FPGA Accelerated Tate Pairing Cryptosystems over Binary Fields

Chang Shu ¹, Soonhak Kwon ², and Kris Gaj ¹

¹ Dept. of ECE, George Mason University
   Fairfax VA, USA
² Dept. of Mathematics, Sungkyukwan University
   Suwon, Korea
Overview

• Introduction
  – Identity based encryption (IBE) scheme
  – Pairing based cryptography

• FPGA implementations
  – Algorithms
  – Architectures
  – Timing diagram
  – Results

• Comparison with software

• Comparison with previous work for comparable schemes

• Conclusions
Identity Based Encryption Scheme

Features of IBE:

1. Use the receiver’s ID as a public key for encryption
2. A third trust authority party is involved in generating the receiver’s private key associated with one’s ID.

<table>
<thead>
<tr>
<th>Notations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>trust authority</td>
</tr>
<tr>
<td>P_TA</td>
<td>TA’s public key</td>
</tr>
<tr>
<td>S</td>
<td>TA’s private key</td>
</tr>
<tr>
<td>ID(Bob)</td>
<td>Bob’s identity</td>
</tr>
<tr>
<td>S_ID(Bob)</td>
<td>Bob’s private key</td>
</tr>
<tr>
<td>M</td>
<td>Message</td>
</tr>
<tr>
<td>C</td>
<td>Cipher text</td>
</tr>
<tr>
<td>E</td>
<td>Encryption</td>
</tr>
<tr>
<td>D</td>
<td>Decryption</td>
</tr>
</tbody>
</table>
Pairing Based Cryptography (1)

- Pairing is a map between groups, where \( e: G_1 \times G_1 \rightarrow G_2, G_1 = E(F_q) \) and \( G_2 = F_{q^k} \).

- The most important property of this map is bilinearity: \( e(aP, bQ) = e(P, Q)^{ab} \).

- In practice, Tate or Weil pairing are used.
Pairing Based Cryptography (2)

s: secret value
P: public value
$P_{TA} = s \cdot P$
public key of TA
$P_{ID(Bob)} = H_1(ID(Bob))$
Bob’s public key
$S_{ID(Bob)} = s \cdot P_{ID(Bob)}$
Bob’s private key
r: random number

\[ C = (U, V) = (rP, M + H_2(e(P_{ID(Bob)}, P_{TA})^r)) \]
\[ M = (V + H_2(e(S_{ID(Bob)}, U))) \]

By bilinearity, $e(S_{ID(Bob)}, U) = e(sP_{ID(Bob)}, rP) = e(P_{ID(Bob)}, sP)^r = e(P_{ID(Bob)}, P_{TA})^r$
Two Selected Algorithms (1)

**Algorithm 1:**

**Input:** $P = (x, y), Q = (\alpha, \beta)$
$x, y, \alpha, \beta \in F_{2^m}$

**Output:** $C = \tau(P, Q)$, where $C \in F_{2^{4m}}$

\[ C \leftarrow 1, \]
\[ \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4, v \leftarrow x^2 + 1, \theta \leftarrow \alpha \cdot v \]
\[ u \leftarrow x^2 + y^2 + b + \frac{m-1}{2} \]

for $i = 0$ to $m-1$

\[ A \leftarrow \beta + \theta + u + (\alpha + v)s + t \]
\[ C \leftarrow C^2 \]
\[ C \leftarrow C \cdot A \]
\[ \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4, u \leftarrow u + v, \]
\[ v \leftarrow v + 1, \theta \leftarrow \alpha \cdot v \]

end for

\[ C \leftarrow C^{2^m-1} \]

