An Automated Schedule-based Approach for the Development of Cryptoprocessors for Pairing-Based Cryptosystems

Theodore (Ted) Winograd
Rabia Shahid
Kris Gaj

Cryptographic Engineering Research Group (CERG)
http://cryptography.gmu.edu
Department of ECE, Volgenau School of Engineering,
George Mason University, Fairfax, VA, USA
Outline

- Introduction and Motivation
- Background on Pairing
  - Identity Based Encryption
  - Optimal Ate Pairing
- Design Methodology
  - Execution Unit
  - Scheduler of Field Operations for Pairing
  - Local Memory Map
  - Assembly Language Version of the Program
  - Pseudocode of Lower Level Functions
- Results and Their Analysis
- Conclusions and Future Work
Outline

- Introduction and Motivation

- Background on Pairing
  - Identity Based Encryption
  - Optimal Ate Pairing

- Design Methodology
  - Execution Unit
  - Scheduler of Field Operations for Pairing
  - Local Memory Map
  - Assembly Language Version of the Program
  - Pseudocode of Lower Level Functions

- Results and Their Analysis

- Conclusions and Future Work
Introduction

Public Key Cryptosystems

• Introduced in 1976 by Diffie and Hellman
• Different keys are used for encryption (public key) and decryption (private key)
• Usually based on number theory
• Addressed the issue of key distribution and digital signatures
Identity Based Encryption

- Proposed by Adi Shamir in 1984
- Similar to Public Key Encryption
- Boneh and Franklin developed a protocol in 2001 based on pairings
- No need for key distribution between parties

Applications of PKC’s

- Encryption/Decryption
- Identity Mapped To Keys
- No Key Exchange
Motivation

- **Challenge:**
  - Bridge the gap between cryptographers and hardware engineers for the development of hardware-friendly cryptosystems, choice of optimal parameters, and design of efficient reconfigurable hardware implementations

- **Focus:**
  - Cryptosystems based on complex finite field arithmetic at the lowest level of hierarchy such as
    - Elliptic Curve Cryptosystems
    - Pairing-based Cryptosystems
    - RSA
Motivation

- New methodology and tools
  - Engineers can use our framework to choose which pairing algorithm
  - Estimate performance

- In Galbraith, et al. “Pairings for cryptographers” 2008:
  - Instances when hardware designers have to treat pairings as black boxes
  - It is easy to make invalid assumptions

- Our proposal:
  - Cryptoprocessor is paired with a scheduler
  - Reconfigurable using a sequence of instructions based on the underlying field arithmetic
  - Can adjust to any other cryptosystem based on finite field arithmetic.
  - The overall design is implemented taking into consideration the trade-off between the minimum resource utilization and low computational time, combined with configurability of arithmetic units.
Outline

- Introduction and Motivation

- Background on Pairing
  - Identity Based Encryption
  - Optimal Ate Pairing

- Design Methodology
  - Execution Unit
  - Scheduler of Field Operations for Pairing
  - Local Memory Map
  - Assembly Language Version of the Program
  - Pseudocode of Lower Level Functions

- Results and Their Analysis

- Conclusions and Future Work
Pairing-based Cryptography

• One of the most promising directions in the theory and practice of computer and network security

• Allow schemes with special properties which cannot be provided through traditional public key cryptography

• First practical applications
  • Tax payment authentication in Sao Paulo Brazil
  • Voltage SecureMail
  • Wireless sensor networks
Applications

Identity-based encryption
• Public key of a user derived from his/her ID
• Allows the sender to encrypt a message based on the knowledge of the receiver's ID
• Used in email encryption. Two major vendors are Voltage Security and Trend Micro

Identity-based Signature
• Allows the receiver to verify the signature based on the knowledge of the sender's ID

Non-interactive two-party key agreement
• Both parties are able to easily calculate the same secret key based on the knowledge of a unique ID of the other side
• Helps when pre-distribution of keys is impractical
Choice of Parameters

Field: prime, binary, ternary

Curve: elliptic ordinary, elliptic supersingular, hyperelliptic

Embedding degree, k: 2, 3, 4, 6, 12
trade-off between security and computational complexity
(k ↗: security ↗, performance ↘)

