Fast hardware accelerator for the Tate pairing based on a fully parallel Karatsuba multiplier

Nicolas Estibals
CACAO project-team, LORIA
INRIA Nancy – Grand-Est
Nicolas.Estibals@loria.fr

Joint work with:

Jean-Luc Beuchat
LCIS, University of Tsukuba, Japan

Jérémie Detrey
CACAO, LORIA, Nancy, France

Eiji Okamoto
LCIS, University of Tsukuba, Japan

Francisco Rodríguez-Henríquez
CINVESTAV-IPN, Mexico City, Mexico
Outline of the talk

- Context
- Reduced Tate pairing
- Non-reduced Tate pairing
- Fully parallel Karatsuba multiplier
- Final Exponentiation
- Results & Conclusion
- Appendix
Pairing-based cryptography

▸ Origin of pairings in cryptography
  • attack against some elliptic curves
    ✫ Menezes–Okamoto–Vanstone, 1993
    ✫ Frey–Rück, 1994
Pairing-based cryptography

- Origin of pairings in cryptography
  - **attack** against some elliptic curves
    - Menezes–Okamoto–Vanstone, 1993
    - Frey–Rück, 1994

- Constructive properties
  - **One-round three party key exchange**
    - Joux, 2000
  - **short digital signature**
    - Boneh–Lynn–Shacham, 2001
    - Zang–Safavi–Naini–Susilo, 2004
  - **identity-based encryption**
    - Boneh–Franklin, 2001
    - Sakai–Kasahara, 2001
  - ...
Pairing-based cryptography

▶ Origin of pairings in cryptography
  • attack against some elliptic curves
    ◦ Menezes–Okamoto–Vanstone, 1993
    ◦ Frey–Rück, 1994

▶ Constructive properties
  • One-round three party key exchange
    ◦ Joux, 2000
  • short digital signature
    ◦ Boneh–Lynn–Shacham, 2001
    ◦ Zang–Safavi–Naini–Susilo, 2004
  • identity-based encryption
    ◦ Boneh–Franklin, 2001
    ◦ Sakai–Kasahara, 2001
  • ...

▶ Standardization in progress
  • ISO/IEC 14888-3
  • IEEE P1363.3
Which pairing?

▶ Reduced Tate pairing
  - common choice for cryptographic applications
Which pairing?

- Reduced Tate pairing
  - common choice for cryptographic applications

- Pairing on supersingular curves
  - easier arithmetic on the curve
  - lower security

- Small characteristic
  - higher embedding degree
  - higher security
Which pairing?

- **Reduced Tate** pairing
  - common choice for cryptographic applications

- **Pairing on supersingular curves**
  - easier arithmetic on the curve
    - lower security

- **Small characteristic**
  - higher embedding degree
  - higher **security**

- Need for **dedicated hardware coprocessor**
  - area optimized (RFID, embedded systems, ...)
  - speed optimized (bank servers, ...)

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Which pairing?

- **Reduced Tate** pairing
  - common choice for cryptographic applications

- **Pairing on** supersingular curves
  - easier arithmetic on the curve
  - lower security

- **Small characteristic**
  - higher embedding degree
  - higher security

- Need for dedicated hardware coprocessor
  - area optimized (RFID, embedded systems, ...)
  - speed optimized (bank servers, ...)

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Outline of the talk

▶ Context

▶ Reduced Tate pairing

▶ Non-reduced Tate pairing

▶ Fully parallel Karatsuba multiplier

▶ Final Exponentiation

▶ Results & Conclusion

▶ Appendix
Pairing is a bilinear map

- $G_1 = \langle P \rangle$: additively-written cyclic group of prime order $\#G_1 = \ell$
- $G_2$: multiplicatively-written cyclic group of order $\#G_2 = \#G_1 = \ell$
Pairing is a bilinear map

- $G_1 = \langle P \rangle$: additively-written cyclic group of prime order $\#G_1 = \ell$
- $G_2$: multiplicatively-written cyclic group of order $\#G_2 = \#G_1 = \ell$
- $\hat{e} : G_1 \times G_1 \rightarrow G_2$ is a bilinear pairing iff:
  - non-degeneracy: $\hat{e}(P, P) \neq 1_{G_2}$
  - bilinearity:
    - $\hat{e}(Q_1 + Q_2, R) = \hat{e}(Q_1, R) \cdot \hat{e}(Q_2, R)$
    - $\hat{e}(Q, R_1 + R_2) = \hat{e}(Q, R_1) \cdot \hat{e}(Q, R_2)$
  - computability: $\hat{e}$ can be efficiently computed
Pairing is a bilinear map

- $\mathbb{G}_1 = \langle P \rangle$: additively-written cyclic group of prime order $\# \mathbb{G}_1 = \ell$
- $\mathbb{G}_2$: multiplicatively-written cyclic group of order $\# \mathbb{G}_2 = \# \mathbb{G}_1 = \ell$
- $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a bilinear pairing iff:
  - non-degeneracy: $\hat{e}(P, P) \neq 1_{\mathbb{G}_2}$
  - bilinearity:
    $\hat{e}(Q_1 + Q_2, R) = \hat{e}(Q_1, R) \cdot \hat{e}(Q_2, R)$
    $\hat{e}(Q, R_1 + R_2) = \hat{e}(Q, R_1) \cdot \hat{e}(Q, R_2)$
  - computability: $\hat{e}$ can be efficiently computed
- Important property for cryptographic applications:

