ECC 2010 — Redmond, USA

Faster Implementation of Pairings

Francisco Rodríguez-Henríquez
CINVESTAV, IPN, Mexico City, Mexico

Joint work with:

Jean-Luc Beuchat
LCIS, University of Tsukuba, Japan

Nicolas Brisebarre
Arénaire, LIP, ÉNS Lyon, France

Jérémy Detrey
Caramel, INRIA Nancy Grand-Est, France

Nicolas Estibals
Caramel, INRIA Nancy Grand-Est, France

Jorge González-Díaz
CINVESTAV, IPN, Mexico City, Mexico

Emmanuel López-Trejo
Intel Guadalajara Design Center, Mexico

Luis Martínez-Ramos
CINVESTAV, IPN, Mexico City, Mexico

Shigeo Mitsunari
Cybozu Labs, Inc., Tokyo, Japan

Eiji Okamoto
LCIS, University of Tsukuba, Japan

Tadanori Teruya
LCIS, University of Tsukuba, Japan
Outline of the talk

1. Context
2. Hardware accelerator for the Tate pairing over supersingular curves
3. Software accelerator for the Tate pairing over supersingular curves
4. Optimal Ate Pairing over Barreto-Naehrig Curves
Bilinear pairings

Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order
\[\#G_1 = \#G_2 = \ell\]

\((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(\#G_\tau = \ell\)
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(\#G_1 = \#G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(\#G_\tau = \ell\)
- A non-degenerate bilinear pairing is a map

\[ \hat{e} : G_1 \times G_2 \rightarrow G_\tau \]

that satisfies the following conditions:
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(\#G_1 = \#G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(\#G_\tau = \ell\)
- A non-degenerate bilinear pairing is a map

\[
\hat{e} : G_1 \times G_2 \rightarrow G_\tau
\]

that satisfies the following conditions:
- **non-degeneracy**: \(\hat{e}(P, P) \neq 1_{G_\tau}\) (equivalently \(\hat{e}(P, P) \) generates \(G_\tau\))
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(\#G_1 = \#G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(\#G_\tau = \ell\)
- A non-degenerate bilinear pairing is a map

\[
\hat{e} : G_1 \times G_2 \to G_\tau
\]

that satisfies the following conditions:

- non-degeneracy: \(\hat{e}(P, P) \neq 1_{G_\tau}\) (equivalently \(\hat{e}(P, P)\) generates \(G_\tau\))
- bilinearity:

\[
\hat{e}(Q_1 + Q_2, R) = \hat{e}(Q_1, R) \cdot \hat{e}(Q_2, R) \quad \hat{e}(Q, R_1 + R_2) = \hat{e}(Q, R_1) \cdot \hat{e}(Q, R_2)
\]
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(\#G_1 = \#G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(\#G_\tau = \ell\)
- A non-degenerate bilinear pairing is a map

\[ \hat{e} : G_1 \times G_2 \to G_\tau \]

that satisfies the following conditions:

- non-degeneracy: \(\hat{e}(P, P) \neq 1_{G_\tau}\) (equivalently \(\hat{e}(P, P)\) generates \(G_\tau\))
- 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
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(\#G_1 = \#G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(\#G_\tau = \ell\)
- A non-degenerate bilinear pairing is a map \(\hat{e} : G_1 \times G_2 \rightarrow G_\tau\)

that satisfies the following conditions:

- non-degeneracy: \(\hat{e}(P, P) \neq 1_{G_\tau}\) (equivalently \(\hat{e}(P, P)\) generates \(G_\tau\))
- 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

- Immediate property: for any two integers \(k_1\) and \(k_2\)
  \[\hat{e}(k_1 Q, k_2 R) = \hat{e}(Q, R)^{k_1 k_2}\]
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order 
  \#G_1 = \#G_2 = \ell

- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \#G_\tau = \ell

- A non-degenerate bilinear pairing is a map
  \[ \hat{e} : G_1 \times G_2 \rightarrow G_\tau \]

  that satisfies the following conditions:
  
  ▶ non-degeneracy: \(\hat{e}(P, P) \not= 1_{G_\tau}\) (equivalently \(\hat{e}(P, P)\) generates \(G_\tau\))
  
  ▶ 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

- Immediate property: for any two integers \(k_1\) and \(k_2\)
  
  \[ \hat{e}(k_1 Q, k_2 R) = \hat{e}(Q, R)^{k_1k_2} \]

- When \(G_1 = G_2\) we say that the pairing is symmetric, otherwise if \(G_1 \not= G_2\), the pairing is asymmetric.
Pairings in cryptography

- At first, used to attack supersingular elliptic curves
    \[ \text{DLP}_{G_1} \xrightarrow{<_P} \text{DLP}_{G_\tau} \]
    \[ kP \rightarrow \hat{e}(kP, P) = \hat{e}(P, P)^k \]
  - for cryptographic applications, we will also require the DLP in \( G_\tau \) to be hard
Pairings in cryptography