Pairing Type: Tate, Eta, Ate, Optimized Ate, Twisted Ate, Optimal Ate

Security Level: 80, 128, 192, 256

Coordinate System: Affine, Projective
Bilinear pairing is a transformation $e$ of the form

$$e : G_1 \times G_2 \rightarrow G_T$$

$G_1$ - subgroup of a group of points on an elliptic curve $E$ over GF($q$), denoted by $E(GF(q))$

$G_2$ - subgroup of a group of points on an elliptic curve $E$ over GF($q^k$), denoted by $E(GF(q^k))$, where $k$ is called the embedding degree

$G_T$ - element of the multiplicative group of an extension field GF($q^k$)
Optimal Ate Pairing

- Operations on **quadratic extension field** instead of operations on an extension field of degree 12
- Elliptic curve point addition and line function only invoked **four** times throughout the Miller loop
- **Shortest Miller loop** length

![Diagram of operations]

- **Optimal Ate Pairing**
  - Elliptic Curve Group Operations
    - Point Addition
    - Point Doubling
    - Point Negation
  - Extension Field Operations (GF(p^{12}))
    - Multiplication
    - Squaring
    - Line function
    - Inversion
  - Tower Field Arithmetic (GF(p^{12}) to GF(p^{4}) to GF(p))
  - Prime Field Operations
    - Modular Multiplication
    - Modular Squaring
    - Modular Addition
    - Modular Subtraction
Barreto and Naehrig introduced a family of pairing-friendly elliptic curves defined by the following equation

$$E : y^2 = x^3 + b$$

where $b \neq 0$ over a prime field $\text{GF}(p)$, and

$$p = 36z^4 + 36z^3 + 24z^2 + 6z + 1$$
$$n = 36z^4 + 36z^3 + 18z^2 + 6z + 1$$
$$z = -(2^{62} + 2^{55} + 1) < 0$$
$$r = 6z + 2$$

where $n$ is the large odd prime dividing the curve order

• BN curves allow the use of sextic twist for fast computations over the point in $G_2$
Optimal Ate Pairing

Algorithm Optimal Ate Pairing over BN Curves

1: **Input:** $P = (x_P, y_P) \in E(\text{GF}(p)[n])$, $Q = (x_Q \gamma^2, y_Q \gamma^3) \in E(\text{GF}(p^{12})) \cap \text{Ker}(\pi_p - p)$ with $x_Q$ and $y_Q \in \text{GF}(p^3)$, $r = |6t + 2| = \sum_{i=0}^{s-1} r_i 2^i$

2: **Output:** $a_{opt}(Q, P) \in \text{GF}(p^{12})$

3: $T \leftarrow (x_Q \gamma^2, y_Q \gamma^3, 1)$, $f \leftarrow 1$

4: **for** $i$ from $s - 2$ **downto** 0 **do**

5: $g \leftarrow l_{(T, T)}(P), T \leftarrow 2T, f \leftarrow f^2, f \leftarrow f \cdot g$

6: **if** $(r_i = 1)$ **then**

7: $g \leftarrow l_{(T, Q)}(P), T \leftarrow T + Q, f \leftarrow f \cdot g$

8: **end if**

9: **end for**

10: $T \leftarrow -T, f \leftarrow f^{p^6}$

11: $Q_1 \leftarrow \pi_p(Q), Q_2 \leftarrow -\pi_p^2(Q)$

12: $g \leftarrow l_{(T, Q_1)}(P), T \leftarrow T + Q_1, f \leftarrow f \cdot g$

13: $g \leftarrow l_{(T, Q_2)}(P), T \leftarrow T + Q_2, f \leftarrow f \cdot g$

14: $f \leftarrow (f^{p^6 - 1})^{p^2 + 1}$

15: $f \leftarrow f^{(p^4 - p^2 + 1)/n}$

16: **return** $f$

- **Doubling step:** An elliptic curve point doubling operation and the computation of a line function $g$.
- **Addition step:** An elliptic curve point addition and the computation of a line function $g$.
- **Squaring:** Squaring of Miller variable $f$.
- **Sparse multiplication:** A multiplication of Miller variable $f$ with $g$ having only half of the non-zero coefficients.
- **Frobenius Maps and Easy exponentiation:** Intermediate operations of Miller loop and hard exponentiation.
- **Final exponentiation:** Powering the intermediate result.
Algorithm 3 Doubling Step and Line Function