  $\hat{e}(k_1 P, k_2 P) = \hat{e}(k_2 P, k_1 P) = \hat{e}(P, P)^{k_1 k_2}$

  Combining secrets without having to reveal them!
Pairing over elliptic curve

Reduced Tate pairing
Pairing over elliptic curve

Input: two points $P$ and $Q$ in $G_1 = E(\mathbb{F}_q)[\ell]$, where:

- $q = p, 2^m$ or $3^m$
- $E$ is an elliptic curve over $\mathbb{F}_q$
- $\ell$ is a large prime factor of $\#E(\mathbb{F}_q)$
- $G_1 = E(\mathbb{F}_q)[\ell] = \{ P \in E(\mathbb{F}_q) | \ell P = O \}$

Reduced Tate pairing

$E(\mathbb{F}_q)[\ell]$
Pairing over elliptic curve

Input: two points $P$ and $Q$ in $G_1 = E(\mathbb{F}_q)[\ell]$, where:
- $q = p$, $2^m$ or $3^m$
- $E$ is an elliptic curve over $\mathbb{F}_q$
- $\ell$ is a large prime factor of $\#E(\mathbb{F}_q)$
- $G_1 = E(\mathbb{F}_q)[\ell] = \{P \in E(\mathbb{F}_q) | \ell P = O\}$

Output: an $\ell$-th root of unity
- $G_2 = \mu_\ell = \left\{ U \in \mathbb{F}_q^\times | U^\ell = 1 \right\}$
- $k$ is the embedding degree: the smallest integer such that $\mu_\ell \subseteq \mathbb{F}_q^\times$
Security considerations

\[ \hat{e} : E(\mathbb{F}_q)[\ell] \times E(\mathbb{F}_q)[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_{q^k} \]

- Security should be enough in \( G_1 \) and in \( G_2 \)
- **Supersingular** curves: \( k \) is bounded
- Some typical cases:

<table>
<thead>
<tr>
<th>Base field ((\mathbb{F}_q))</th>
<th>( \mathbb{F}_{2^m} )</th>
<th>( \mathbb{F}_{3^m} )</th>
<th>( \mathbb{F}_p )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Embedding degree ((k))</strong></td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td><strong>Lower security ( \sim 2^{64} )</strong></td>
<td>( m = 239 )</td>
<td>( m = 97 )</td>
<td>(</td>
</tr>
<tr>
<td><strong>Medium security ( \sim 2^{80} )</strong></td>
<td>( m = 373 )</td>
<td>( m = 163 )</td>
<td>(</td>
</tr>
<tr>
<td><strong>Higher security ( \sim 2^{128} )</strong></td>
<td>( m = 1103 )</td>
<td>( m = 503 )</td>
<td>(</td>
</tr>
</tbody>
</table>
Security considerations

\[ \hat{e} : E(\mathbb{F}_q)[\ell] \times E(\mathbb{F}_q)[\ell] \rightarrow \mu_{\ell} \subseteq \mathbb{F}_q^* \]

- Security should be enough in \( G_1 \) and in \( G_2 \)
- Supersingular curves: \( k \) is bounded
- Some typical cases:

<table>
<thead>
<tr>
<th>Base field ((\mathbb{F}_q))</th>
<th>( \mathbb{F}_{2^m} )</th>
<th>( \mathbb{F}_{3^m} )</th>
<th>( \mathbb{F}_p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding degree ((k))</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Lower security ((\sim 2^{64}))</td>
<td>( m = 239 )</td>
<td>( m = 97 )</td>
<td>(</td>
</tr>
<tr>
<td>Medium security ((\sim 2^{80}))</td>
<td>( m = 373 )</td>
<td>( m = 163 )</td>
<td>(</td>
</tr>
<tr>
<td>Higher security ((\sim 2^{128}))</td>
<td>( m = 1103 )</td>
<td>( m = 503 )</td>
<td>(</td>
</tr>
</tbody>
</table>

- \( G_2 = \mu_{\ell} \subseteq \mathbb{F}_q^* \) is the bottleneck
- Low characteristic \((p = 2 \text{ or } 3)\) because of higher embedding degree
Reduced Tate pairing

\[ E(F_{p^m})[\ell] \]

\[ \mu_\ell \subseteq F_{p^{km}}^\times \]

Two very different steps:
• non-reduced pairing: Miller’s iterative algorithm
• final exponentiation: irregular computation

Idea: use two separate coprocessors
• pipeline the two computations
• balance the latencies
Reduced Tate pairing

\[ E(\mathbb{F}_{p^m})[\ell] \]

\[ \mu \subseteq \mathbb{F}_{p_{km}}^{\times} \]

Two very different steps:
Reduced Tate pairing

Two very different steps:

- non-reduced pairing: Miller’s iterative algorithm
Reduced Tate pairing

Two very different steps:

- non-reduced pairing: Miller’s iterative algorithm
- final exponention: irregular computation
Reduced Tate pairing

Two very different steps:
- non-reduced pairing: Miller’s iterative algorithm
- final exponentiation: irregular computation

Idea: use two separate coprocessors
Reduced Tate pairing

Two very different steps:
- non-reduced pairing: Miller’s iterative algorithm
- final exponentiation: irregular computation

Idea: use two separate coprocessors
- pipeline the two computations
- balance the latencies
Outline of the talk

