha v$
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 $2^i P = \phi^i(\alpha^{(2i)}, \beta^{(2i)})$, where $\phi$ is defined as $\phi(x, y) = (x + 1, y + x)$ and $\alpha^{(i)}$ denotes $\alpha^{(i)} = \alpha^{2i}$ 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
\[ g_{2i}(x, y) = (\alpha_i^{(2i+1)} + 1)x + \beta_i^{(2i+1)} + b + y \] (3)
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 \(\phi\) and the Frobenius map are commutative to each other.

By refining the Eta pairing approach, Barreto et al. [13] successfully reduced the number of loop iterations by half so that they showed
\[
\tau(P, Q) = \left( \ell(\psi Q) \prod_{i=0}^{m-1} g_{2i} P(\psi Q)^{2\frac{m-1-1}{2}} \right)^{MT} 
\] (4)
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 \(\epsilon P\) with \(\epsilon = (-1)^{b + \frac{m+1}{2}}\). \(\ell(X, Y)\) is given by
\[ \ell(X, Y) = Y + \beta + b + \frac{m+1}{2} + (\alpha + \frac{m-1}{2})(X + \alpha). \]

**Algorithm 2** A modified algorithm from [13] for parallel computation of Tate pairing.

**Require:** \(P = (\alpha, \beta), Q = (x, y)\)

**Ensure:** \(C = \tau(P, Q)\)

1: \(C \leftarrow 1\)
2: \(\alpha \leftarrow \alpha^2 + 1, \quad \beta \leftarrow \beta^2 + 1, \quad u \leftarrow y + b + 1, \quad v \leftarrow x + 1, \quad \theta \leftarrow \alpha v \quad \{\text{Initialize}\}\)
3: \(\text{for } i = 0 \text{ to } \frac{m-1}{2} \text{ do}\)
4: \(A \leftarrow \beta + \theta + u + (\alpha + v + 1)s + t\)
5: \(C \leftarrow C^2\)
6: \(C \leftarrow C \cdot A \quad \{\text{Computations of Steps 6-9 can be performed in parallel.}\}\)
7: \(\text{if } i < \frac{m-1}{2} \text{ then}\)
8: \(\quad \alpha \leftarrow \alpha^4, \quad \beta \leftarrow \beta^4, \quad u \leftarrow u + v + 1, \quad v \leftarrow v + 1, \quad \theta \leftarrow \alpha v\)
9: \(\text{end if}\)
10: \(\text{end for}\)
11: \(A \leftarrow A + (\alpha^2 + v + 1) + s\)
12: \(C \leftarrow C \cdot A\)
13: \(C \leftarrow C^{MT}, \quad MT = (2^{2m-1})(2^m \mp 2^{\frac{m+1}{2}} + 1)(2^{\frac{m+1}{2}} \pm 1) \quad \{\text{Final exponentiation}\}\)

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 [13]. This is possible because the last element \(g_{2^{\frac{m-1}{2}}} P(\psi Q)\) of the product in Equation 4 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 $\alpha$ and $\beta$ 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 $\mathbb{F}_{2^{283}}$ for our single FPGA implementations. 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 [23].

IV. 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 I are taken from [10], [13] and the binary elliptic cases are computed using MAPLE [31].

In our case of binary elliptic curves, the form of the pairing friendly supersingular curves is given in Equation 1, and the corresponding order of the elliptic curve is given in Equation 2. We want this order to be a prime or a near prime. That is $|E_b(\mathbb{F}_{2^m})|/c$ is a prime for small $c$. Using Maple, we did exhausted search for $m$ between 200 and 600 so that we tried to factor $|E_b(\mathbb{F}_{2^m})|$ to check whether $|E_b(\mathbb{F}_{2^m})|$ is a prime or has a very small cofactor. The resulting table of binary elliptic case given in Table I is a complete list of applicable curves with small cofactors. Also in each applicable $m$, one may always choose a low hamming weight polynomial basis such as a trinomial or a pentanomial basis [32]. We used the trinomial basis if it exists, otherwise the pentanomial basis is used.

In Table I, 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].

V. Software Results

We have implemented Alg. 1 and 2 for Tate pairing over binary fields listed in Table I. The generating polynomials for these fields are provided in Table II. The subfield arithmetic was realized via a public domain C++ library named
TABLE I
APPLICABLE CURVES FOR CUBIC ELLIPTIC, BINARY HYPERELLIPTIC AND BINARY ELLIPTIC CASES.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Curves</th>
<th>Co-Factors</th>
<th>MOV Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_{3^{79}}$</td>
<td>$Y^2 = X^3 - X - 1$</td>
<td>1</td>
<td>750</td>
</tr>
<tr>
<td>$\mathbb{F}_{3^97}$</td>
<td>$Y^2 = X^3 - X + 1$</td>
<td>7</td>
<td>922</td>
</tr>
<tr>
<td>$\mathbb{F}_{3^{163}}$</td>
<td>$Y^2 = X^3 - X - 1$</td>
<td>1</td>
<td>1548</td>
</tr>
<tr>
<td>$\mathbb{F}_{3^{193}}$</td>
<td>$Y^2 = X^3 - X - 1$</td>
<td>1</td>
<td>1830</td>
</tr>
<tr>
<td>$\mathbb{F}_{3^{239}}$</td>
<td>$Y^2 = X^3 - X - 1$</td>
<td>1</td>
<td>2268</td>
</tr>
<tr>
<td>$\mathbb{F}_{3^{353}}$</td>
<td>$Y^2 = X^3 - X - 1$</td>
<td>1</td>
<td>3354</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^79}$</td>
<td>$Y^2 + Y = X^5 + X^3 + 1$</td>
<td>151681</td>
<td>948</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{103}}$</td>
<td>$Y^2 + Y = X^5 + X^3$</td>
<td>13 · 1237</td>
<td>1236</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{127}}$</td>
<td>$Y^2 + Y = X^5 + X^3 + 1$</td>
<td>198168459411337</td>
<td>1524</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{199}}$</td>
<td>$Y^2 + Y = X^5 + X^3 + 1$</td>
<td>2389 · 121789</td>
<td>2388</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{239}}$</td>
<td>$Y^2 + Y = X^5 + X^3 + 1$</td>
<td>1</td>
<td>2808</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{313}}$</td>
<td>$Y^2 + Y = X^5 + X^3 + 1$</td>
<td>1</td>
<td>3756</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{239}}$</td>
<td>$Y^2 + Y = X^3 + X + 1$</td>
<td>1</td>
<td>956</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{241}}$</td>
<td>$Y^2 + Y = X^3 + X + 1$</td>
<td>1</td>
<td>964</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{283}}$</td>
<td>$Y^2 + Y = X^3 + X$</td>
<td>5</td>
<td>1132</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{283}}$</td>
<td>$Y^2 + Y = X^3 + X + 1$</td>
<td>1</td>
<td>1412</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{353}}$</td>
<td>$Y^2 + Y = X^3 + X + 1$</td>
<td>1</td>
<td>1468</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{379}}$</td>
<td>$Y^2 + Y = X^3 + X + 1$</td>
<td>1</td>
<td>1516</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{457}}$</td>
<td>$Y^2 + Y = X^3 + X + 1$</td>
<td>1</td>
<td>1828</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{557}}$</td>
<td>$Y^2 + Y = X^3 + X$</td>
<td>5</td>
<td>2228</td>
</tr>
</tbody>
</table>