1: Input: $P = (x_P, y_P) \in E(GF(p)); T = (X_T \gamma^2, Y_T \gamma^3, Z_T) \in E(GF(p^{12}))$ with $X_T$, $Y_T$ and $Z_T \in GF(p^2)$
2: Output: $2T, l_{(T,T)}(P)$
3: $B \leftarrow Y_T^2, E \leftarrow 2Y_TZ_T, C \leftarrow 3Z_T^2, D \leftarrow 2X_TY_T$
4: $A \leftarrow X_T^2, H \leftarrow 3C$
5: $F \leftarrow B + iH, G \leftarrow B - iH, J \leftarrow 4HC$
6: $g_0 \leftarrow E y_P, g_1 \leftarrow -3Ax_P, g_3 \leftarrow B + iC$
7: $I \leftarrow G^2, Z_{2T} \leftarrow 4BE$
8: $X_{2T} \leftarrow DF, Y_{2T} \leftarrow I + J$
9: return $(X_{2T} \gamma^2, Y_{2T} \gamma^3, Z_{2T}), g_0, g_1, g_3$

Lower Level Functions

- Multiplication in GF($p^2$)
- Squaring in GF($p^2$)
- Addition in GF($p^2$)
- Negation in GF($p^2$)
- Multiplication by i
Outline

- Introduction and Motivation

- Background on Pairing
  - Identity Based Encryption
  - Optimal Ate Pairing

- Design Methodology
  - Execution Unit
  - Scheduler of Field Operations for Pairing
  - Local Memory Map
  - Assembly Language Version of the Program
  - Pseudocode of Lower Level Functions

- Results and Their Analysis

- Conclusions and Future Work
Design Methodology

Cryptographer

- Main Pseudocode
- Pseudocode for lower-level functions
- Cryptographic Algorithm
- Formulas
- Memory Map
- Tmp Vars
- Top-Level Code
- Three Operand Code
- Configuration Files
- Parameters
- Equivalent Software Implementation
- HW Output
- SW Output
- Inputs
- Sequence of Instructions for the Execution Unit (VHDL Version of the Program Memory)
- Tabular Representation of the Schedule
- Assembly Language Version of the Schedule
- Total Latency
- Total Area Estimate

Hardware Designer

- Scheduler
Execution Unit

* Inputs from Local Memory

Arithmetic Unit

Local Memory Module

* From Input storage

* To Arithmetic Unit or Output storage

\[ k \]: Word size
\[ \alpha \]: Number of \( 2^k \) words/multiplier
\[ r \]: Number of Multipliers
\[ S_{src1} \]: Port 1 data (Local Memory)
\[ S_{src2} \]: Port 2 data (Local Memory)
\[ M_i \]: \( i \)-th \( 2k \)-bit word of \( M \)
Execution Unit

- Arithmetic Unit (AU)
  - Several multipliers (MULT0, MULT1, ..., MULTr−1)
  - Adder/subtractor (A/S)
- Local Memory Module
- Global Control Unit (GCU)
- Input Storage
- Output Storage

- The scheduler yields a sequence of instructions in the form of a VHDL ROM file
  - Preloaded into a program memory implemented as a ROM
  - Dispatched to the Arithmetic Unit by the Instruction Decoder

- Execution:
  - Sequence of multiple additions during one multiplication
  - Several multipliers allow for independent multiplications in parallel
  - Scheduler determines an optimal number of multipliers required to achieve either minimum execution time or minimum execution time·area product.
Scheduler of Field Operations

Cryptographer

- Main Pseudocode
- Pseudocode for lower-level functions
- Memory Map
- Tabular Representation of the Schedule
- Assembly Language Version of the Schedule
- Formulas
- Total Latency
- Total Area Estimate
- Sequence of Instructions for the Execution Unit (VHDL Version of the Program Memory)
- Equivalent Software Implementation

Hardware Designer

- Top-Level Code
- Three Operand Code
- Tmp Vars
- Configuration Files
- Parameters

