Hardware implementation of the arithmetic of fields of characteristic 2 and 3

Nicolas Estibals
CARAMEL project-team, LORIA, Nancy Université / CNRS / INRIA
Nicolas.Estibals@loria.fr
Introduction

Curve-based cryptography relies on finite fields

- Among them: small characteristic fields (2 and 3)
  - Example: pairings on supersingular elliptic curves

Need for hardware implementations

- Embedded systems (RFID, smart card, sensors, ...)
- High-performance cryptographic coprocessor (bank servers, ...)
  - No native support in CPUs (Partial support in Intel AVX instruction set)
Introduction

- Curve-based cryptography relies on finite fields
  - Among them: small characteristic fields (2 and 3)
    - Example: pairings on supersingular elliptic curves

- Need for hardware implementations
  - Embedded systems (RFID, smart card, sensors, ...)
  - High-performance cryptographic coprocessor (bank servers, ...)
    - No native support in CPUs (Partial support in Intel AVX instruction set)

- Reconfigurable hardware (FPGA)
  - great prototyping platform
  - flexibility when increasing security is needed
Introduction

Curve-based cryptography relies on finite fields

- Among them: small characteristic fields (2 and 3)
  - Example: pairings on supersingular elliptic curves

Need for hardware implementations

- Embedded systems (RFID, smart card, sensors, ...)
- High-performance cryptographic coprocessor (bank servers, ...)
  - No native support in CPUs (Partial support in Intel AVX instruction set)

Reconfigurable hardware (FPGA)

- great prototyping platform
- flexibility when increasing security is needed

Join work with:

- Jean-Luc Beuchat, LCIS, University of Tsukuba, Japan.
- Jérémie Detrey, CARAMEL project-team, INRIA Nancy Grand-Est, France.
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks
  - small embedded multipliers
  - etc.
  - Inside a logic cell:
    - connections to the routing matrix
    - programmable lookup-tables
      - 4 inputs, 1 output
      - 6 inputs, 1 output
      - 6 inputs, 2 outputs
    - optional registers
      - free pipelining
    - more logic for fast carry-propagation
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells

![Diagram of FPGA architecture]
FPGAs are composed of:

- programmable logic cells
- a configurable routing matrix
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells

- Inside a logic cell:
  - connections to the routing matrix
  - programmable lookup-tables:
    - 4 inputs, 1 output
    - 6 inputs, 1 output
    - 6 inputs, 2 outputs
  - optional registers:
    - free pipelining
  - more logic for fast carry-propagation
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks

![Diagram of FPGA architecture with connections and lookup-tables](image-url)
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks
  - small embedded multipliers
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks
  - small embedded multipliers
  - etc.
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks
  - small embedded multipliers
  - etc.

- Inside a logic cell:
  - connections to the routing matrix
  - programmable lookup-tables
    - 4 inputs, 1 output
    - 6 inputs, 1 output
    - 6 inputs, 2 outputs
FPGA architecture

- **FPGAs** are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks
  - small embedded multipliers
  - etc.

- **Inside a logic cell:**
  - connections to the routing matrix
  - programmable lookup-tables
    - 4 inputs, 1 output
    - 6 inputs, 1 output
    - 6 inputs, 2 outputs
  - optional registers
    - free pipelining
FPGA architecture

FPGAs are composed of:
- programmable logic cells
- a configurable routing matrix
- input/output cells
- embedded memory blocks
- small embedded multipliers
- etc.

Inside a logic cell:
- connections to the routing matrix
- programmable lookup-tables
  - 4 inputs, 1 output
  - 6 inputs, 1 output
  - 6 inputs, 2 outputs
- optional registers
  - free pipelining
- more logic for fast carry-propagation
Outline of the talk

▶ Small characteristic finite fields

▶ Multiplication algorithms and hardware implementation

▶ A finite field coprocessor

▶ Finite fields of composite extension degree

▶ Conclusion
Representation of the elements

- Small characteristic: $p = 2$ or $3$

- Polynomial basis:
  - $\mathbb{F}_{p^m} = \mathbb{F}_p[x]/(f(x))$
  - $f(x)$ irreducible polynomial of degree $m$
  - $\mathbb{F}_{p^m}$ represented by $\mathbb{F}_p[x]^{\leq (m-1)}$
  - Reduction modulo $f(x)$ possibly required
Representation of the elements