LiDIA, 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. Additionally, the comparisons of timing between hardware and software are performed in the following sections. In Table III, we found that Alg 2 is 1.5 times faster than Alg. 1 on average because half iterations in the accumulative multiplication stage are performed. The speedup of Alg. 2 vs. Alg. 1 is lower for $\mathbb{F}_{2^{379}}$ because more squarings in $\mathbb{F}_{2^{241}}$ will be performed sequentially in the final exponentiation stage of Alg. 2.

TABLE II
GENERATING POLYNOMIALS FOR THOSE SELECTED BINARY FIELDS.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Generating Polynomials $f_m(x)$</th>
<th>Fields</th>
<th>Generating Polynomials $f_m(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_{2^{239}}$</td>
<td>$x^{239} + x^{36} + 1$</td>
<td>$\mathbb{F}_{2^{367}}$</td>
<td>$x^{367} + x^{21} + 1$</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{241}}$</td>
<td>$x^{241} + x^{70} + 1$</td>
<td>$\mathbb{F}_{2^{379}}$</td>
<td>$x^{379} + x^{10} + x^8 + x^5 + 1$</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{283}}$</td>
<td>$x^{283} + x^{12} + x^7 + x^5 + 1$</td>
<td>$\mathbb{F}_{2^{457}}$</td>
<td>$x^{457} + x^{16} + 1$</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{353}}$</td>
<td>$x^{353} + x^{69} + 1$</td>
<td>$\mathbb{F}_{2^{557}}$</td>
<td>$x^{557} + x^7 + x^6 + x^3 + 1$</td>
</tr>
</tbody>
</table>
TABLE III
THE TIMING OF BOTH ALGORITHMS OF THE TATE PAIRING (IN MS) OVER BINARY FIELDS, ON A XEON WORKSTATION AT 2.8 GHz.

<table>
<thead>
<tr>
<th>Finite Fields $\mathbb{F}_{2^m}$</th>
<th>Latency of Alg. 1</th>
<th>Latency of Alg. 2</th>
<th>Speedup Alg. 2 vs. Alg. 1</th>
<th>Finite Fields $\mathbb{F}_{2^m}$</th>
<th>Latency of Alg. 1</th>
<th>Latency of Alg. 2</th>
<th>Speedup Alg. 2 vs. Alg. 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_{2^{239}}$</td>
<td>10.8</td>
<td>7.5</td>
<td>1.44</td>
<td>$\mathbb{F}_{2^{267}}$</td>
<td>28.8</td>
<td>18.4</td>
<td>1.57</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{241}}$</td>
<td>11.0</td>
<td>7.7</td>
<td>1.43</td>
<td>$\mathbb{F}_{2^{279}}$</td>
<td>31.0</td>
<td>22.5</td>
<td>1.38</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{283}}$</td>
<td>18.1</td>
<td>12.2</td>
<td>1.48</td>
<td>$\mathbb{F}_{2^{347}}$</td>
<td>42.5</td>
<td>26.7</td>
<td>1.59</td>
</tr>
<tr>
<td>$\mathbb{F}_{2^{353}}$</td>
<td>27.0</td>
<td>16.9</td>
<td>1.60</td>
<td>$\mathbb{F}_{2^{557}}$</td>
<td>84.3</td>
<td>52.3</td>
<td>1.61</td>
</tr>
</tbody>
</table>

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 makes the data path complicated for FPGAs, see [1], [3], [4]. On the other hand, the arithmetic of pairing computations on binary elliptic curves is very simple. Therefore, 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, binary hyperelliptic, and binary elliptic cases, we chose two fields $\mathbb{F}_{2^{239}}$ for the comparison with the cubic case [1], [3], [5]–[8] and $\mathbb{F}_{2^{283}}$ for the comparisons with the binary hyperelliptic case [4] and the binary elliptic case [8].

VI. 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 $s \in \mathbb{F}_{2^m}$, $t \in \mathbb{F}_{2^m}$ satisfying:

$$s^2 + s + 1 = 0 \text{ and } t^2 + t + s = 0. \quad (5)$$

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.