Scheduler

Execution Unit

Comparison of SW/HW Output
Operation of the Scheduler

Pre-processor

Top-level Code

Configuration Files

Parameters

Three-Operand Code Functions

Scheduler

Execution Unit

VHDL Assembly

Analysis Tool

Total latency

Area Estimate
First Phase: Preprocessor

• Accepts as input a three-operand code (op3) files from the Hyperelliptic.org Explicit Formulas Database, extended with a header describing inputs, constants, and outputs, as well as their locations in the Local Memory and a description of the function

• Outputs: 3-operand code in a single flat file for the scheduler for the entire operation
Second Phase: Scheduler

- An automated scheduler written in **Python** based on compiler concepts
- Accepts as input a **three-operand code (op3)** file (format from the Hyperelliptic.org Explicit Formulas Database)
- Accepts the following additional parameters
  - **Number of multipliers** – The number of multipliers available in the circuit
  - **Assembly file** – The output file to which to store assembly code
  - **VHDL ROM file** – The output VHDL file to which the Program ROM contents will be stored
  - **CSV file** – The tabular output of the expected behavior
- Uses Directed Acyclic Graphs to optimize the inputs
Operation of the Scheduler

- Process the op3 code
- Generate a syntax tree
- Calculate the adder and multiplier depths for each node

\[
\begin{align*}
\text{ZZ1} &= Z1^2 \\
\text{t1} &= Z1 + \text{ZZ1} \\
\text{S2} &= Y2 \times \text{t1}
\end{align*}
\]

\[
\begin{array}{c|c|c}
\text{Node} & \text{Adder Depth} & \text{Multiplier Depth} \\
\hline
\text{ZZ1} & 1 & 2 \\
\text{t1} & 1 & 1 \\
\text{S2} & 0 & 1 \\
\end{array}
\]
• Modeled after Application Specific Instruction Processors (ASIPs)
  • Data Acyclic Graphs: Each node represents an operation

• Take into account some key heuristics:
  • Keep track of the ideal: the number of multiplications left is equivalent to the multiplication depth
  • If we are not ideal, process additions/subtractions for which operands are available and funnel to another adder

• Monitor variable usage
  • Variables are mapped to Local Memory locations
  • As a variable is used and no longer needed, the Local Memory location is marked as free
### Local Memory Map

<table>
<thead>
<tr>
<th>Location in LMEM</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Xp:Xp*R</td>
<td>Xp:Xp in Montgomery Domain</td>
</tr>
<tr>
<td>1</td>
<td>Yp:Yp*R</td>
<td>Yp:Yp in Montgomery Domain</td>
</tr>
<tr>
<td>2</td>
<td>Xq₀:Xq₀*R</td>
<td>Xq₀:Xq₀ in Montgomery Domain</td>
</tr>
<tr>
<td>3</td>
<td>Xq₁:Xq₁*R</td>
<td>Xq₁:Xq₁ in Montgomery Domain</td>
</tr>
<tr>
<td>4</td>
<td>Yq₀:Yq₀*R</td>
<td>Yq₀:Yq₀ in Montgomery Domain</td>
</tr>
<tr>
<td>5</td>
<td>Yq₁:Yq₁*R</td>
<td>Yq₁:Yq₁ in Montgomery Domain</td>
</tr>
<tr>
<td>6</td>
<td>Zq₀:Zq₀*R = 1:R</td>
<td>Zq₀:Zq₀ in Montgomery Domain</td>
</tr>
<tr>
<td>7</td>
<td>Zq₁:Zq₁*R = 1:R</td>
<td>Zq₁:Zq₁ in Montgomery Domain</td>
</tr>
<tr>
<td>8</td>
<td>M</td>
<td>Modulus</td>
</tr>
</tbody>
</table>

*Lines 9-52 omitted for brevity*

### Register Map

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>S = ceil(log₂r)-2</td>
</tr>
<tr>
<td>R1</td>
<td>N = ceil(log₂(-2t))-2</td>
</tr>
</tbody>
</table>
## Assembly Language: Top-level Code