▶ Context

▶ Reduced Tate pairing

▶ Non-reduced Tate pairing

▶ Fully parallel Karatsuba multiplier

▶ Final Exponentiation

▶ Results & Conclusion

▶ Appendix
A closer look on Miller’s loop (char. 3)

- $\eta_T$ pairing: shorter loop

```latex
\begin{align*}
\text{for } i & \leftarrow 0 \text{ to } (m - 1)/2 \text{ do } \\
\text{end for}
\end{align*}
```
A closer look on Miller’s loop (char. 3)

- $\eta_T$ pairing: shorter loop

- Based on Miller’s algorithm:

```plaintext
for i ← 0 to (m − 1)/2 do
    \(x_P \leftarrow \sqrt[3]{x_P}\) ; \(y_P \leftarrow \sqrt[3]{y_P}\)
    \(x_Q \leftarrow x_Q^3\) ; \(y_Q \leftarrow y_Q^3\)
    \(t \leftarrow x_P + x_Q\) \(u \leftarrow y_P y_Q\)
    \(S \leftarrow −t^2 ± uσ − tρ − ρ^2\)
    \(R \leftarrow R \cdot S\)
end for
```
A closer look on Miller’s loop (char. 3)

- $\eta_T$ pairing: shorter loop

- Based on Miller’s algorithm:
  1. update of point coordinates

```plaintext
for i ← 0 to (m − 1)/2 do
    $x_P ← \sqrt[3]{x_P}$ ; $y_P ← \sqrt[3]{y_P}$
    $x_Q ← x_Q^3$ ; $y_Q ← y_Q^3$
    $t ← x_P + x_Q$  $u ← y_P y_Q$
    $S ← -t^2 ± u\sigma - t\rho - \rho^2$
    $R ← R \cdot S$
end for
```
A closer look on Miller’s loop (char. 3)

- $\eta_T$ pairing: shorter loop

- Based on Miller’s algorithm:
  1. update of point coordinates
  2. computation of line equation

```latex
\begin{align*}
\text{for } i &\leftarrow 0 \text{ to } (m - 1)/2 \text{ do} \\
\quad &\text{for } i \leftarrow 0 \text{ to } (m - 1)/2 \text{ do} \\
&\quad t \leftarrow x_P + x_Q\text{; } u \leftarrow y_P y_Q \text{; } S \leftarrow -t^2 \pm u \sigma - t \rho - \rho^2 \text{; } R \leftarrow R \cdot S.
\end{align*}
```
A closer look on Miller’s loop (char. 3)

- $\eta_T$ pairing: shorter loop

- Based on Miller’s algorithm:
  1. update of point coordinates
  2. computation of line equation
  3. accumulation of the new factor

```plaintext
for $i \leftarrow 0$ to $(m - 1)/2$ do
    \( x_P \leftarrow \sqrt[3]{x_P} \); \( y_P \leftarrow \sqrt[3]{y_P} \)
    \( x_Q \leftarrow x_Q^3 \); \( y_Q \leftarrow y_Q^3 \)
    \( t \leftarrow x_P + x_Q \); \( u \leftarrow y_P y_Q \)
    \( S \leftarrow -t^2 \pm u \sigma - t \rho - \rho^2 \)
    \( R \leftarrow R \cdot S \)
end for
```

Objective: keep the multiplier pipeline busy
- 7-stages pipeline
- one product per cycle
- 17 cycles per iteration

Sparse multiplication over $\mathbb{F}_{3^m}$ (Gorla et al., SAC 2007)
- $12 \times , 59 +$ over $\mathbb{F}_{3^m}$ (Beuchat et al., ARITH 18)
A closer look on Miller’s loop (char. 3)

- $\eta_T$ pairing: shorter loop

- Based on Miller’s algorithm:
  1. update of point coordinates
  2. computation of line equation
  3. accumulation of the new factor

- Multiplication is critical

- Fully parallel, pipelined multiplier over $\mathbb{F}_{3^m}$

for $i \leftarrow 0$ to $(m - 1)/2$ do

1. $x_P \leftarrow \sqrt[3]{x_P}$ ; $y_P \leftarrow \sqrt[3]{y_P}$
   $x_Q \leftarrow x_Q^3$ ; $y_Q \leftarrow y_Q^3$

2. $t \leftarrow x_P + x_Q$ ; $u \leftarrow y_P y_Q$
   $S \leftarrow -t^2 \pm u \sigma - t \rho - \rho^2$

3. $R \leftarrow R \cdot S$

end for

Sparse multiplication over $\mathbb{F}_{3^m}$ (Gorla et al., SAC 2007)

Objective: keep the multiplier pipeline busy

• 7-stages pipeline
• one product per cycle
• 17 cycles per iteration
A closer look on Miller’s loop (char. 3)

- \( \eta_T \) pairing: shorter loop

- Based on Miller’s algorithm:
  1. update of point coordinates
  2. computation of line equation
  3. accumulation of the new factor

- Multiplication is critical

- Fully parallel, pipelined multiplier over \( \mathbb{F}_{3^m} \)

- Sparse multiplication over \( \mathbb{F}_{3^6m} \)

```plaintext
for i ← 0 to (m − 1)/2 do
  \begin{align*}
  t &← x_P + x_Q; \quad u ← y_P y_Q \\
  S &← −t^2 ± uσ − tρ − ρ^2
  \end{align*}
  \begin{align*}
  &2 \sqrt{3}, 2 + \\
  &1 \times (\mathbb{F}_{3^6m})
\end{align*}
end for
```
A closer look on Miller’s loop (char. 3)