**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 $\mathbb{F}_{2^m}$ and $\mathbb{F}_{2^{2m}}$, the inversion in 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

---

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

---

February 4, 2009 DRAFT
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 most significant digit (MSD) serial multipliers. The multiplicative inversion in \( \mathbb{F}_{2^m} \) is computed using Itoh-Tsujii algorithm [29]. 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 \( x^k + x^{k2} + x^{k1} + 1 \), 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:** Hardware architectures proposed in the literature for \( \mathbb{F}_{2^m} \) multiplication can be treated in three major classes: bit parallel, bit serial, and digit serial multipliers. Bit parallel multipliers can complete one multiplication in one clock cycle with maximal circuit complexity which makes it unsuitable for those multiplications with large operand sizes. Bit serial multipliers with polynomial basis (PB) representation can be derived from Galois linear-feedback-shift-registers (LFSRs). The updated value of Galois LFSRs for next clock cycle is equal to the product of the primitive element (the root of the primitive generating polynomial) and the current field element stored in LFSRs, i.e., Galois LFSRs can be treated as a special bit serial multiplier with a constant (the primitive element). For a bit serial multiplier with two variables, LFSRs can be used either to store the final product, or the product of one operand and the power of the generator. The former is named as most-significant-bit (MSB) serial multiplier, and the latter is named as least-significant-bit (LSB) serial multiplier. It should be noticed that
the generating polynomial of Galois LFSRs must be primitive but not necessarily primitive for bit serial multiplier. Even though bit serial multipliers are efficient in terms of area, the operation speed is sacrificed. Digit serial multipliers, obtained by parallelizing bit serial multipliers, allow tradeoff between speed and area, which makes them more suitable for cryptographic applications. Correspondingly, there are two basic digit serial algorithms for multiplications with PB representation in $F_{2^m}$, most-significant-digit (MSD) and least-significant-digit (LSD), see Alg. 3 and Alg. 4.

**Algorithm 3** Pseudocode for MSD serial multiplication.

**Require:** $a(x) = \sum_{i=0}^{m-1} a_ix^i$, $b(x) = \sum_{i=0}^{m-1} b_ix^i \in F_{2^m}$ and generating polynomial $f_m(x)$. Let $D$ denote the digit size. Let $n = \min\{l \mid l \in \mathbb{Z}, l \geq m, \text{ and } D \mid l\}$. Let $a'(x) = \sum_{i=0}^{n-1} a'_ix^i$, where $a'_i = a_i$ if $0 \leq i \leq m-1$, otherwise $a'_i = 0$.

**Ensure:** $c(x) = a(x) \cdot b(x) \mod f_m(x)$.

1: $c(x) \leftarrow 0$, $a'(x) \leftarrow a(x)$.
2: for $i = \lceil \frac{m}{D} \rceil - 1$ downto 0 do
3: for $j = D - 1$ downto 0 do
4: $d_j(x) \leftarrow a'_{n-D+j}x^j \cdot b(x) \mod f_m(x)$.
5: end for {Computations of $d_j(x)$ can be performed in parallel.}
6: $c(x) \leftarrow (x^D \cdot c(x) \mod f_m(x)) + \sum_{j=0}^{D-1} d_j(x)$.
7: $a'(x) \leftarrow a'(x) \ll D$.
8: end for

**Algorithm 4** Pseudocode for LSD serial multiplication.

**Require:** $a(x) = \sum_{i=0}^{m-1} a_ix^i$, $b(x) = \sum_{i=0}^{m-1} b_ix^i \in F_{2^m}$ and generating polynomial $f_m(x)$. Let $D$ denote the digit size. Let $n = \min\{l \mid l \in \mathbb{Z}, l \geq m, \text{ and } D \mid l\}$. Let $a'(x) = \sum_{i=0}^{n-1} a'_ix^i$, where $a'_i = a_i$ if $0 \leq i \leq m-1$, otherwise $a'_i = 0$.

**Ensure:** $c(x) = a(x) \cdot b(x) \mod f_m(x)$.

1: $c(x) \leftarrow 0$, $a'(x) \leftarrow a(x)$.
2: for $i = \lceil \frac{m}{D} \rceil - 1$ downto 0 do
3: for $j = D - 1$ downto 0 do
4: $d_j(x) \leftarrow a'_ix^j \cdot b(x) \mod f_m(x)$.
5: end for {Computations of $d_j(x)$ can be done in parallel.}
6: $c(x) \leftarrow c(x) + \sum_{j=0}^{D-1} d_j(x)$.
7: $b(x) \leftarrow x^D \cdot b(x) \mod f_m(x)$ {Computations of $c(x)$ and $b(x)$ can be performed in parallel.}
8: $a'(x) \leftarrow a'(x) \gg D$.
9: end for
The digit serial multiplier contains two main parts:

1) XOR-AND arrays computing $s(x) = \sum_{j=0}^{D-1} a'_{n-D+j} x^j b(x) \mod f_m(x)$ in Alg. 3, or $s(x) = \sum_{j=0}^{D-1} a'_j x^j b(x) \mod f_m(x)$ in Alg. 4.

2) LFSRs for computing $c(x) \leftarrow c(x) \cdot x^D + s(x)$ in Alg. 3, or $b(x) \leftarrow b(x) \cdot x^D$ in Alg. 4.

In Fig. 2 and Fig. 3, we provide both architectures via an example of $\mathbb{F}_{2^{239}}$ in which the generating polynomial is chosen as $f_m(x) = x^{239} + x^{36} + 1$, and $D = 4$. For LSD serial multiplier, all three polynomials, $a(x)$, $b(x)$, and $c(x)$, need to be updated every clock cycle, see Fig. 3. However, MSD based architecture does not require updating $b(x)$, see Fig. 2. Therefore registers for $b(x)$ as well as power can be saved when using Alg. 3, for more details, see [27]. Hence we choose MSD serial multiplier for our implementations of pairing cryptosystems. For the XOR-AND array, there are two approaches, see Fig. 4. The first approach is to compute $x^j b(x) \mod f_m(x)$ individually and keep the partial sum in $m$ bits. For the second approach, 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 $\mathbb{F}_{2^{2m}}$ considering that the XOR arrays for $x^j b(x) \mod f_m(x)$ can be shared among different multipliers in $\mathbb{F}_{2^m}$ in case they share one operand, see Alg. 5 and Fig. 5. Another advantage is its low wire density which makes it easy for placing and routing.