- Small characteristic: $p = 2$ or $3$

- Polynomial basis:
  - $\mathbb{F}_{p^m} = \mathbb{F}_p[x]/(f(x))$
  - $f(x)$ irreducible polynomial of degree $m$
  - $\mathbb{F}_{p^m}$ represented by $\mathbb{F}_p[x]^{\leq(m-1)}$
  - Reduction modulo $f(x)$ possibly required

- Operations in the field
  - Addition
  - Frobenius automorphism: $(.)^p$
  - Multiplication
  - Inversion
    - Itoh & Tsujii algorithm (Fermat’s little theorem)
    - or Extended Euclidean algorithm
Representation of the elements

- Small characteristic: $p = 2$ or $3$

- Polynomial basis:
  - $\mathbb{F}_{p^m} = \mathbb{F}_p[x]/(f(x))$
  - $f(x)$ irreducible polynomial of degree $m$
  - $\mathbb{F}_{p^m}$ represented by $\mathbb{F}_p[x]/\leq(m-1)$
  - Reduction modulo $f(x)$ possibly required

- Operations in the field
  - Addition
  - Frobenius automorphism: $(.)^p$
  - Multiplication
  - Inversion
    - Itoh & Tsujii algorithm (Fermat’s little theorem)
    - or Extended Euclidean algorithm
Hardware implementation of addition

\[
\begin{align*}
\text{Add coefficient-wise} \\
\text{Representation of coefficient} \\
\text{• } F_2: \text{two values } \rightarrow \text{one wire, addition is the XOR boolean operator} \\
\text{• } F_3: \text{three values } \rightarrow \text{borrow-save representation on two wires} \\
\end{align*}
\]
Hardware implementation of addition

Add coefficient-wise

\[ a_{m-1} b_{m-1} a_{m-2} b_{m-2} \]
Hardware implementation of addition

Add coefficient-wise

Representation of coefficient
- \( \mathbb{F}_2 \): two values → one wire, addition is the XOR boolean operator
- \( \mathbb{F}_3 \): three values → borrow-save representation on two wires
  - Opposite of a coefficient: swap the two wires
Computation of the Frobenius automorphism

\[ a^p \equiv (a_{m-1}x^{m-1} + \cdots + a_1 x + a_0)^p \pmod{f(x)} \]

Raising to the \( p \)-th power

Since \( (p^i) \equiv 0 \pmod{p} \) when \( i \neq 0 \), non-linear terms disappear

Need reduction

reduce each \( x^p \cdot i \cdot \text{linear combination of the coefficients} \)

\( f \) with low Hamming weight

tri- or pentanomials

each coefficient of the results is the sum of few coefficients

Hardware implementation

Selection of coefficient is free (just wiring!)

Same cost as for a few additions

Depending on LUTs’ size, one LUT per coefficient might be enough
Computation of the Frobenius automorphism

\[ a^p \equiv a_{m-1}x^{p(m-1)} + \cdots + a_1x^p + a_0 \pmod{f(x)} \]

- **Raising to the** \( p \)-**th power**

- **Linear operation**
  - Since \( \binom{p}{i} \equiv 0 \pmod{p} \) when \( i \neq 0 \), non-linear terms disappear

- **Need reduction**
  - reduce each \( x^{p \cdot i} \)
  - linear combination of the coefficients
Computation of the Frobenius automorphism

\[ a^p \equiv (a_{\sigma_{m-1}(0)} + a_{\sigma_{m-1}(1)} + \cdots)x^{p-1} + \cdots \pmod{f(x)} \]

- Raising to the \( p \)-th power
- Linear operation
  - Since \( \binom{p}{i} \equiv 0 \pmod{p} \) when \( i \neq 0 \), non-linear terms disappear
- Need reduction
  - reduce each \( x^{p^i} \)
  - linear combination of the coefficients
- \( f \) with low Hamming weight
  - tri- or pentanomials
  - each coefficient of the results is the sum of few coefficients
Computation of the Frobenius automorphism

\[ a^p \equiv (a_{\sigma_{m-1}(0)} + a_{\sigma_{m-1}(1)} + \cdots) x^{p-1} + \cdots \quad (\text{mod } f(x)) \]