<table>
<thead>
<tr>
<th>Normal to Montgomery Conversion (N2M)</th>
<th>Intermediate Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL 0 0 10</td>
<td>CALL PP_NEG</td>
</tr>
<tr>
<td>MUL 1 1 10</td>
<td>CALL PP_INV</td>
</tr>
<tr>
<td>MUL 2 2 10</td>
<td>CALL PP_FRB1</td>
</tr>
<tr>
<td>MUL 3 3 10</td>
<td>CALL PP_FRB2</td>
</tr>
<tr>
<td>MUL 4 4 10</td>
<td>CALL PP_LN_ADD_Q1</td>
</tr>
<tr>
<td>MUL 5 5 10</td>
<td>CALL PP_FG</td>
</tr>
<tr>
<td>MUL 6 6 10</td>
<td>CALL PP_LN_ADD_Q2</td>
</tr>
<tr>
<td>MUL 7 7 10</td>
<td>CALL PP_FG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Copy Q to T</th>
<th>Final Exponentiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV 15 2</td>
<td>CALL PP_FEXP1</td>
</tr>
<tr>
<td>MOV 16 3</td>
<td>CALL PP_FEXP2</td>
</tr>
<tr>
<td>MOV 17 4</td>
<td></td>
</tr>
<tr>
<td>MOV 18 5</td>
<td>MUL 21 51 21</td>
</tr>
<tr>
<td>MOV 19 6</td>
<td>MUL 22 51 22</td>
</tr>
<tr>
<td>MOV 20 7</td>
<td>MUL 23 51 23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Miller Loop</th>
<th>Montgomery to Normal Conversion (M2N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>PP_LN,DBL,SQR</td>
</tr>
<tr>
<td>CALL</td>
<td>PP_FG</td>
</tr>
<tr>
<td>BIT</td>
<td>11</td>
</tr>
<tr>
<td>JMP NC</td>
<td>M1</td>
</tr>
<tr>
<td>CALL</td>
<td>PP_LN_ADD</td>
</tr>
<tr>
<td>CALL</td>
<td>PP_FG</td>
</tr>
<tr>
<td>M1:</td>
<td>DEC R0</td>
</tr>
<tr>
<td>JMP NZ</td>
<td>L1</td>
</tr>
<tr>
<td>MUL</td>
<td>21</td>
</tr>
<tr>
<td>MUL</td>
<td>22</td>
</tr>
<tr>
<td>MUL</td>
<td>23</td>
</tr>
<tr>
<td>MUL</td>
<td>24</td>
</tr>
<tr>
<td>MUL</td>
<td>25</td>
</tr>
<tr>
<td>MUL</td>
<td>26</td>
</tr>
<tr>
<td>MUL</td>
<td>27</td>
</tr>
<tr>
<td>MUL</td>
<td>28</td>
</tr>
<tr>
<td>MUL</td>
<td>29</td>
</tr>
<tr>
<td>MUL</td>
<td>30</td>
</tr>
<tr>
<td>MUL</td>
<td>31</td>
</tr>
<tr>
<td>MUL</td>
<td>32</td>
</tr>
</tbody>
</table>
Pseudocodes of Functions

- Describes an operation among two variables of type GFP2.
  - Each of the operand: two variables in GF(p)
  - Results: stored in a tuple consisting of c0 and c1.

- Macro expansion
  - Track variables’ value
  - Convert functions used in top-level code
  - Unroll the entire sequence

```plaintext
*, GFP2, GFP2
IN GFP2 (a0, a1), (b0, b1)
OUT GFP2 (c0, c1) v0: a0 * b0
  v1: a1 * b1
  c0: v0 + v1
```
Outline

- Introduction and Motivation

- Background on Pairing
  - Identity Based Encryption
  - Optimal Ate Pairing

- Design Methodology
  - Execution Unit
  - Scheduler of Field Operations for Pairing
  - Local Memory Map
  - Assembly Language Version of the Program
  - Pseudocode of Lower Level Functions

- Results and Their Analysis

- Conclusions and Future Work
Algorithm Statistics

• Shortest ideal execution time:

\[ CC_{ideal} = \left\lfloor \frac{\#muls}{Nm} \right\rfloor \cdot CC(Nm) + CC(\#muls \mod Nm) \]

• Clock cycles to execute a batch:

\[ CC(i) = i \cdot Mrd + Mcomp + i \cdot Mwr \]