In the following, we will introduce the optimization technique for the extension field multiplier $CA$. With the basis $\{1, s, t, st\}$ of $\mathbb{F}_{2^{2m}}$ 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, because 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 Alg. 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, see Fig. 6(a). Alternatively, if 3 multipliers are adopted in case of limited resources, the computation of $C \cdot A$ will take two multiplication rounds. During the first round, the select signal is asserted high, and two coefficients of the product, namely $c'_0$ and $c'_1$, will be computed. Once the first round multiplications are completed, the enable signal is asserted high to store $c'_0$ and $c'_1$ in the registers. During
Fig. 5. Two MSD serial multipliers in $\mathbb{F}_{2^{239}}$ with $D = 4$ sharing the component for $\sum_{j=0}^{3} x^j b(x) \mod f_{239}(x)$.

Fig. 6. Alternative schemes for $CA$, where $A = w + z s + e t$, $w, z \in \mathbb{F}_{2^m}$ and $e \in \mathbb{F}_2$, $C = c_0 + c_1 s + c_2 t + c_3 st$, $c_i \in \mathbb{F}_{2^m}$ for $0 \leq i \leq 3$, and $C' = C \cdot A = c'_0 + c'_1 s + c'_2 t + c'_3 st$, $c'_i \in \mathbb{F}_{2^m}$ for $0 \leq i \leq 3$. 
Algorithm 5 Pseudocode for the extension field multiplication $C \cdot A$.

**Require:** $A = w + z\bar{s} + e\bar{t}$ where $w, z \in \mathbb{F}_{2^m}$ and $e \in \mathbb{F}_2$, $C = c_0 + c_1s + c_2t + c_3st$, $c_i \in \mathbb{F}_{2^m}$ for $0 \leq i \leq 3$.

**Ensure:** $C' = C \cdot A = c_0' + c_1's + c_2't + c_3'st'$, $c_i' \in \mathbb{F}_{2^m}$ for $0 \leq i \leq 3$.

1: $c_0' \leftarrow c_0w + c_1z + ec_3$.
2: $c_1' \leftarrow (c_0 + c_1)(w + z) + c_0w + ec_2 + c_3$.
3: $c_2' \leftarrow c_2w + c_3z + ec_0 + c_2$.
4: $c_3' \leftarrow (c_2 + c_3)(w + z) + c_2w + ec_1 + c_3$.

**TABLE IV**

FPGA implementation results for the multipliers $CA$ over $\mathbb{F}_{2^{4\times239}}$ and $\mathbb{F}_{2^{4\times283}}$, and the target device is Xilinx XC2VP100-6FF-1704.

<table>
<thead>
<tr>
<th>Scheme 1: 6 multipliers adopted for $\mathbb{F}_{2^{239}}$</th>
<th>Scheme 1: 6 multipliers adopted for $\mathbb{F}_{2^{283}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D = 4$</td>
<td>$D = 8$</td>
</tr>
<tr>
<td># FF</td>
<td>3664(4%)</td>
</tr>
<tr>
<td># LUT</td>
<td>7799(8%)</td>
</tr>
<tr>
<td># CLB slices</td>
<td>4268(9%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scheme 2: 3 multipliers adopted for $\mathbb{F}_{2^{239}}$</th>
<th>Scheme 2: 3 multipliers adopted for $\mathbb{F}_{2^{283}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D = 4$</td>
<td>$D = 8$</td>
</tr>
<tr>
<td># FF</td>
<td>2675(3%)</td>
</tr>
<tr>
<td># LUT</td>
<td>5580(6%)</td>
</tr>
<tr>
<td># CLB slices</td>
<td>3617(8%)</td>
</tr>
</tbody>
</table>

the second round, the select signal is asserted low, so that the other two coefficients, namely $c_2'$ and $c_3'$, will be computed, see Fig. 6(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. Comparisons in terms of timing and area are demonstrated in Table IV. The optimization goal is speed instead of area. In addition, the hierarchy is not kept in order to share registers and XOR arrays among multipliers, because several underlying field multiplications share one operand.

According to Table IV, Scheme 1 is always better 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.

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. VI-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
\[
\ceil{\log_2(m - 1)} + \text{HW}(m - 1) - 1
\]
multiplications in $\mathbb{F}_{2^m}$ to complete one inversion, where $\text{HW}(m - 1)$ denotes the Hamming weight of $m - 1$.

**B. Simplifying the Data-path for the Final Exponentiation**

Let $C = C_1 + C_2t$ with $C_1, C_2 \in \mathbb{F}_{2^m}$. Using subfield arithmetic and repeated norm calculations, we may compute $C^{2^m-1}$. 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
\[
C_1^{2^m-1} = \frac{1}{c_0^2 + c_0c_1 + c_1} \cdot (c_0 + c_1(s + 1)) \cdot (C_1^2 + c_1^2(1 + s) + C_1^2 t),
\]
where the total cost of the above computations is one $\mathbb{F}_{2^m}$-inversion plus 12 $\mathbb{F}_{2^m}$-multiplications. That is, we need 4 $\mathbb{F}_{2^m}$-multiplications (1 $\mathbb{F}_{2^m}$-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 $\mathbb{F}_{2^m}$-multiplications for $\frac{1}{c_0^2 + c_0c_1 + c_1} \cdot (c_0 + c_1(s + 1))$, and 6 $\mathbb{F}_{2^m}$-multiplications (2 $\mathbb{F}_{2^m}$-multiplications) which come from the final multiplication by $C_1^2 + c_1^2(1 + s) + C_1^2 t$. Therefore only six multiplication units (for parallel processing) are needed here and one can reuse the same arithmetic units as shown in Fig. 7 for multiplications, and thus the additional cost of the final exponentiation is the inversion circuit for computing $\frac{1}{c_0^2 + c_0c_1 + c_1}$.