- **ηₜ** pairing: shorter loop

- Based on Miller’s algorithm:
  1. update of point coordinates
  2. computation of line equation
  3. accumulation of the new factor

- Multiplication is critical

- Fully parallel, pipelined
  multiplier over \( F_{3^m} \)

- Sparse multiplication over \( F_{36^m} \)
  - 12 × and 59 + over \( F_{3^m} \) (Gorla et al., SAC 2007)

```plaintext
for i ← 0 to (m − 1)/2 do
    t ← \( x_P + x_Q \); u ← \( y_P y_Q \)
    S ← \( -t^2 ± u\sigma - t\rho - \rho^2 \)
    \( R ← R \cdot S \)
end for
```
A closer look on Miller’s loop (char. 3)

- $\eta_T$ pairing: shorter loop

- Based on Miller’s algorithm:
  1. update of point coordinates
  2. computation of line equation
  3. accumulation of the new factor

- Multiplication is critical

- Fully parallel, pipelined multiplier over $\mathbb{F}_{3^m}$

- Sparse multiplication over $\mathbb{F}_{3^{6m}}$
  - $12 \times$ and $59 +$ over $\mathbb{F}_{3^m}$ (Gorla et al., SAC 2007)
  - $15 \times$ and $29 +$ over $\mathbb{F}_{3^m}$ (Beuchat et al., ARITH 18)

```
for i ← 0 to (m − 1)/2 do
    $t ← x_P + x_Q$ ; $u ← y_P y_Q$
    $S ← -t^2 ± u\sigma − t\rho − \rho^2$
    $R ← R · S$
end for
```
A closer look on Miller’s loop (char. 3)

- \( \eta_T \) pairing: shorter loop
- Based on Miller’s algorithm:
  1. update of point coordinates
  2. computation of line equation
  3. accumulation of the new factor
- Multiplication is critical
- Fully parallel, pipelined multiplier over \( \mathbb{F}_{3^m} \)
- Sparse multiplication over \( \mathbb{F}_{3^{6m}} \)
  - 12 \( \times \) and 59 + over \( \mathbb{F}_{3^m} \) (Gorla et al., SAC 2007)
  - 15 \( \times \) and 29 + over \( \mathbb{F}_{3^m} \) (Beuchat et al., ARITH 18)
- Objective: keep the multiplier pipeline busy
  - 7-stages pipeline
  - one product per cycle
  - 17 cycles per iteration

\[
\begin{align*}
\text{for } i & \leftarrow 0 \text{ to } (m - 1)/2 \text{ do} \\
1 & \quad x_P & \leftarrow & \sqrt[3]{x_P} \\
& & & y_P \leftarrow \sqrt[3]{y_P} \\
& & & x_Q \leftarrow x_Q^3 \\
& & & y_Q \leftarrow y_Q^3 \\
2 & \quad t & \leftarrow & x_P + x_Q \\
& & & u \leftarrow y_P y_Q \\
& & & S \leftarrow -t^2 \pm u \sigma - t \rho - \rho^2 \\
3 & \quad R & \leftarrow & R \cdot S
\end{align*}
\]

Sparse multiplication over \( \mathbb{F}_{3^{6m}} \)

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Coprocessor for the non-reduced pairing (char. 3)

Parallel multiplier
- 7 pipeline stages
- 1 product / cycle

Operands preparation
Post-processing

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Coprocessor for the non-reduced pairing (char. 3)

- Operands preparation
- Parallel multiplier
  7 pipeline stages
  1 product / cycle
- Post-processing
Coprocessor for the non-reduced pairing (char. 3)

Operands preparation

Parallel multiplier
7 pipeline stages
1 product / cycle

Post-processing
Coprocessor for the non-reduced pairing (char. 3)
Coprocessor for the non-reduced pairing (char. 3)

- Register file
- Parallel multiplier
  - 7 pipeline stages
  - 1 product / cycle
- Operands preparation
- Coordinates update
- Post-processing
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)
Detailed architecture of the coprocessor (char. 3)

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Coprocessor for the non-reduced pairing (char. 2)

- Similar algorithm
  - \( \frac{m+1}{2} \) iterations

Diagram:
- Register file
- Parallel multiplier
  - 5 pipeline stages
  - 1 product / cycle
- Post-processing
- Coefficients preparation
- Post-processing
- X
- Coordinates update
Coprocessor for the non-reduced pairing (char. 2)

- Similar algorithm
  - $\frac{m+1}{2}$ iterations

- But some differences:
  - only 6 products over $\mathbb{F}_{2^m}$ per iteration
  - different scheduling
  - 5-stages multiplier pipeline

Diagram:

- Coordinates update
- Operands preparation
- Parallel multiplier
  - 5 pipeline stages
  - 1 product / cycle
- Post-processing
- Register file
- Parallel multiplier
  - 5 pipeline stages
  - 1 product / cycle
- Post-processing

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Coproessor for the non-reduced pairing (char. 2)

Similar algorithm
- \( \frac{m+1}{2} \) iterations

But some differences:
- only 6 products over \( \mathbb{F}_{2^m} \) per iteration
- different scheduling
- 5-stages multiplier pipeline