**Algorithm 2:**

**Input:** $P = (x, y), Q = (\alpha, \beta)$
$x, y, \alpha, \beta \in F_{2^m}$

**Output:** $C = \tau(P, Q)$, where $C \in F_{2^{4m}}$

\[ C \leftarrow 1, \alpha \leftarrow \alpha^2 + 1, \beta \leftarrow \beta^2 + 1, \]
\[ u \leftarrow y + b + 1, \theta \leftarrow \alpha \cdot v \]

for $i = 0$ to $(m-1)/2$

\[ C \leftarrow C^2, C \leftarrow C \cdot A \]
\[ \text{if } i < (m-1)/2 \text{ then} \]
\[ \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4, u \leftarrow u + v + 1, \]
\[ v \leftarrow v + 1, \theta \leftarrow \alpha \cdot v \]
\[ \text{end if} \]

end for

\[ A \leftarrow A + (\alpha^2 + v + 1) + s \]
\[ C \leftarrow C \cdot A \]
\[ C \leftarrow C^{MT}, \quad MT = (2^{2m} - 1)(2^m \mp 2^{\frac{m+1}{2}} + 1)(2^{\frac{m+1}{2}} \pm 1) \]
Two Selected Algorithms (2)

- Two main stages: accumulative multiplication and final powering
- Only 7 underlying field multiplications are needed in each iteration for both algorithms
- All these multiplications can be completed simultaneously.
- Half iterations can be saved when using Algorithm 2, however the computation for final powering is much easier for Algorithm 1.
- In our work, we use polynomial basis for both algorithms, and we choose $F_{2^{239}}$ and $F_{2^{283}}$ for our single FPGA implementations.
Top Architecture of Pairing Processor

Features:

- Hardwired logic instead of stored-programmed machine
- Iterative structure
- Register files for intermediate results
- Main controller designed as a finite state machine
- The extension field multiplier CA and Multiplier 1 are working for both stages
Two Possible Architectures for CA

6 multipliers:
1. lower latency
2. larger area
3. lower product of latency by area

3 multipliers:
1. higher latency
2. smaller area
3. higher product of latency by area
Timing Diagram for Algorithm 1

Initialization of $\theta$

Storing results to Registers

$m$ times

Accumulative multiplications

$T_1 + 2$

$...$

$...$

$0$: start

Final exponentiation

$T_1 + 2$

$...$

$...$

$(m+1)(T_1 + 2)$

$(m+1)(T_1 + 2)$

$(m+1)(T_1 + 2) + T_2 + 2$

$(m+2)(T_1 + 2) + 2(T_2 + 2)$

$(m+3)(T_1 + 2) + 3T_2 + T_3 + 8$

done

Time (clock cycles)

Notations:
- $T_1$: Latency of CA and Multiplier 1
- $T_2$: Latency of Multiplier 2
- $T_3$: Latency of Inverter
- MUL 1: Multiplier 1
- MUL 2: Multiplier 2
- CA: Special multiplier over $F_{2^{4m}}$
- INV: Inverter

Accumulative multiplications

$A \cdot B$: Multiplication over $F_{2^{2m}}$

$1 \over c_0^2 + c_0 c_1 + c_1^2$ $(c_0 + c_1(s + 1))$
Timing Diagram for Algorithm 2

Notations:
T1: Latency of CA and Multiplier 1
T2: Latency of Multiplier 2
T3: Latency of Inverter
MUL 1: Multiplier 1
MUL 2: Multiplier 2
CA: Special multiplier over $F_{2^{4m}}$
INV: Inverter

Final exponentiation

$T_4 = \frac{m+1}{2} T_{\text{ckt}} + 4 T_i + 2)$
Timing Diagrams

• The large extension field multiplier CA works for both stages, and two different sources of data

• Multiplier 1 also works for both stages

• Compared with CA and Multiplier 1, Multiplier 2 has smaller digit size.

• Only one subfield inversion is needed

• Compared with Algorithm 1, one half of iterations can be saved for Algorithm 2, however more multiplications and squares are involved.

• For Algorithm 2, CA has 4 different sources of data, i.e., more levels of multiplexers are used.
Implementation Results for GF($2^{239}$)

Target device: Xilinx XC2CP100-6FF-1704

Algorithm 1
D=32

Algorithm 2
D=16

Lower product of latency by area
Implementation Results for $GF(2^{283})$

Target Device: Xilinx XC2VP100-6FF-1704

Algorithm 1
D=16

Algorithm 2
D=32

Lower product of latency by area
Speed-up over Software

Software Platform: Intel Xeon 2.8 GHz; C++ library, LiDIA, for subfield arithmetic