In Algorithm 2, we have more complicated final exponentiation, where $M = \frac{2^{4m-1}}{2^{2m} - 2^{m} + 1}$ and $T = 2^{m+1} \pm 1$. Here we should be cautious about appropriate sign of $T$. 

---

**Fig. 7.** Timing diagram of pairing computations via Alg. 1.
Fig. 8. Timing diagram of pairing computations via Alg. 2.

The original definition of \( T \) [13] 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 \( z^{2^{m+1}} = w^{2^{4m} - 1} = 1 \) 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

\[
z(2^{m+1} + 2^{\frac{m+1}{2}}) (2^{m+1} - 2^{\frac{m+1}{2}}) = z(2^m + 2^{\frac{m+1}{2}}) 2^m
\]

Therefore the exponent \( MT \) can be interpreted as:

\[
MT = (2^{2m} - 1)(2^m + 2^{\frac{m+1}{2}} + 1)(2^{\frac{m+1}{2}} - 1) \\
= (2^{2m} - 1)(2^m + 2^{\frac{m+1}{2}} + 1) 2^m
\]
C. Parameter Choices for ALU

In Fig. 7, we provide the timing diagram of scheduling computations for pairing according to Alg 1. Additions and squarings realized via combinational circuits need not to be taken into account for estimating the latency.

Let $\Delta_1$ 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. 7. Then we can estimate the latency for computing one Tate pairing using Alg. 1 as follows:

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

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 for the inversion in $\mathbb{F}_{2^{239}}$. Then $T_3 \approx (60 + 12 \cdot [239/D_3])T_{clk}$. Similarly $T_3 \approx (71 + 11 \cdot [283/D_3])T_{clk}$ 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 $\mathbb{F}_{2^{283}}$.

The timing diagram of Alg. 2 is shown in Fig. 8. The computation of $C^{MT}$ needs to be additionally taken into account for latency estimation, see Equation 7. The extension field exponentiation can be accelerated via the component computing $C^{2^4}$, and one extension field multiplication with two operands in $\mathbb{F}_{2^{4m}}$ can be completed by CA in two underlying field multiplication rounds. We can estimate the latency of pairing for Alg. 2 via the following equations,

$$\Delta_2 = \frac{m + 9}{2}(T_1 + 2) + 3T_2 + T_3 + T_4 + 8;$$

$$T_4 = \frac{m + 1}{2}T_{clk} + 4(T_1 + 2)$$

where $T_4$ denotes the latency for $C^{MT}$. The same ratio of $D_1, D_2$ and $D_3$ as Alg. 1 is also adopted for Alg. 2.

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 V. Compared with software implementations described in Section V, our FPGA accelerator can run 150-to-300 times faster.

If the digit sizes are chosen the same for both algorithms, the latency of Alg. 2 is smaller than the latency of Alg. 1. However more resources are utilized for Alg. 2 because the multiple squarers in $\mathbb{F}_{2^{4m}}$ are adopted for $C^{MT}$.
TABLE V

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

TABLE VI

<table>
<thead>
<tr>
<th></th>
<th>Curves</th>
<th>Underlying fields</th>
<th>MOV Security</th>
<th>Xilinx FPGA device</th>
<th>Controller</th>
<th># CLB slices</th>
<th>Size of BRAMs</th>
<th>Digit size D</th>
<th>f(MHz)</th>
<th>Latency (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Elliptic</td>
<td>( F_{3^{97}} )</td>
<td>922</td>
<td>XC2VP125</td>
<td>HL</td>
<td>55,616</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>850</td>
</tr>
<tr>
<td>[3] (^b)</td>
<td>Elliptic</td>
<td>( F_{3^{97}} )</td>
<td>922</td>
<td>XC2VP4FF672</td>
<td>M</td>
<td>4,481</td>
<td>NA</td>
<td>4</td>
<td>150</td>
<td>432</td>
</tr>
<tr>
<td>[4]</td>
<td>Hyperelliptic</td>
<td>( F_{2103} )</td>
<td>1236</td>
<td>XC2VP125</td>
<td>HL</td>
<td>43,986</td>
<td>8Kb</td>
<td>16</td>
<td>32</td>
<td>749</td>
</tr>
<tr>
<td>[5] (^c)</td>
<td>Elliptic</td>
<td>( F_{3^{97}} )</td>
<td>922</td>
<td>XC2VP100</td>
<td>HL</td>
<td>14,267</td>
<td>0</td>
<td>1</td>
<td>77</td>
<td>251</td>
</tr>
<tr>
<td>[6], [7]</td>
<td>Elliptic</td>
<td>( F_{3^{97}} )</td>
<td>922</td>
<td>XC2VP4</td>
<td>p-FSM</td>
<td>1,833</td>
<td>50Kb</td>
<td>3</td>
<td>145</td>
<td>192</td>
</tr>
<tr>
<td>[8]</td>
<td>Elliptic</td>
<td>( F_{3^{97}} )</td>
<td>922</td>
<td>XC2VP20</td>
<td>p-FSM</td>
<td>4,455</td>
<td>50Kb</td>
<td>15</td>
<td>105</td>
<td>92</td>
</tr>
<tr>
<td>[8]</td>
<td>Elliptic</td>
<td>( F_{2^{239}} )</td>
<td>956</td>
<td>XC2VP20</td>
<td>p-FSM</td>
<td>4,557</td>
<td>50Kb</td>
<td>31</td>
<td>123</td>
<td>107</td>
</tr>
<tr>
<td>[8]</td>
<td>Elliptic</td>
<td>( F_{2^{283}} )</td>
<td>1132</td>
<td>XC2VP20</td>
<td>p-FSM</td>
<td>5,350</td>
<td>50Kb</td>
<td>31</td>
<td>127</td>
<td>140</td>
</tr>
<tr>
<td>Alg. 1</td>
<td>Elliptic</td>
<td>( F_{2^{239}} )</td>
<td>956</td>
<td>XC2VP100</td>
<td>HL</td>
<td>25,487</td>
<td>0</td>
<td>16</td>
<td>84</td>
<td>41</td>
</tr>
<tr>
<td>Alg. 1</td>
<td>Elliptic</td>
<td>( F_{2^{283}} )</td>
<td>1132</td>
<td>XC2VP100</td>
<td>HL</td>
<td>37,803</td>
<td>0</td>
<td>32</td>
<td>72</td>
<td>61</td>
</tr>
</tbody>
</table>

\(^a\)HL denotes hardwired logic, M denotes dedicated microprocessor, and p-FSM denotes programmable finite state machine.