• Features of the execution unit for operand sizes of 256 bits:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of clock cycles per multiplication, ( M_{comp} )</td>
<td>60</td>
</tr>
<tr>
<td>Number of clock cycles per add/sub, ( AS_{comp} )</td>
<td>5</td>
</tr>
<tr>
<td>Number of clock cycles per move, ( CM_{ove} )</td>
<td>1</td>
</tr>
<tr>
<td>Time to read data, ( Mrd )</td>
<td>1</td>
</tr>
<tr>
<td>Time to write data, ( M_{wr} )</td>
<td>1</td>
</tr>
</tbody>
</table>

• Overhead: \( \frac{CC_{actual} - CC_{ideal}}{CC_{ideal}} \cdot 100\% \)
Algorithm Statistics

- Statistics collected for various functions in the Optimal Ate Pairing Algorithm

<table>
<thead>
<tr>
<th>Function</th>
<th>#calls</th>
<th>#mul calls</th>
<th>#add sub calls</th>
<th>#mov calls</th>
<th>total #muls</th>
<th>%muls</th>
</tr>
</thead>
<tbody>
<tr>
<td>pp_ln dbl sq</td>
<td>107</td>
<td>84</td>
<td>262</td>
<td>315</td>
<td>8988</td>
<td>58.11</td>
</tr>
<tr>
<td>pp_fg</td>
<td>113</td>
<td>48</td>
<td>122</td>
<td>135</td>
<td>5424</td>
<td>35.07</td>
</tr>
<tr>
<td>pp_ln_add</td>
<td>4</td>
<td>56</td>
<td>83</td>
<td>69</td>
<td>224</td>
<td>1.45</td>
</tr>
<tr>
<td>pp_neg</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>pp_inv</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>pp_frb1</td>
<td>1</td>
<td>8</td>
<td>12</td>
<td>12</td>
<td>8</td>
<td>0.05</td>
</tr>
<tr>
<td>pp_frb2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>0.01</td>
</tr>
<tr>
<td>pp_ln_add_q1</td>
<td>1</td>
<td>56</td>
<td>83</td>
<td>69</td>
<td>56</td>
<td>0.36</td>
</tr>
<tr>
<td>pp_ln_add_q2</td>
<td>1</td>
<td>56</td>
<td>83</td>
<td>69</td>
<td>56</td>
<td>0.36</td>
</tr>
<tr>
<td>pp_fexp1</td>
<td>1</td>
<td>364</td>
<td>912</td>
<td>898</td>
<td>364</td>
<td>2.35</td>
</tr>
<tr>
<td>pp_fexp2</td>
<td>1</td>
<td>345</td>
<td>1789</td>
<td>500</td>
<td>345</td>
<td>2.23</td>
</tr>
</tbody>
</table>
## Representative Functions