- At first, used to attack supersingular elliptic curves

\[
\text{DLP}_{G_1} \overset{<}{\longrightarrow} \text{DLP}_{G_\tau}
\]
\[
kP \rightarrow \hat{e}(kP, P) = \hat{e}(P, P)^k
\]

- for cryptographic applications, we will also require the DLP in $G_\tau$ to be hard

- One-round three-party key agreement (Joux, 2000)
Pairings in cryptography

- At first, used to attack supersingular elliptic curves
    \[
    \text{DLP}_{G_1} <_P \text{DLP}_{G_\tau}
    \]
    
    \[
    kP \rightarrow \hat{e}(kP, P) = \hat{e}(P, P)^k
    \]
  - for cryptographic applications, we will also require the DLP in \( G_\tau \) to be hard

- One-round three-party key agreement (Joux, 2000)

- Identity-based encryption
  - Boneh–Franklin, 2001
  - Sakai–Kasahara, 2001
Pairings in cryptography

- At first, used to attack supersingular elliptic curves
    \[
    \text{DLP}_{G_1} <_P \text{DLP}_{G_\tau}
    \]
    \[kP \rightarrow \hat{e}(kP, P) = \hat{e}(P, P)^k\]
  - for cryptographic applications, we will also require the DLP in \(G_\tau\) to be hard
- One-round three-party key agreement (Joux, 2000)
- Identity-based encryption
  - Boneh–Franklin, 2001
  - Sakai–Kasahara, 2001
- Short digital signatures
  - Boneh–Lynn–Shacham, 2001
  - Zang–Safavi-Naini–Susilo, 2004
- ...

...
The Tate Pairing over Supersingular elliptic curves

We first define

- $\mathbb{F}_q$, a finite field, with $q = 2^m$ or $3^m$
- $E$, an elliptic curve defined over $\mathbb{F}_q$
- $\ell$, a large prime factor of $\#E(\mathbb{F}_q)$
The Tate Pairing over Supersingular elliptic curves

- We first define
  - $\mathbb{F}_q$, a finite field, with $q = 2^m$ or $3^m$
  - $E$, an elliptic curve defined over $\mathbb{F}_q$
  - $\ell$, a large prime factor of $\#E(\mathbb{F}_q)$

- $G_1 = E(\mathbb{F}_q)[\ell]$, the $\mathbb{F}_q$-rational $\ell$-torsion of $E$:

$$G_1 = \{ P \in E(\mathbb{F}_q) \mid \ell P = \mathcal{O} \}$$
The Tate Pairing over Supersingular elliptic curves

- We first define
  - $\mathbb{F}_q$, a finite field, with $q = 2^m$ or $3^m$
  - $E$, an elliptic curve defined over $\mathbb{F}_q$
  - $\ell$, a large prime factor of $\#E(\mathbb{F}_q)$

- $G_1 = E(\mathbb{F}_q)[\ell]$, the $\mathbb{F}_q$-rational $\ell$-torsion of $E$:
  \[ G_1 = \{ P \in E(\mathbb{F}_q) \mid \ell P = O \} \]

- $G_\tau = \mu_\ell$, the group of $\ell$-th roots of unity in $\mathbb{F}_{q^k}^*$:
  \[ G_\tau = \{ U \in \mathbb{F}_{q^k}^* \mid U^\ell = 1 \} \]
The Tate Pairing over Supersingular elliptic curves

- We first define
  - $\mathbb{F}_q$, a finite field, with $q = 2^m$ or $3^m$
  - $E$, an elliptic curve defined over $\mathbb{F}_q$
  - $\ell$, a large prime factor of $\#E(\mathbb{F}_q)$

- $G_1 = E(\mathbb{F}_q)[\ell]$, the $\mathbb{F}_q$-rational $\ell$-torsion of $E$:

\[ G_1 = \{ P \in E(\mathbb{F}_q) \mid \ell P = \mathcal{O} \} \]

- $G_\tau = \mu_\ell$, the group of $\ell$-th roots of unity in $\mathbb{F}_q^\times$:

\[ G_\tau = \{ U \in \mathbb{F}_q^\times \mid U^\ell = 1 \} \]

- $k$ is the embedding degree, the smallest integer such that $\mu_\ell \subseteq \mathbb{F}_q^\times$
  - usually large for ordinary elliptic curves
  - bounded in the case of supersingular elliptic curves
    (4 in characteristic 2; 6 in characteristic 3; and 2 in characteristic $> 3$)
Security considerations for Symmetric Pairings

\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_p^{\times k_m} \]