No need for a register file
Coprocessor for the non-reduced pairing (char. 2)

- Similar algorithm
  - $\frac{m+1}{2}$ iterations

- But some differences:
  - only 6 products over $\mathbb{F}_{2^m}$ per iteration
  - different scheduling
  - 5-stages multiplier pipeline

- No need for a register file
  - all data in shift register
  - more complex architecture
Detailed architecture of the coprocessor (char. 2)
Detailed architecture of the coprocessor (char. 2)
Detailed architecture of the coprocessor (char. 2)
Detailed architecture of the coprocessor (char. 2)
Outline of the talk

- Context
- Reduced Tate pairing
- Non-reduced Tate pairing
- Fully parallel Karatsuba multiplier
- Final Exponentiation
- Results & Conclusion
- Appendix
Finite field representation and Karatsuba’s formula

- Polynomial basis:
  - $\mathbb{F}_p^m \cong \mathbb{F}_p[x]/(f(x))$
  - $f(x)$ irreducible polynomial of degree $m$ in $\mathbb{F}_p[x]$
  - $\mathbb{F}_p^m$ represented by $\mathbb{F}_p[x]^{\leq (m-1)}$
Finite field representation and Karatsuba’s formula

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

- Karatsuba algorithm for polynomials
Finite field representation and Karatsuba’s formula

- Polynomial basis:
  - \( \mathbb{F}_p^m \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \) irreducible polynomial of degree \( m \) in \( \mathbb{F}_p[x] \)
  - \( \mathbb{F}_p^m \) represented by \( \mathbb{F}_p[x]^{\leq (m-1)} \)

- Karatsuba algorithm for polynomials

\[
A_H B_H X^{2n} + (A_H B_L + A_L B_H)X^n + A_L B_L
\]
Finite field representation and Karatsuba’s formula

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

- Karatsuba algorithm for polynomials

$$a b' + a' b = (a + a')(b + b') - a b - a' b'$$

$A_H B_H X^{2n} + (A_H B_L + A_L B_H) X^n + A_L B_L$

$A_H B_H X^{2n} + ((A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L) X^n + A_L B_L$
Finite field representation and Karatsuba’s formula

- **Polynomial basis:**
  - \( \mathbb{F}_p^m \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \) irreducible polynomial of degree \( m \) in \( \mathbb{F}_p[x] \)
  - \( \mathbb{F}_p^m \) represented by \( \mathbb{F}_p[x]^{\leq (m-1)} \)

- **Karatsuba algorithm for polynomials**
Finite field representation and Karatsuba’s formula

- **Polynomial basis:**
  - $\mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x))$
  - $f(x)$ irreducible polynomial of degree $m$ in $\mathbb{F}_p[x]$
  - $\mathbb{F}_{p^m}$ represented by $\mathbb{F}_p[x]^{\leq (m-1)}$

- **Karatsuba algorithm for polynomials**
Finite field representation and Karatsuba’s formula

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

- Karatsuba algorithm for polynomials
Odd–even split for Karatsuba multiplication

\[ A \cdot B \]
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

\[ (A_O B_O X^2 + A_E B_E) + X(A_O B_E + A_E B_O) \]

[Diagram]

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

[Diagram]

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

$A_O \cdot B_O + A_E \cdot B_E$

$A \cdot B$

$(A_O \cdot B_O + A_E \cdot B_E) + (A_O \cdot B_E + A_E \cdot B_O)$
Odd–even split for Karatsuba multiplication

\[
A \cdot B
\]
Some other subquadratic multiplication algorithms

- Karatsuba-like algorithms: detailed algorithm
  - original formula
  - 3-way Karatsuba (7 instead of 9 subproducts)
  - odd–even split
  - 3-way odd–even split-like
Some other subquadratic multiplication algorithms

- **Karatsuba-like algorithms:**
  - original formula
  - 3-way Karatsuba (7 instead of 9 subproducts)
  - odd–even split
  - 3-way odd–even split-like

- **Toom-Cook algorithms:**
  - evaluation–interpolation scheme
  - split operands in 3 or more parts
  - symmetric or asymmetric splitting
  - odd–even trick (work in progress)

- **Montgomery’s formulae**

- ...
Some other subquadratic multiplication algorithms

- Karatsuba-like algorithms: [detailed algorithm]
  - original formula
  - 3-way Karatsuba (7 instead of 9 subproducts)
  - odd–even split
  - 3-way odd–even split-like

- Toom-Cook algorithms:
  - evaluation–interpolation scheme
  - split operands in 3 or more parts
  - symmetric or asymmetric splitting
  - odd–even trick (work in progress)

- Montgomery’s formulae

- ...