\(^b\)Controller not included in the CLB slice count.

\(^c\)The final exponentiation was not implemented.

and more complicated datapath in final exponentiation cannot be avoided so that the critical path is longer than in Alg. 1.

In Table VI, we compared our FPGA implementations of Tate pairing accelerator with peer works published in [1], [3]–[8]. The parameter \( D \) denotes the digit size of multipliers working at the accumulative stage. In [1], the pairing accelerator for the elliptic curve over \( F_{3^{97}} \) 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 \( F_{3^{6m}} \). 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 $\mathbb{F}_{2^{1\times 239}}$, 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 $\mu$s 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. So the controller is not included in CLB slice count. 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 $\mu$s. The latency of the design by Grabher et al. [3] is at least 432 $\mu$s. At the same time, our resource utilization is only 5 times larger as theirs.

Ronan et al. have implemented Tate pairing cryptosystems on hyperelliptic curves over binary fields via Xilinx XC2VP125, the same device which was used in [1], and have published the results in [4]. In their works, 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 $\mathbb{F}_{2^{239}}$ 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 $\mu$s and 43,986 slices are used. Our accelerator can run 12 times faster, whereas the resource utilization is only 37,803 slices, see Table VI.

Recently, Komursu et al. have published their implementation of the improved Duursma-Lee algorithm in [5]. The final exponentiation is excluded from their works. In their design, one $\mathbb{F}_{3^6 \times 97}$ multiplier, which contains 18 $\mathbb{F}_{3^{97}}$ multipliers and 52 $\mathbb{F}_{3^{97}}$ adders, is used. Bit-serial architecture is adopted for the $\mathbb{F}_{3^{97}}$ multiplier by which one $\mathbb{F}_{3^{97}}$ multiplication can be completed in 97 cycles. If we use Alg. 1 and choose the digit size of multiplier as $D = 16$, our accelerator can run 4.6 times as fast as theirs, see the second row in Table V. However resource utilization is almost the same in case that final exponentiation is also subtracted from our design.

At present, the most compact design in terms of FPGA implementations of pairing over cubic and binary elliptic curves was proposed by Beuchat et al. [6]–[8]. They developed a dedicated micro-code driven finite state machine (FSM) to schedule field operations in $\mathbb{F}_{3^{97}}$. Essentially, it is SPM based architecture. In addition, only one MSD serial multiplier with $D = 3$ is adopted in the arithmetic unit. Besides 1,833 CLB slices, in which the controller
has been counted, 6 Virtex-II SelectRAM+ blocks are configured to form a dual-port RAM of 194-bit wide, which allows short latency of switching data between the memory and the arithmetic unit. The depth of the dual-port RAM is around 256, so the size of memory is approximately 50Kb. The compactness of their design can be ascribed to three factors: micro-code driven FSM, single multiplier, and a dual port RAM with large word size. In our implementations, we use flip-flops instead of RAMs to store the intermediate results. That’s why it takes around 10 times as many as CLBs in our design. In [8], larger digit sizes of multipliers for binary fields and cubic fields are chosen to further shorten latency. Since our target platform is high performance reconfigurable computer, which can be used as a server, our optimization goal is definitely speed instead of area. Therefore, we exploited the parallelism to obtain faster operation speed. Our accelerator can run twice as fast as the accelerator by Beuchat et al. However their architecture is more suitable for applications with limited resources, such as consumer electronics.

In general, the speed up of our implementation of the binary elliptic Tate pairing scheme compared to the other researchers’ implementations of the cubic and hyperelliptic binary schemes is a function of multiple factors. These factors mainly include,

1) Computational efficiency at high level, which can be evaluated by counting the number of subfield multiplications in each iteration of the accumulative multiplication.

2) Parallelization for the accumulative multiplication.
   • How many subfield multiplication rounds will be taken in case of full power parallelization.
   • How many subfield multipliers will be adopted to achieve the full power parallelization.

3) Efficiency of subfield arithmetic: one advantage of the cubic elliptic scheme is that the field size can be smaller than the binary elliptic scheme, however it takes $2^m$ bits to represent an element in $\mathbb{F}_{3^m}$. In addition, the operations in $\mathbb{F}_{3^m}$ is more complicated than the ones in $\mathbb{F}_2$.

4) Choice of top architecture: There are mainly two kinds of architectures, SPM and hardwired logic as claimed previously. SPM is suitable for the cubic elliptic and the binary hyperelliptic schemes because high level operations are much more complicated than the binary elliptic case. On the other hand, hardwired logic can be chosen for the binary elliptic case due to its simplicity of group operations.

The detailed determination of the relative influence of various factors would require an implementation of all three schemes by the same team, using identical assumptions, design techniques, optimization schemes, tools, and coding style. This kind of comparison is beyond the scope of this paper, and will be addressed in the future publications.