- **Raising to the** \( p \)-**th power**

- **Linear** operation
  - Since \( \binom{p}{i} \equiv 0 \pmod{p} \) when \( i \neq 0 \), non-linear terms disappear

- **Need reduction**
  - reduce each \( x^{p \cdot i} \)
  - linear combination of the coefficients

- **\( f \) with low Hamming weight**
  - tri- or pentanomials
  - each coefficient of the results is the sum of few coefficients

- **Hardware** implementation
  - Selection of coefficient is free (just wiring!)
  - Same cost as for a few additions
  - Depending on LUTs’ size, one LUT per coefficient might be enough
Itoh & Tsujii algorithm

Fermat’s little theorem

\[ a^{-1} \equiv a^{p^m-2} \pmod{f} \]
Fermat’s little theorem

\[ a^{-1} \equiv a^{p^m-2} \pmod{f} \]

Computation of \( a^{p^m-2} \) only needs:

- \((m - 1)\) Frobenius automorphism applications
- few multiplications
- an inversion in \( \mathbb{F}_p \)
Fermat’s little theorem

\[ a^{-1} \equiv a^{p^m - 2} \pmod{f} \]

Computation of \( a^{p^m - 2} \) only needs:

- \((m - 1)\) Frobenius automorphism applications
- few multiplications
- an inversion in \( \mathbb{F}_p \to \text{identity} \) for \( p = 2 \) or 3

No need for supplementary hardware
Itoh & Tsujii algorithm

- Fermat’s little theorem
  \[ a^{-1} \equiv a^{p^m - 2} \pmod{f} \]

- Computation of \( a^{p^m - 2} \) only needs:
  - \((m - 1)\) Frobenius automorphism applications
  - few multiplications
  - an inversion in \( \mathbb{F}_p \rightarrow \text{identity} \) for \( p = 2 \) or 3

- No need for supplementary hardware
Outline of the talk

▶ Small characteristic finite fields

▶ Multiplication algorithms and hardware implementation

▶ A finite field coprocessor

▶ Finite fields of composite extension degree

▶ Conclusion
Naive algorithm

- Schoolbook algorithm
  - express each partial product
  - add them

\[
a \cdot b \mod f
\]
Naive algorithm

- Schoolbook algorithm
  - express each partial product
  - add them

- Reduction modulo $f$ needed
Naive algorithm

- **Schoolbook algorithm**
  - express each partial product
  - add them

- **Reduction modulo** $f$ needed

- **Reduce** each partial product sequentially

\[
\begin{align*}
    a \cdot b \\[2pt]
    a \cdot b_0 \\[2pt]
    a \cdot b_1 \\[2pt]
    a \cdot b_{m-2} \\[2pt]
    a \cdot b_{m-1} \cdot x \pmod{f} \\
    a \cdot b \pmod{f}
\end{align*}
\]
Naive algorithm

- **Schoolbook algorithm**
  - express each partial product
  - add them

- Reduction modulo $f$ needed

- **Reduce** and **accumulate** each partial product sequentially

\[ a \cdot b \mod f \]
Parallel-serial multiplier

- Operand \(a\) is treated in parallel
- Operand \(b\) is treated \(D\) coefficients per cycle
- Need \(\lceil m/D \rceil\) cycles to complete the product
Karatsuba algorithm

\[ A \cdot B = A \cdot B \]

**3-way Karatsuba**

- split operands in three parts
- only 6 subproducts needed
Karatsuba algorithm

\[ A_H B_H X^{2n} + (A_H B_L + A_L B_H) X^n + A_L B_L \]
Karatsuba algorithm

\[ A_H B_H X^{2n} + (A_H B_L + A_L B_H)X^n + A_L B_L \]

\[ ab' + a'b = (a + a')(b + b') - ab - a'b' \]

\[ A_H B_H X^{2n} + ((A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L)X^n + A_L B_L \]
Karatsuba algorithm

\[ A_H \cdot B_H + (A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L \]
Karatsuba algorithm

\[ A_H \cdot B_H + (A_H + A_L) \cdot (B_H + B_L) - A_H B_H - A_L B_L \]

\[ a \cdot b' + a' \cdot b = (a + a') \cdot (b + b') - ab - a'b' \]
Karatsuba algorithm

\[ A \cdot B = (A_H + A_L)(B_H + B_L) - A_HB_H - A_LB_L \]