- Select best method for each stage of recursion
Multiplier architecture

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

- **Fully parallel** evaluation of the subproducts
Multiplier architecture

- **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

---

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Multiplier architecture

▶ 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
Multiplier architecture

- 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 $f$
  - small operator if $f$ has low Hamming weight
Choice of the recursion and FPGA implementation

▶ Leading zeros problem
  - add them when the inputs are not perfectly splittable
  - increase the size of subproducts
Choice of the recursion and FPGA implementation

► Leading zeros problem

- add them when the inputs are not perfectly splittable
- increase the size of subproducts
- correctly choose the recursion
- use different kinds of multiplier for the different subproducts
Choice of the recursion and FPGA implementation

- Leading zeros problem
  - add them when the inputs are not perfectly splittable
  - increase the size of subproducts
  - correctly choose the recursion
  - use different kinds of multiplier for the different subproducts

- Form of addition trees
  - depends on characteristic
  - depends on FPGA technology
  - maximize LUTs utilization
Choice of the recursion and FPGA implementation

- **Leading zeros problem**
  - add them when the inputs are not perfectly splittable
  - increase the size of subproducts
  - correctly choose the recursion
  - use different kinds of multiplier for the different subproducts

- **Form of addition trees**
  - depends on characteristic
  - depends on FPGA technology
  - maximize LUTs utilization

- **Why the odd–even trick does not always work**
  - depends also on characteristic and FPGA
  - have to estimate the area in number of LUT not in number of additions
An example of multiplier over $\mathbb{F}_{3^{239}}$

- Polynomial basis

$$\mathbb{F}_{3^{239}} \cong \mathbb{F}_3[X]/(X^{239} - X^5 + 1)$$
An example of multiplier over $\mathbb{F}_{3^{239}}$

- Polynomial basis

$$\mathbb{F}_{3^{239}} \cong \mathbb{F}_3[X]/(X^{239} - X^5 + 1)$$

- 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>
An example of multiplier over $\mathbb{F}_{3^{239}}$

- Polynomial basis

\[ \mathbb{F}_{3^{239}} \cong \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \]

- 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
  
  - 49984 slices
  - 200 MHz

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Outline of the talk

- Context
- Reduced Tate pairing
- Non-reduced Tate pairing
- Fully parallel Karatsuba multiplier
- Final Exponentiation
- Results & Conclusion
- Appendix

$E(F_{p^m})[\ell] \subseteq F \times p^{km}$

(iterative algorithm)

(irregular computation)

Non-reduced pairing

Final exponentiation

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Final exponentiation

Design rationale:

- as small as possible
- at least as fast as the computation of the non-reduced pairing
Coprocessor for the final exponentiation (char. 3)

- Highly sequential computation
- Very heterogeneous
Coproccessor for the final exponentiation (char. 3)

- Highly sequential computation
- Very heterogeneous

\[
\begin{align*}
\text{general-purpose} \
\Rightarrow \\
\text{finite-field arithmetic} \\
\text{processor}
\end{align*}
\]
Coprocesor for the final exponentiation (char. 3)

- Highly sequential computation
- Very heterogeneous

⇒ general-purpose finite-field arithmetic processor

Register file

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Coprocessor for the final exponentiation (char. 3)

- Highly sequential computation
- Very heterogeneous

\[ \text{Register file} \]

Parallel–serial multiplier

\[ D \text{ coeffs / cycle} \]

\[ \lceil \frac{m}{D} \rceil \text{ cycles / product} \]

\[ \Rightarrow \text{general-purpose finite-field arithmetic processor} \]
Coprocessor for the final exponentiation (char. 3)

- Highly sequential computation
- Very heterogeneous

⇒ general-purpose finite-field arithmetic processor

- Register file
- Unified operator addition / subtraction Frobenius $(\cdot)^3$
- Parallel–serial multiplier
  
  $D$ coeffs / cycle
  
  $\lceil m/D \rceil$ cycles / product

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Coprocessor for the final exponentiation (char. 3)

- Highly **sequential** computation
- Very **heterogeneous**

⇒ general-purpose finite-field arithmetic processor

- **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

▶ Context

▶ Reduced Tate pairing

▶ Non-reduced Tate pairing

▶ Fully parallel Karatsuba multiplier

▶ Final Exponentiation

▶ Results & Conclusion

▶ Appendix
Experimental setup

▶ Full Tate pairing computation:
  • non-reduced pairing and
  • final exponentiation

▶ Prototyped on Xilinx Virtex-II Pro and Virtex-4 LX FPGAs

▶ Post-place-and-route timing and area estimations
Calculation time

Calculation time [\(\mu s\)]

Security [bits]

<table>
<thead>
<tr>
<th>Security [bits]</th>
<th>Calculation time [(\mu s)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>20.9 (\mu s / F_{397})</td>
</tr>
<tr>
<td>70</td>
<td>100.8 (\mu s / F_{2457})</td>
</tr>
<tr>
<td>80</td>
<td>675.5 (\mu s / F_{2557})</td>
</tr>
</tbody>
</table>

Virtex-I

Virtex-IIPro

Virtex-4LX
Calculation time

![Graph showing calculation time versus security bits for different hardware accelerators including Virtex-II Pro and Virtex-4 LX.](image)

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Calculation time

Calculation time [µs]

Security [bits]

Virtex-II Pro
6.2 µs / F_3^{97}
12.8 µs / F_3^{193}
20.9 µs / F_3^{97}

Virtex-4LX
100.8 µs / F_2^{457}
675.5 µs / F_2^{557}

Security [bits]
Calculation time

Security [bits] vs. Calculation time [$\mu$s]

- **Virtex-I Pro**
  - $6.2 \mu s / F_{397}$
  - $12.8 \mu s / F_{3193}$
  - $20.9 \mu s / F_{397}$
  - $100.8 \mu s / F_{2457}$
  - $675.5 \mu s / F_{2557}$

- **Virtex-II Pro**
  - $6.2 \mu s / F_{397}$
  - $12.8 \mu s / F_{3193}$
  - $20.9 \mu s / F_{397}$
  - $100.8 \mu s / F_{2457}$
  - $675.5 \mu s / F_{2557}$
Calculation time