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>22665</td>
<td>39.0 ( \cdot 10^6 )</td>
<td>0</td>
<td>7222</td>
<td>12.4 ( \cdot 10^6 )</td>
<td>3</td>
<td>3058</td>
<td>5.3 ( \cdot 10^6 )</td>
<td>1719</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>11958</td>
<td>\textbf{32.4} ( \cdot 10^6 )</td>
<td>14</td>
<td>4228</td>
<td>\textbf{11.5} ( \cdot 10^6 )</td>
<td>15</td>
<td>1768</td>
<td>\textbf{4.8} ( \cdot 10^6 )</td>
<td>2712</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>9679</td>
<td>35.9 ( \cdot 10^6 )</td>
<td>60</td>
<td>4121</td>
<td>15.3 ( \cdot 10^6 )</td>
<td>28</td>
<td>1353</td>
<td>5.0 ( \cdot 10^6 )</td>
<td>3705</td>
</tr>
<tr>
<td>4</td>
<td>57</td>
<td>9685</td>
<td>45.5 ( \cdot 10^6 )</td>
<td>106</td>
<td>4069</td>
<td>19.1 ( \cdot 10^6 )</td>
<td>42</td>
<td>1161</td>
<td>5.5 ( \cdot 10^6 )</td>
<td>4698</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>9199</td>
<td>52.4 ( \cdot 10^6 )</td>
<td>134</td>
<td>3920</td>
<td>22.3 ( \cdot 10^6 )</td>
<td>62</td>
<td>1128</td>
<td>6.4 ( \cdot 10^6 )</td>
<td>5691</td>
</tr>
<tr>
<td>6</td>
<td>107</td>
<td>9076</td>
<td>60.7 ( \cdot 10^6 )</td>
<td>175</td>
<td>3934</td>
<td>26.3 ( \cdot 10^6 )</td>
<td>98</td>
<td>1138</td>
<td>7.6 ( \cdot 10^6 )</td>
<td>6684</td>
</tr>
<tr>
<td>7</td>
<td>135</td>
<td>9036</td>
<td>69.4 ( \cdot 10^6 )</td>
<td>209</td>
<td>3864</td>
<td>29.7 ( \cdot 10^6 )</td>
<td>117</td>
<td>1120</td>
<td>8.6 ( \cdot 10^6 )</td>
<td>7677</td>
</tr>
<tr>
<td>8</td>
<td>155</td>
<td>8900</td>
<td>77.2 ( \cdot 10^6 )</td>
<td>243</td>
<td>3884</td>
<td>33.7 ( \cdot 10^6 )</td>
<td>145</td>
<td>1115</td>
<td>9.7 ( \cdot 10^6 )</td>
<td>8670</td>
</tr>
<tr>
<td>9</td>
<td>180</td>
<td>8912</td>
<td>86.1 ( \cdot 10^6 )</td>
<td>289</td>
<td>3932</td>
<td>38.0 ( \cdot 10^6 )</td>
<td>145</td>
<td>1117</td>
<td>10.8 ( \cdot 10^6 )</td>
<td>9663</td>
</tr>
<tr>
<td>10</td>
<td>197</td>
<td>8748</td>
<td>93.2 ( \cdot 10^6 )</td>
<td>311</td>
<td>3917</td>
<td>41.7 ( \cdot 10^6 )</td>
<td>177</td>
<td>1098</td>
<td>11.7 ( \cdot 10^6 )</td>
<td>10656</td>
</tr>
<tr>
<td>11</td>
<td>213</td>
<td>8657</td>
<td>100.8 ( \cdot 10^6 )</td>
<td>331</td>
<td>\textbf{3842}</td>
<td>44.8 ( \cdot 10^6 )</td>
<td>177</td>
<td>\textbf{1096}</td>
<td>12.8 ( \cdot 10^6 )</td>
<td>11649</td>
</tr>
<tr>
<td>12</td>
<td>232</td>
<td>\textbf{8604}</td>
<td>108.8 ( \cdot 10^6 )</td>
<td>370</td>
<td>3914</td>
<td>49.5 ( \cdot 10^6 )</td>
<td>226</td>
<td>\textbf{1096}</td>
<td>13.9 ( \cdot 10^6 )</td>
<td>12642</td>
</tr>
</tbody>
</table>
Algorithm Statistics

• We calculated $t$ and $t \cdot \text{area}$ for each step in the optimal ate pairing.

• To calculate $t$ for the Miller loop, we used a value of $s = 108$, with the number of doublings $s - 1 = 107$.

• Memory locations required stayed relatively constant regardless of the choice of $Nm$.