\[ ab' + a'b = (a + a')(b + b') - ab - a'b' \]
Karatsuba algorithm

3-way Karatsuba

- split operands in three parts
- only 6 subproducts needed
Odd-even split for Karatsuba multiplication

\[ A \times B = (A_0 A_1) \times (B_0 B_1) \]

\[ A \times B = (A_0 B_1 + A_1 B_0) \times 2 + A_0 B_0 \]

\[ A \times B = (A_0 B_1 + A_1 B_0) \times 2 + (A_0 + A_1)(B_0 + B_1) - A_0 B_0 - A_1 B_1 \]
Odd-even split for Karatsuba multiplication

\[(A_O B_O X^2 + A_E B_E) + X(A_O B_E + A_E B_O)\]
Odd-even split for Karatsuba multiplication

\[(A_OB_OX^2 + A_EB_E) + X(A_OB_E + A_EB_O)\]

\[ab' + a'b = (a + a')(b + b') - ab - a'b'\]

\[(A_OB_OX^2 + A_EB_E) + X((A_O + A_E)(B_O + B_E) - A_OB_O - A_EB_E)\]
Odd-even split for Karatsuba multiplication

\[(A \cdot B) = (A_O + A_E)(B_O + B_E) - A_OB_O - A_EB_E \]

\[ab + a'b = (a+ a')(b+ b') - ab - a'b' \]
Odd-even split for Karatsuba multiplication

\[ A \cdot B \]

\[ (A_O B_O X_2 + A_E B_E) + X(A_O B_E + A_E B_O) \]

\[ A \cdot B \]

\[ ab + a'b = (a + a')(b + b') - ab - a'b' \]

\[ (A_O B_O X_2 + A_E B_E) + X((A_O + A_E)(B_O + B_E) - A_O B_O - A_E B_E) \]
Fully parallel pipelined Karatsuba multiplier

Karatsuba-like algorithm:
- split the operands
- compute the subproducts
- recompose the result

Fully parallel evaluation of the subproducts

Diagram:
- Splitter
- Recomposer
- $A \cdot B$
Fully parallel pipelined Karatsuba multiplier

Karatsuba-like algorithm:
- split the operands
- compute the subproducts
- recompose the result

Fully parallel evaluation of the subproducts

Recursive scheme
- eventually use different multiplication algorithms
- end with the quadratic paper-and-pencil algorithm
Fully parallel pipelined Karatsuba multiplier

▶ Karatsuba-like algorithm:
  • split the operands
  • compute the subproducts
  • recompose the result

▶ Fully parallel evaluation of the subproducts

▶ Recursive scheme
  • eventually use different multiplication algorithms
  • end with the quadratic paper-and-pencil algorithm

▶ Pipelined
  • with the help of optional registers
  • cut the critical path
  • increase the frequency
Fully parallel pipelined Karatsuba multiplier

▶ Karatsuba-like algorithm:
  • split the operands
  • compute the subproducts
  • recompose the result

▶ Fully parallel evaluation of the subproducts

▶ Recursive scheme
  • eventually use different multiplication algorithms
  • end with the quadratic paper-and-pencil algorithm

▶ Pipelined
  • with the help of optional registers
  • cut the critical path
  • increase the frequency

▶ Final reduction modulo $f$
An example of multipliers over $\mathbb{F}_{3^{239}}$

$\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \rightarrow \sim 380$-bit field
An example of multipliers over $\mathbb{F}_{3^{239}}$

- $\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \rightarrow \sim 380$-bit field

- Recursion choice

<table>
<thead>
<tr>
<th>Polynomial size</th>
<th>Used algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>239</td>
<td>3-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>80</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>40</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>20</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>10</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>5</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>3</td>
<td>quadratic multiplication</td>
</tr>
</tbody>
</table>

- Post-place-and-route estimation for Xilinx Virtex-II Pro
  - $\sim 50000$ slices ($2$ LUTs $4 \rightarrow 1$ per slice)
  - $200$ MHz
  - Computes $200 \cdot 10^6$ products per second
  - $4000$ products per second and per slice

- Comparison with parallel-serial multiplier with $D = 16$
  - $\sim 8700$ slices
  - $180$ MHz
  - Computes $12 \cdot 10^6$ products per second
  - $1400$ products per second and per slice
An example of multipliers over $\mathbb{F}_{3^{239}}$