VII. RECONFIGURABLE COMPUTING APPROACH OF TATE PAIRING OVER SELECTED BINARY FIELDS

When implementing pairing-based key management and digital signature generation or verification in a client-server scenario, fast operation speed as well as a variety of operand sizes must be supported for pairing computations executed at the server end. Flexibility can be realized via software program on the server. However the system will be overburdened in case that intensive pairing operations need to be completed in a short time. On the contrary, pure ASIC solutions of pairing can achieve fast operation speed easily, but it is difficult to switch between different security levels in real time, because of the loss of circuit flexibility. Reconfigurable computers, based on a hybrid
architecture combining a reconfigurable hardware processing unit (such as FPGAs) with a software-programmable processor, are devised to fill the gap between the hardwired technology and the general-purpose processor. They allow users to customize the reconfigurable processing unit to complete this computation-intensive task without loss of flexibility of a software solution. FPGAs are commonly chosen to be integrated into the system as reconfigurable components so that users can manipulate gate-level flip-flops, memory, and other gate logics. This makes it well suitable for applications, such as cryptography, in which intensive computations with unconventionally large operand sizes are involved. We choose SGI Altix-4700, a leading reconfigurable computer from Silicon Graphics, Inc., as the target platform for our experiments with Tate pairing cryptosystems over 8 selected binary fields ranging from \( \mathbb{F}_{2^{239}} \) to \( \mathbb{F}_{2^{557}} \) listed in Table I. Moreover, the generic IP core of pairing of variable sizes can be wrapped to form reconfigurable computing library (RCLib) via SGI methodology, so that the hardware macros can be called from a high level language, such as C, and each macro call will correspond to one instantiation of the corresponding hardware component. In the following, we first briefly introduce the hardware and software architecture of SGI reconfigurable application-specific computing (RASC). Then we provide details of porting pairing to SGI Altix 4700 in Sec. VII-B. Finally, performance and cost are demonstrated in Sec. VII-C.

A. RASC Hardware/Software Overview

The Altix 4700 series ¹ is a family of multiprocessor distributed shared memory (DSM) computer systems. All processors and memory tied together into a single logical system with special crossbar switches. This combination of processors, memory, and crossbar switches constitutes the interconnect fabric called NUMAlink. RASC blade, which is powered by dual Xilinx Virtex-4 LX200 FPGAs, 80 MB QDR SRAM and dual NUMAlink 4 ports, can be integrated into SGI Altix 4700 system. 4 working frequencies, 66 MHz, 100 MHz, 150 MHz and 200 MHz are supported. In Fig. 9, the RASC hardware blade contains two computational FPGAs, two TIO ASICs which attach to the Altix system NUMAlink interconnect directly, and a loader FPGA for loading bitstreams into the computational FPGAs. Fig. 10(a) shows a block diagram of the RASC FPGA with core services and the algorithm block. The computational FPGAs connect directly into the NUMAlink fabric via Scalable System Ports (SSP) on the TIO ASICs. The FPGA is loaded with a bitstream that contains two major functional blocks:

- The reprogrammable algorithm
- The core services that facilitate running the algorithm

There are three groups of interface signals between the algorithm block and the core services logic:

- **General algorithm control interface** provides the algorithm with a clock, reset, triggering and stepping control. When the algorithm is done, it should pulse its `alg_done` output shown in Fig. 10(b).
- **External memory interface** provides the algorithm with a simple interface to the read and write ports of two banks of SRAMs connected to the FPGA. Access to the SRAM ports by the algorithm block is specified by macros defined at the synthesis of the FPGA bitstream.

¹ Most of the contents related to Altix system is from [30]
Fig. 9. RASC blade hardware. TIO denotes peer-to-peer I/O. SSP denotes Scalable System Port. QDR denotes Quad Data Rate. DIMM denotes Dual In-line Memory Module.

Fig. 10. The diagram of core services block and algorithm block.
• **Debug port interface** contains 64 64-bit debug outputs with respect to the algorithm block and 8 optional algorithm defined registers which can be written by software and read by hardware. The algorithm block makes its internal signals visible to the GNU debugger software by connecting them to one or more debug output signals. The use of those additional control registers is determined by algorithm need.

The RASC abstraction layer provides an application programming interface (API) for the kernel device driver and the RASC hardware. It is intended to provide a similar level of support for application development as the standard open/close/read/write/io_control calls for IO peripherals.

### B. Port Tate Pairing Cryptosystems to SGI Altix 4700

We chose one of the two FPGAs, shown in Fig. 9, as our target device. We developed the generic IP core of Tate pairing in VHDL and its architecture has been proposed in previous section.

The process of generating the bitstream for our Tate pairing cryptosystems is described as follows. There are two supported ways to create a bitstream. The first method is to implement the Algorithm FPGA using the pre-synthesized core services block EDIF netlist generated by Synplicity Pro. This allows faster synthesis time. The second method, which allows more optimization between the core services and the algorithm block, is to synthesize from the top-down, including re-synthesis of the core services block source codes. In our experiments, we choose the second method.

Apart from the bitstream file, we need two configure files to register the algorithm to the device manager. The first, for core services, is generated automatically by the RASC tool. The second is the user space configure file, generated by the RASC extractor which can contain those information, such as the name of SRAM and algorithm-defined-registers referred in the application software.

By now, we can register the bitstreams of 8 field sizes to the device manager to form the library of hardware macros. Next, we will show that the FPGA-based accelerator can change the parameters of the pairing-based cryptosystem in real time. We only need consider latency for both reconfiguration between two key sizes and pairing computation for one fixed size, i.e., the communication overhead between FPGAs and SRAMs can be neglected, because the data transferring latency is much shorter than pairing computation, see Fig. 9., secondly ping-pong mode can be adopted in RASC platform which allows pipelining input, computation, and output. For the reconfiguration process, it takes approximately several milliseconds to complete programming the FPGAs. In order to perform quantitative analysis, we can have the following notations and assumptions without loss of generality:

1) The field size is chosen as 239. According to the results listed in Table V, it takes 41 \( \mu s \) to compute one pairing.

2) The reconfiguration latency is 5 \( ms \). And SGI Altix-4700 can perform around 120 pairing operations of size 239 during the reconfiguration stage.

3) Let \( nc \) be the number of consecutive pairing computations sharing the same key size.