<table>
<thead>
<tr>
<th>$Nm$</th>
<th>Overhead</th>
<th>$t$ [cycles]</th>
<th>$t \cdot \text{area}$ [cycles-LUTs]</th>
<th>area [LUTs]</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1%</td>
<td>972,549</td>
<td>1678.1 $\cdot$ 10^6</td>
<td>1719</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>10%</td>
<td>546,363</td>
<td>1481.7 $\cdot$ 10^6</td>
<td>2712</td>
<td>1.78</td>
</tr>
<tr>
<td>3</td>
<td>40%</td>
<td>475,150</td>
<td>1760.4 $\cdot$ 10^6</td>
<td>3705</td>
<td>2.05</td>
</tr>
<tr>
<td>4</td>
<td>63%</td>
<td>427,753</td>
<td>2009.6 $\cdot$ 10^6</td>
<td>4698</td>
<td>2.27</td>
</tr>
<tr>
<td>5</td>
<td>96%</td>
<td>424,229</td>
<td>2414.3 $\cdot$ 10^6</td>
<td>5691</td>
<td>2.29</td>
</tr>
<tr>
<td>6</td>
<td>124%</td>
<td>416,059</td>
<td>2780.9 $\cdot$ 10^6</td>
<td>6684</td>
<td>2.34</td>
</tr>
<tr>
<td>7</td>
<td>149%</td>
<td>407,957</td>
<td>3129.1 $\cdot$ 10^6</td>
<td>7677</td>
<td>2.39</td>
</tr>
<tr>
<td>8</td>
<td>178%</td>
<td>408,904</td>
<td>3545.2 $\cdot$ 10^6</td>
<td>8670</td>
<td>2.38</td>
</tr>
<tr>
<td>9</td>
<td>201%</td>
<td>403,952</td>
<td>3903.4 $\cdot$ 10^6</td>
<td>9663</td>
<td>2.41</td>
</tr>
<tr>
<td>10</td>
<td>228%</td>
<td>405,920</td>
<td>4325.5 $\cdot$ 10^6</td>
<td>10656</td>
<td>2.40</td>
</tr>
<tr>
<td>11</td>
<td>247%</td>
<td><strong>399,837</strong></td>
<td>4657.7 $\cdot$ 10^6</td>
<td>11649</td>
<td>2.43</td>
</tr>
<tr>
<td>12</td>
<td>274%</td>
<td>403,993</td>
<td>5119.9 $\cdot$ 10^6</td>
<td>12642</td>
<td>2.41</td>
</tr>
</tbody>
</table>
## Comparisons

<table>
<thead>
<tr>
<th>Reference</th>
<th>Resources [Slices, DSPs]</th>
<th>Freq [MHz]</th>
<th>Cycles $\times 10^3$</th>
<th>Time [ms]</th>
<th>Time $\times$ Area [ms $\times$ Slices]</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work</td>
<td>1636, 36</td>
<td>227</td>
<td>546</td>
<td>2.405</td>
<td>3935</td>
</tr>
<tr>
<td>Yao et al. 2012</td>
<td>5237, 64</td>
<td>210</td>
<td>78</td>
<td>0.338</td>
<td>1770</td>
</tr>
<tr>
<td>Ghosh et al. 2013</td>
<td>5163, 144</td>
<td>166</td>
<td>62</td>
<td>0.375</td>
<td>1926</td>
</tr>
<tr>
<td>Sghaier et al. 2015</td>
<td>5976, 30</td>
<td>145</td>
<td>80</td>
<td>0.552</td>
<td>3299</td>
</tr>
<tr>
<td>Cheung et al. 2011</td>
<td>7032, 32</td>
<td>250</td>
<td>143</td>
<td>0.572</td>
<td>4022</td>
</tr>
<tr>
<td>Fan et al. 2012</td>
<td>4014, 42</td>
<td>210</td>
<td>245</td>
<td>1.167</td>
<td>4684</td>
</tr>
</tbody>
</table>
• Comparison implementations:
  • Equivalent functionality and security level
  • Virtex 6 and Zynq-7000 have similar slice and DSP architecture
  • Our design with 2 Montgomery multipliers has the smallest resource utilization
  • Fourth smallest time · area even without optimizations
  • Our overhead is 10%
Outline

- Introduction and Motivation

- Background on Pairing
  - Identity Based Encryption
  - Optimal Ate Pairing

- Design Methodology
  - Execution Unit
  - Scheduler of Field Operations for Pairing
  - Local Memory Map
  - Assembly Language Version of the Program
  - Pseudocode of Lower Level Functions

- Results and Their Analysis

- Conclusions and Future Work
Conclusions and Future Work

• Successful methodology:
  • Allows for choosing an optimal number of multipliers in the general-purpose execution unit
  • Was applied to the specific case of Optimal Ate Pairing.
  • Supports any elliptic curve or paring-based cryptographic algorithms that can be described using three-operand code

• Future Work
  • Extending the scheduler to support multiple algorithms and execution units
  • More modern ASIP techniques
  • New architectures to support Residue Number System, Lazy Reduction, Karatsuba multiplication
  • Pipelined adders
Questions

http://clipart-library.com/creative-commons-clipart.html