$\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \rightarrow \sim 380$-bit field

Recursion choice

<table>
<thead>
<tr>
<th>Polynomial size</th>
<th>Used algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>239</td>
<td>3-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>80</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>40</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>20</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>10</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>5</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>3</td>
<td>quadratic multiplication</td>
</tr>
</tbody>
</table>

Choose to have 7 pipeline stages
An example of multipliers over $\mathbb{F}_{3^{239}}$

$\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \rightarrow \sim 380$-bit field

Recursion choice

<table>
<thead>
<tr>
<th>Polynomial size</th>
<th>Used algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>239</td>
<td>3-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>80</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>40</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>20</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>10</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>5</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>3</td>
<td>quadratic multiplication</td>
</tr>
</tbody>
</table>

Choose to have 7 pipeline stages

Post-place-and-route estimation for Xilinx Virtex-II Pro

- $\sim 50000$ slices (2 LUTs 4 $\rightarrow$ 1 per slice)
- 200 MHz
- computes $200 \cdot 10^6$ products per second
An example of multipliers over $\mathbb{F}_{3^{239}}$

$\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \rightarrow \sim 380$-bit field

Recursion choice

<table>
<thead>
<tr>
<th>Polynomial size</th>
<th>Used algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>239</td>
<td>3-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>80</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>40</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>20</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>10</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>5</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>3</td>
<td>quadratic multiplication</td>
</tr>
</tbody>
</table>

Choose to have 7 pipeline stages

Post-place-and-route estimation for Xilinx Virtex-II Pro
- $\sim 50000$ slices (2 LUTs 4 $\rightarrow$ 1 per slice)
- 200 MHz
- computes $200 \cdot 10^6$ products per second

Comparison with parallel-serial multiplier with $D = 16$
- $\sim 8700$ slices
- 180 MHz
- computes $12 \cdot 10^6$ products per second
An example of multipliers over $\mathbb{F}_{3^{239}}$

- $\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \rightarrow \sim 380$-bit field

- Recursion choice

<table>
<thead>
<tr>
<th>Polynomial size</th>
<th>Used algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>239</td>
<td>3-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>80</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>40</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>20</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>10</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>5</td>
<td>2-way Karatsuba with odd-even trick</td>
</tr>
<tr>
<td>3</td>
<td>quadratic multiplication</td>
</tr>
</tbody>
</table>

- Choose to have 7 pipeline stages

- Post-place-and-route estimation for Xilinx Virtex-II Pro
  - $\sim 50000$ slices (2 LUTs $4 \rightarrow 1$ per slice)
  - 200 MHz
  - computes $200 \cdot 10^6$ products per second
  - 4000 products per second and per slice

- Comparison with parallel-serial multiplier with $D = 16$
  - $\sim 8700$ slices
  - 180 MHz
  - computes $12 \cdot 10^6$ products per second
  - 1400 products per second and per slice
Outline of the talk

▶ Small characteristic finite fields

▶ Multiplication algorithms and hardware implementation

▶ A finite field coprocessor

▶ Finite fields of composite extension degree

▶ Conclusion
Designing a finite field coprocessor

- Determine the specific needs of operations of your computation
Designing a finite field coprocessor

- Determine the **specific needs** of operations of your computation
- **Example:** final exponentiation in pairing computation
Designing a finite field coprocessor

► Determine the **specific needs** of operations of your computation

► **Example**: final exponentiation in pairing computation
  - Low **silicon footprint** design

![Diagram of Register file and Unified operator]

- **Register file**
- **Unified operator**
  - addition / subtraction
  - Frobenius (\(\cdot^3\))

\[\text{Parallel–serial multiplier} \quad D \text{ coeffs / cycle} \quad \lceil m / D \rceil \text{ cycles / product}\]
Designing a finite field coprocessor

- Determine the **specific needs** of operations of your computation

- **Example**: final exponentiation in pairing computation
  - Low **silicon footprint** design
  - Many multiplications

```
Parallel–serial multiplier
D coeffs / cycle
⌈m/D⌉ cycles / product

Unified operator
addition / subtraction
Frobenius (·)^3

Register file
```

N. Estibals — Hardware implementation of the arithmetic of fields of characteristic 2 and 3
Designing a finite field coprocessor