Calculation time $[\mu s]$

Security [bits] vs. Calculation time $[\mu s]$

- **Virtex-I**
  - $675.5 \mu s / F_{2557}$
  - $100.8 \mu s / F_{2457}$
  - $20.9 \mu s / F_{397}$

- **Virtex-II Pro**
  - $6.2 \mu s / F_{397}$
  - $100.8 \mu s / F_{2457}$
  - $20.9 \mu s / F_{397}$

- **Virtex-4 LX**
  - $12.8 \mu s / F_{3193}$
  - $16.9 \mu s / F_{3313}$

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Area

Area [slices]

Security [bits]

Virtex-I

Virtex-II Pro

Virtex-4 LX

xc2vp30

xc2vp100

xc4vlx200

18,360 sl. / $\mathbb{F}_{3^{97}}$

46,360 sl. / $\mathbb{F}_{3^{193}}$

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Area–Time product

Area–time product [slices \cdot s]

Security [bits]

Area–time product [slices \cdot s]

Security [bits]

Virtex-I

I Pro

AES-128?

Char. 2 (Virtex-4
LX)

3.5 µs – 16,203 sl. /F 2^{239}

7.5 µs – 44,223 sl. /F 2^{457}

18.8 µs – 78,874 sl. /F 2^{691}

AES-128?
Area–Time product

Area–time product [slices \cdot s]

Security [bits]

Area–time product [slices \cdot s]

Virtex-I

Virtex-4

Char. 2 (Virtex-4
LX)

AES-128?

3.5 µs – 16,203 sl. /F
7.5 µs – 44,223 sl. /F
18.8 µs – 78,874 sl. /F
Area–Time product

Area–time product [slices \cdot s]

Security [bits]

Area–time product [slices \cdot s]

Virtex-I

Virtex-II Pro

Virtex-4 LX

Char. 2 (Virtex-4 LX)

AES-128?

3.5 µs – 16,203 sl. /F

7.5 µs – 44,223 sl. /F

18.8 µs – 78,874 sl. /F

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Area–Time product

Area–time product [slices \cdot s]

Security [bits]  

Area–time product [slices \cdot s]  

Virtex-I
Virtex-II Pro
Char. 2 (Virtex-4 LX)
Virtex-4 LX

18.8 \mu s – 78,874 sl. / F_{2^{691}}
7.5 \mu s – 44,223 sl. / F_{2^{2457}}
3.5 \mu s – 16,203 sl. / F_{2^{239}}

Nicolas Estibals — Fast hardware accelerator for the Tate pairing
Area–Time product

Area–time product [slices · s]

Security [bits]

Area–time product [slices · s]

Virtex-II Pro

Char. 2 (Virtex-4 LX)

Virtex-4 LX

AES-128?

18.8 µs – 78,874 sl. / $F_{2691}$

7.5 µs – 44,223 sl. / $F_{2457}$

3.5 µs – 16,203 sl. / $F_{239}$


Conclusion

- A new architecture for pairing computation
  - two specialized coprocessors
  - bet on parallelizing multiplier
  - based on Karatsuba multiplication scheme
  - importance of architecture–algorithm co-design
  - careful bubble-free scheduling of Miller’s loop

High-performance accelerator
- the fastest coprocessor (17 µs for 10^9 bits of security)
- the best area–time trade-off
- scales to higher security levels
Conclusion

▶ A new architecture for pairing computation
  - two specialized coprocessors
  - bet on parallelizing multiplier
  - based on Karatsuba multiplication scheme
  - importance of architecture–algorithm co-design
  - careful bubble-free scheduling of Miller’s loop

▶ High-performance accelerator
  - the fastest coprocessor (17 µs for 109 bits of security)
  - the best area–time trade-off
  - scales to higher security levels
Future work

▶ Fully parallel multipliers
  • tune finely Karatsuba algorithm and multiplier architecture
  • try other algorithms: Toom–Cook, Montgomery’s formulae
  • try less parallel multipliers: slower but smaller
Future work

- **Fully parallel multipliers**
  - tune finely Karatsuba algorithm and multiplier architecture
  - try other algorithms: Toom–Cook, Montgomery’s formulae
  - try less parallel multipliers: slower but smaller

- **Final-exponentiation coprocessor**
  - full-featured finite-field processor
  - compute the full pairing with it (promising first experimental results)
Future work

▶ Fully parallel multipliers
  • tune finely Karatsuba algorithm and multiplier architecture
  • try other algorithms: Toom–Cook, Montgomery’s formulae
  • try less parallel multipliers: slower but smaller

▶ Final-exponentiation coprocessor
  • full-featured finite-field processor
  • compute the full pairing with it (promising first experimental results)

▶ Toward AES-128 security level
  • explore supersingular pairing over $\mathbb{F}_{2^{nn'}}$ and $\mathbb{F}_{3^{nn'}}$ (work in progress)
  • genus-2 supersingular curves in characteristic 2 (work in progress)
  • Barreto–Naehrig curves
Thank you for your attention