The overhead of reconfiguration of the circuit on a single FPGA chip can be neglected in case of \( nc > 1000 \). If \( 120 \leq nc \leq 1000 \), dual FPGAs in Altix-4700 can be used to achieve seamless computations of pairing as long as the...
system can predict that the change of key size will happen in 5 ms, see Fig. 9. Then Altix-4700 can perform pairing computations on one of the two FPGA chips while reconfiguring the circuit on the other FPGA chip. However if the key sizes switch frequently, there is no satisfying solution to shorten the overhead of reconfiguration except of either using more FPGA chips or implementing operations for multiple key sizes on a single FPGA. In the latter case, the smaller digit sizes would need to be used, and thus the latency of pairing computations would have to increase.

C. Performance and cost of Tate pairing cryptosystem on SGI Altix 4700

<table>
<thead>
<tr>
<th>Underlying fields</th>
<th>Digit size of multiplier in CA</th>
<th>Frequency (MHz)</th>
<th>Algorithm block resource utilization # slices / (%)</th>
<th>Total resource Utilization # slices / (%)</th>
<th>Latency of Software LiDIA (ms)</th>
<th>Latency of SGI Altix 4700 (µs)</th>
<th>Speed-up Altix vs. Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{2^{239}}$</td>
<td>32</td>
<td>100</td>
<td>29,920 (34%)</td>
<td>41,641 (46%)</td>
<td>11.00</td>
<td>36.47</td>
<td>302</td>
</tr>
<tr>
<td>$F_{2^{241}}$</td>
<td>32</td>
<td>100</td>
<td>30,286 (34%)</td>
<td>41,989 (47%)</td>
<td>11.10</td>
<td>37.23</td>
<td>298</td>
</tr>
<tr>
<td>$F_{2^{283}}$</td>
<td>32</td>
<td>100</td>
<td>36,481 (41%)</td>
<td>48,202 (56%)</td>
<td>18.20</td>
<td>46.14</td>
<td>394</td>
</tr>
<tr>
<td>$F_{2^{353}}$</td>
<td>32</td>
<td>100</td>
<td>45,543 (51%)</td>
<td>57,264 (64%)</td>
<td>26.90</td>
<td>67.26</td>
<td>400</td>
</tr>
<tr>
<td>$F_{2^{367}}$</td>
<td>32</td>
<td>100</td>
<td>47,271 (53%)</td>
<td>58,992 (66%)</td>
<td>28.40</td>
<td>70.47</td>
<td>403</td>
</tr>
<tr>
<td>$F_{2^{379}}$</td>
<td>32</td>
<td>100</td>
<td>49,819 (56%)</td>
<td>61,540 (69%)</td>
<td>30.07</td>
<td>72.70</td>
<td>414</td>
</tr>
<tr>
<td>$F_{2^{457}}$</td>
<td>32</td>
<td>100</td>
<td>58,956 (66%)</td>
<td>70,677 (79%)</td>
<td>42.70</td>
<td>100.76</td>
<td>424</td>
</tr>
<tr>
<td>$F_{2^{557}}$</td>
<td>8</td>
<td>66</td>
<td>37,931 (43%)</td>
<td>49,652 (55%)</td>
<td>83.8</td>
<td>675.50</td>
<td>124</td>
</tr>
</tbody>
</table>

We first implemented the Tate pairing cryptosystems on the elliptic curves over the 8 binary fields listed in Table I via LiDIA. We used g++ 3.0.4 to compile the program and chose a Xeon station working at 2.8 GHz as the platform for our software implementations. Then we ported Tate pairing IP core over these 8 binary fields to SGI Altix 4700. Placing and routing are completed by Xilinx ISE. As a result, our pairing processor can work at 100 MHz for the first 7 field sizes. As the field size become larger, e.g. 557, it is difficult for EDA tools to place and route to meet timing such as 100 MHz. Instead, we decreased the circuit complexity by using smaller digit size of multipliers and loosed the timing constraint to 66 MHz for the size 557. The FPGA device integrated into RASC blade is Xilinx XC4VLX200, with totally 89,088 slices. As claimed before, the total resource utilization contains two parts, core services, which costs 11,721 slices (13%), and Tate pairing algorithm block. In Table VII, the latency of hardware computation is end-to-end. The FPGA computational time is approximately several thousand clock cycles. In contrast, the overhead of communications between SRAMs and FPGAs is only 20-to-40 clock cycles. Second, the FPGA chip need to be configured only once for several hundred pairing computations and the configuration latency is approximately 5 ms. Therefore the end-to-end latency is almost equal to the FPGA computational time.
Compared with the software (LiDIA) results, our pairing processor, ported on SGI reconfigurable platform, can run 120-to-420 times faster.

VIII. Conclusion

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 for computing pairing by which full power of parallel computations can be exploited. We performed security analysis for different curves over binary and cubic fields via a MAPLE program, by which we can choose two binary fields $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$ for our single FPGA implementations to achieve 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 techniques simplifying the final exponentiations for both algorithms are addressed. Additionally, the implementations are further optimized by optimal parameter choices for ALU. Compared with earlier implementations of Tate pairing computations based on the cubic and binary hyperelliptic schemes, our pairing processors can run 2-to-20 times faster. Furthermore, we ported the Tate pairing cryptosystems for 8 field sizes, ranging from 239 to 557, to the reconfigurable computer, SGI Altix-4700. The issues related to data communications, memory allocation and controller for the reconfigurable approach of pairing cryptosystems are discussed in details. Consequently, our pairing processor on SGI reconfigurable platform can run 120-to-420 times faster than the software implementation based on LiDIA. To the author’s knowledge, this is the first complete investigation of pairing cryptosystems over binary elliptic curves on a reconfigurable system.

ACKNOWLEDGMENT

The authors would like to thank Brian Larson, Dick Riegner, and other engineers from Silicon Graphics, Inc., for their counseling and expertise. It is difficult to imagine reaching this point without their help.

REFERENCES


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