- Determine the **specific needs** of operations of your computation

- **Example:** final exponentiation in pairing computation
  - Low **silicon footprint** design
  - Many multiplications
  - Long **chains of Frobenius** automorphism application

```
Register file
Unified operator
  addition / subtraction
  Frobenius (·)^3
  double Frobenius (·)^9
  feedback loop
Parallel–serial multiplier
  D coeffs / cycle
  \[ \lceil m/D \rceil \] cycles / product
```
Designing a finite field coprocessor

- Determine the **specific needs** of operations of your computation

**Example:** final exponentiation in pairing computation

- Low **silicon footprint** design
- Many multiplications
- Long chains of Frobenius automorphism application
- Only one inversion

![Diagram](image)

- **Register file**
- **Unified operator**
  - addition / subtraction
  - Frobenius $(\cdot)^3$
  - double Frobenius $(\cdot)^9$
  - feedback loop
- **Parallel–serial multiplier**
  - $D$ coeffs / cycle
  - $\lceil m/D \rceil$ cycles / product
Detailed architecture of the coprocessor (char. 3)
Detailed architecture of the coprocessor (char. 3)
Detailed architecture of the coprocessor (char. 3)
Detailed architecture of the coprocessor (char. 3)
Outline of the talk

▶ Small characteristic finite fields

▶ Multiplication algorithms and hardware implementation

▶ A finite field coprocessor

▶ Finite fields of composite extension degree

▶ Conclusion
Composite extension degree

- Needed field might have a composite extension degree

- Tower field construction:
  - \( \mathbb{F}_{p^m \cdot n} = \mathbb{F}_{p^m}[y]/(g(y)) \)
  - \( g \) irreducible polynomial of degree \( n \)
Composite extension degree

- Needed field might have a composite extension degree

- **Tower field** construction:
  - $\mathbb{F}_{p^m \cdot n} = \mathbb{F}_{p^m}[y]/(g(y))$
  - $g$ irreducible polynomial of degree $n$

- Reducing the **datapath**
  - design a **coprocessor** for $\mathbb{F}_{p^m}$
  - **program it** to implement arithmetic of $\mathbb{F}_{p^m \cdot n}$
  - reduce **area** of the design

- **Operations**
  - same algorithms
  - coefficients are now in $\mathbb{F}_{p^m}$
Some other multiplication algorithms

- Subproducts between field elements, not polynomials
  - no overlapping at reconstruction step

- Apply reduction modulo $g$
  - some subproducts may not be needed after reconstruction step
Some other multiplication algorithms

- Subproducts between field elements, not polynomials
  - no overlapping at reconstruction step

- Apply reduction modulo $g$
  - some subproducts may not be needed after reconstruction step

- Toom–Cook algorithms
  - evaluate and interpolate at some points
  - $\mathbb{F}_p$ does not provide enough interpolation points
  - hardly usable in this case
Some other multiplication algorithms

- Subproducts between field elements, not polynomials
  - no overlapping at reconstruction step

- Apply reduction modulo $g$
  - some subproducts may not be needed after reconstruction step

- Toom–Cook algorithms
  - evaluate and interpolate at some points
  - $\mathbb{F}_p$ does not provide enough interpolation points
  - hardly usable in this case

- CRT-based algorithms
  - evaluate the product modulo some irreducible polynomials
  - reconstruct the result thanks to CRT
Some other multiplication algorithms

- Subproducts between field elements, not polynomials
  - no overlapping at reconstruction step

- Apply reduction modulo $g$
  - some subproducts may not be needed after reconstruction step

- Toom–Cook algorithms
  - evaluate and interpolate at some points
  - $\mathbb{F}_p$ does not provide enough interpolation points
  - hardly usable in this case

- CRT-based algorithms
  - evaluate the product modulo some irreducible polynomials
  - reconstruct the result thanks to CRT

- Montgomery’s Karatsuba-like formulae
  - ad hoc formulae for degree 4, 5 and 6 polynomials

- Algorithmic search for optimal formulae
  - Work in progress with J. Detrey, R. Barbulescu and P. Zimmermann
Choosing multiplication algorithm

* Evaluate the cost of the different algorithms
  * choice depend on the hardware implementation of $\mathbb{F}_{p^m}$
  * additions not always negligible
Choosing multiplication algorithm