Questions?
Outline of the talk

▶ Context

▶ Reduced Tate pairing

▶ Non-reduced Tate pairing

▶ Fully parallel Karatsuba multiplier

▶ Final Exponentiation

▶ Results & Conclusion

▶ Appendix
Detailed Karatsuba algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Splitting</th>
<th>Subproducts</th>
<th>Recomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-way split</td>
<td>$A \rightarrow A_L + X^{\lceil m/2 \rceil}A_H$</td>
<td>$p_H \leftarrow A_H \times B_H$</td>
<td>$S \leftarrow p_H X^{2\lceil m/2 \rceil}$</td>
</tr>
<tr>
<td></td>
<td>$B \rightarrow B_L + X^{\lceil m/2 \rceil}B_H$</td>
<td>$p_M \leftarrow A_M \times B_M$</td>
<td>$+ \left( p_M - p_H - p_L \right) X^{\lceil m/2 \rceil}$</td>
</tr>
<tr>
<td></td>
<td>$A_M \leftarrow A_H + A_L$</td>
<td>$p_L \leftarrow A_L \times B_L$</td>
<td>$+ p_L$</td>
</tr>
<tr>
<td></td>
<td>$B_M \leftarrow B_H + B_L$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-way split</td>
<td>$A \rightarrow A_0 + X^{\lceil m/3 \rceil}A_1 + X^{2\lceil m/3 \rceil}A_2$</td>
<td>$p_0 \leftarrow A_0 \times B_0$</td>
<td>$S \leftarrow p_0 X^{4\lceil m/3 \rceil}$</td>
</tr>
<tr>
<td></td>
<td>$B \rightarrow B_0 + X^{\lceil m/3 \rceil}B_1 + X^{2\lceil m/3 \rceil}B_2$</td>
<td>$p_1 \leftarrow A_1 \times B_1$</td>
<td>$+ \left( p_0' - p_1 - p_2 \right) X^{3\lceil m/3 \rceil}$</td>
</tr>
<tr>
<td></td>
<td>$A_{S_0} \leftarrow A_1 + A_2$</td>
<td>$p_2 \leftarrow A_2 \times B_2$</td>
<td>$+ \left( p_1' - p_0 + p_1 - p_2 \right) X^{2\lceil m/3 \rceil}$</td>
</tr>
<tr>
<td></td>
<td>$A_{S_1} \leftarrow A_0 + A_2$</td>
<td>$p_0' \leftarrow A_{S_0} \times B_{S_0}$</td>
<td>$+ \left( p_2' - p_1 - p_0 \right) X^{\lceil m/3 \rceil}$</td>
</tr>
<tr>
<td></td>
<td>$A_{S_2} \leftarrow A_1 + A_0$</td>
<td>$p_1' \leftarrow A_{S_1} \times B_{S_1}$</td>
<td>$+ p_0$</td>
</tr>
<tr>
<td></td>
<td>$B_{S_0} \leftarrow B_1 + B_2$</td>
<td>$p_2' \leftarrow A_{S_2} \times B_{S_2}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$B_{S_1} \leftarrow B_0 + B_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$B_{S_2} \leftarrow B_1 + B_0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Detailed odd–even split Karatsuba algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Splitting</th>
<th>Subproducts</th>
<th>Recomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-way split</td>
<td>( A \rightarrow A_E(X^2) + XA_O(X^2) )&lt;br&gt;( B \rightarrow B_E(X^2) + XB_O(X^2) )&lt;br&gt;( A_M \leftarrow A_O + A_E )&lt;br&gt;( B_M \leftarrow B_O + B_E )</td>
<td>( p_O \leftarrow A_O \times B_O )&lt;br&gt;( p_M \leftarrow A_M \times B_M )&lt;br&gt;( p_E \leftarrow A_E \times B_E )</td>
<td>( S \leftarrow (p_E + Xp_O)(X^2) )&lt;br&gt;( + X(p_M - p_E - p_O)(X^2) )</td>
</tr>
<tr>
<td>3-way split</td>
<td>( A \rightarrow A_0(X^3) +XA_1(X^3) + X^2A_2(X^3) )&lt;br&gt;( B \rightarrow B_0(X^3) + XB_1(X^3) + X^2B_2(X^3) )&lt;br&gt;( A_{S_0} \leftarrow A_1 + A_2 )&lt;br&gt;( A_{S_1} \leftarrow A_0 + A_2 )&lt;br&gt;( A_{S_2} \leftarrow A_1 + A_0 )&lt;br&gt;( B_{S_0} \leftarrow B_1 + B_2 )&lt;br&gt;( B_{S_1} \leftarrow B_0 + B_2 )&lt;br&gt;( B_{S_2} \leftarrow B_1 + B_0 )</td>
<td>( p_0 \leftarrow A_0 \times B_0 )&lt;br&gt;( p_1 \leftarrow A_1 \times B_1 )&lt;br&gt;( p_2 \leftarrow A_2 \times B_2 )&lt;br&gt;( p'<em>0 \leftarrow A</em>{S_0} \times B_{S_0} )&lt;br&gt;( p'<em>1 \leftarrow A</em>{S_1} \times B_{S_1} )&lt;br&gt;( p'<em>2 \leftarrow A</em>{S_2} \times B_{S_2} )</td>
<td>( S \leftarrow (p_0 + X(p'_0 - p_1 - p_2))(X^3) )&lt;br&gt;( + X(p'_2 - p_0 - p_1 + Xp_2)(X^3) )&lt;br&gt;( + X^2(p_1 + p'_1 - p_2 - p_0)(X^3) )</td>
</tr>
</tbody>
</table>