Hardware Platform: Xilinx XC2VP100-6FF-1704

Algorithm 1, $D = 32$

Algorithm 2, $D = 16$

<table>
<thead>
<tr>
<th>Field</th>
<th>Speed-up (Algorithm 1)</th>
<th>Speed-up (Algorithm 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GF(2^{239})$</td>
<td>251</td>
<td>183</td>
</tr>
<tr>
<td>$GF(2^{283})$</td>
<td>297</td>
<td>156</td>
</tr>
</tbody>
</table>
Comparison with Hardware Implementation of Comparable Schemes (1)

<table>
<thead>
<tr>
<th>Comparable Schemes</th>
<th>Binary elliptic (our scheme)</th>
<th>Cubic elliptic</th>
<th>Binary hyper-elliptic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field $F_q$</td>
<td>$q = 2^m$</td>
<td>$q = 3^m$</td>
<td>$q = 2^m$</td>
</tr>
<tr>
<td>Curve</td>
<td>elliptic</td>
<td>elliptic</td>
<td>hyper-elliptic</td>
</tr>
<tr>
<td>Embedded Degree, $k$</td>
<td>4</td>
<td>6</td>
<td>12</td>
</tr>
</tbody>
</table>
## Comparison with Hardware Implementation of Comparable Schemes (2)

### MOV Security

<table>
<thead>
<tr>
<th>Elliptic Curve Discrete Logarithm Problem Over E(GF(q))</th>
<th>Menezes-Okamoto-Vanstone algorithm</th>
<th>Discrete Logarithm Problem over GF(q^k)</th>
</tr>
</thead>
</table>

**Field F_q**

<table>
<thead>
<tr>
<th>Binary elliptic</th>
<th>q = 2^m</th>
<th>k·m</th>
<th>4 m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary hyper-elliptic</td>
<td>q = 2^m</td>
<td>k·m</td>
<td>12 m</td>
</tr>
<tr>
<td>Cubic elliptic</td>
<td>q = 3^m</td>
<td>k·(log_23)·m</td>
<td>9.5 m</td>
</tr>
</tbody>
</table>
Comparison with Hardware Implementation of Comparable Schemes (3)

<table>
<thead>
<tr>
<th></th>
<th>Our Alg. 2</th>
<th>Kerin</th>
<th>Grabher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curves</td>
<td>Elliptic</td>
<td>Elliptic</td>
<td>Elliptic</td>
</tr>
<tr>
<td>Fields</td>
<td>GF(2(^{239}))</td>
<td>GF(3(^{97}))</td>
<td>GF(3(^{97}))</td>
</tr>
<tr>
<td>MOV Security</td>
<td>956</td>
<td>922</td>
<td>922</td>
</tr>
<tr>
<td>FPGA Device</td>
<td>XC2VP 100</td>
<td>XC2VP 125</td>
<td>XC2VP4 FF672</td>
</tr>
<tr>
<td>Controller</td>
<td>Hard wired logic</td>
<td>Hard wired logic</td>
<td>Microprocessor</td>
</tr>
</tbody>
</table>
Comparison with Hardware Implementation of Comparable Schemes (4)

![Graph showing latency and CLB slices comparison between Our Alg. 1 and Ronan's schemes.](image)

<table>
<thead>
<tr>
<th></th>
<th>Alg. 1</th>
<th>Ronan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curves</td>
<td>Elliptic</td>
<td>Hyper-elliptic</td>
</tr>
<tr>
<td>Fields</td>
<td>GF(2(^{283}))</td>
<td>GF(2(^{103}))</td>
</tr>
<tr>
<td>MOV Security</td>
<td>1132</td>
<td>1236</td>
</tr>
<tr>
<td>FPGA Device</td>
<td>XC2VP 100</td>
<td>XC2VP 125</td>
</tr>
<tr>
<td>Controller</td>
<td>Hardwired logic</td>
<td>Hardwired logic</td>
</tr>
</tbody>
</table>
Conclusions

• First FPGA implementation of the Tate pairing schemes for binary elliptic curves.

• Two algorithms improved, implemented and compared

• Algorithm 2 is faster, but its implementation takes more area

• Speed-ups in the range 150-300 demonstrated for Xilinx XC2VP100 vs. Xeon 2.8 GHz

• Our designs outperform existing implementation of comparable schemes in terms of the execution time by a factor 10-20, the product of latency by area by a factor 12-46.