- Evaluate the cost of the different algorithms
  - choice depend on the hardware implementation of $\mathbb{F}_p$
  - additions not always negligible

**Multiplication in $\mathbb{F}_3$**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\times$</th>
<th>$\plus$</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schoolbook</td>
<td>25</td>
<td>24</td>
<td>0.96</td>
</tr>
<tr>
<td>One-level Karatsuba (Montgomery’s trick)</td>
<td>21</td>
<td>29</td>
<td>1.38</td>
</tr>
<tr>
<td>Recursive Karatsuba</td>
<td>15</td>
<td>39</td>
<td>2.60</td>
</tr>
<tr>
<td>Recursive Karatsuba (Montgomery’s trick)</td>
<td>14</td>
<td>43</td>
<td>3.07</td>
</tr>
<tr>
<td>Montgomery’s Karatsuba-like</td>
<td>13</td>
<td>54</td>
<td>4.153</td>
</tr>
<tr>
<td>CRT-based</td>
<td>12</td>
<td>53</td>
<td>4.42</td>
</tr>
</tbody>
</table>
Choosing multiplication algorithm

- Evaluate the cost of the different algorithms
  - choice depend on the hardware implementation of $\mathbb{F}_{p^m}$
  - additions not always negligible

### Multiplication in $\mathbb{F}_{3^m5}$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\times$</th>
<th>$+$</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schoolbook</td>
<td>25</td>
<td>24</td>
<td>0.96</td>
</tr>
<tr>
<td>One-level Karatsuba (Montgomery’s trick)</td>
<td>21</td>
<td>29</td>
<td>1.38</td>
</tr>
<tr>
<td>Recursive Karatsuba</td>
<td>15</td>
<td>39</td>
<td>2.60</td>
</tr>
<tr>
<td>Recursive Karatsuba (Montgomery’s trick)</td>
<td>14</td>
<td>43</td>
<td>3.07</td>
</tr>
<tr>
<td>Montgomery’s Karatsuba-like</td>
<td>13</td>
<td>54</td>
<td>4.153</td>
</tr>
<tr>
<td>CRT-based</td>
<td>12</td>
<td>53</td>
<td>4.42</td>
</tr>
</tbody>
</table>

### Multiplication in $\mathbb{F}_{2^m7}$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\times$</th>
<th>$+$</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schoolbook</td>
<td>49</td>
<td>48</td>
<td>0.98</td>
</tr>
<tr>
<td>One-level Karatsuba (Montgomery’s trick)</td>
<td>40</td>
<td>52</td>
<td>1.30</td>
</tr>
<tr>
<td>Recursive Karatsuba</td>
<td>25</td>
<td>51</td>
<td>2.04</td>
</tr>
<tr>
<td>Recursive Karatsuba (Montgomery’s trick)</td>
<td>23</td>
<td>76</td>
<td>3.30</td>
</tr>
<tr>
<td>Montgomery’s Karatsuba-like</td>
<td>22</td>
<td>84</td>
<td>3.818</td>
</tr>
<tr>
<td>CRT-based</td>
<td>22</td>
<td>88</td>
<td>4.05</td>
</tr>
</tbody>
</table>
Outline of the talk

▶ Small characteristic finite fields

▶ Multiplication algorithms and hardware implementation

▶ A finite field coprocessor

▶ Finite fields of composite extension degree

▶ Conclusion
Accelarators for curve-based cryptography rely on finite field arithmetic.
Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic

- Many criterion for optimization
  - area: low cost devices
Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic.

- Many criterion for optimization:
  - area: low cost devices
  - speed: if security should not introduce latency
Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic.

- Many criterion for optimization:
  - area: low cost devices
  - speed: if security should not introduce latency
  - area-speed tradeoff: high-throughput application
Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic

- Many criterion for optimization
  - area: low cost devices
  - speed: if security should not introduce latency
  - area-speed tradeoff: high-throughput application

- Multiplication is the critical operation
  - many implementation strategies
Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic.

- Many criterion for optimization:
  - **area**: low cost devices
  - **speed**: if security should not introduce latency
  - **area-speed tradeoff**: high-throughput application

- **Multiplication** is the critical operation:
  - many implementation strategies

- Need for algorithms/hardware codesign
Thank you for your attention!

Questions?