- The discrete logarithm problem should be hard in both \( G_1 \) and \( G_T \)
Security considerations for Symmetric Pairings

\[ \hat{e} : E(F_{p^m})[\ell] \times E(F_{p^m})[\ell] \to \mu_\ell \subseteq F_{p^{km}}^\times \]

- The discrete logarithm problem should be hard in both \( G_1 \) and \( G_T \)

<table>
<thead>
<tr>
<th>Base field ( (F_{p^m}) )</th>
<th>( F_{2m} )</th>
<th>( F_{3m} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lower security</strong> ( (\sim 2^{64}) )</td>
<td>( m = 239 )</td>
<td>( m = 97 )</td>
</tr>
<tr>
<td><strong>Medium security</strong> ( (\sim 2^{80}) )</td>
<td>( m = 373 )</td>
<td>( m = 163 )</td>
</tr>
<tr>
<td><strong>Higher security</strong> ( (\sim 2^{128}) )</td>
<td>( m = 1103 )</td>
<td>( m = 503 )</td>
</tr>
</tbody>
</table>

- \( F_{2m} \): simpler finite field arithmetic
- \( F_{3m} \): smaller field extension
Computation of the Tate pairing

\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_{p^{km}}^\times \]
Computation of the Tate pairing

\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \to \mu_\ell \subseteq \mathbb{F}_{p^km} \]

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)
Computation of the Tate pairing
\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_{p^{km}}^\times \]

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)

- Arithmetic over \( \mathbb{F}_{p^{km}}^\times \):
  - tower-field representation
  - only arithmetic over the underlying field \( \mathbb{F}_{p^m} \)
Computation of the Tate pairing
\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_{p^{km}}^\times \]

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)

- Arithmetic over \( \mathbb{F}_{p^{km}}^\times \):
  - tower-field representation
  - only arithmetic over the underlying field \( \mathbb{F}_{p^m} \)

- Operations over \( \mathbb{F}_{p^m} \):
  - \( O(m) \) additions / subtractions
  - \( O(m) \) multiplications
  - \( O(m) \) Frobenius maps (\( a \mapsto a^p \), i.e. squarings or cubings)
  - 1 inversion
Computation of the Tate pairing
\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \to \mu_\ell \subseteq \mathbb{F}_{p^{km}}^\times \]

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)

- Arithmetic over \( \mathbb{F}_{p^{km}}^\times \):
  - tower-field representation
  - only arithmetic over the underlying field \( \mathbb{F}_{p^m} \)

- Operations over \( \mathbb{F}_{p^m} \):
  - \( O(m) \) additions / subtractions
  - \( O(m) \) multiplications
  - \( O(m) \) Frobenius maps (\( a \mapsto a^p \), i.e. squarings or cubings)
  - 1 inversion

- A first idea: an all-in-one unified operator:
  - shared resources
  - scalable architecture
Motivations

- High speed is more important than low resources for some cryptographic applications
- Explore the other end of the area vs. time tradeoff:
  - faster but larger than the unified operator
  - what about the area-time product?
Motivations

- High speed is more important than low resources for some cryptographic applications.
- Explore the other end of the area vs. time tradeoff:
  - faster but larger than the unified operator
  - what about the area-time product?
- Accelerate the computation by extracting as much parallelism as possible...
- ... Without increasing dramatically the resource requirements.
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps

\[ \hat{e}(P, Q) \]
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps

$$\hat{e}(P, Q) = \eta_T(P, Q)$$
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps

\[ \hat{e}(P, Q) = \eta_T(P, Q)^M \]
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps
  \[ \hat{e}(P, Q) = \eta_T(P, Q)^M \]

- Computation of the $\eta_T$ pairing
  - via Miller’s algorithm: loop of $(m + 1)/2$ iterations
  - result only defined modulo $N$-th powers in $\mathbb{F}_{p^m}^\times$, with $N = \#E(\mathbb{F}_{p^m})$
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps
  \[ \hat{e}(P, Q) = \eta_T(P, Q)^M \]

- Computation of the $\eta_T$ pairing
  - via Miller's algorithm: loop of $(m + 1)/2$ iterations
  - result only defined modulo $N$-th powers in $\mathbb{F}_{p^km}^\times$, with $N = \#E(\mathbb{F}_{p^m})$

- Final exponentiation by $M = (p^{km} - 1)/N$
  - required to obtain a unique value for each congruence class
  - example in characteristic 3 ($k = 6$ and $N = 3^m + 1 \pm 3^{(m+1)/2}$):
  \[
  M = \frac{3^{6m} - 1}{3m + 1 \pm 3^{(m+1)/2}} = (3^{3m} - 1)(3^m + 1)(3^m + 1 \mp 3^{(m+1)/2})
  \]
  - exploit the special form of the exponent: *ad-hoc* algorithm
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps
  $$\hat{e}(P, Q) = \eta_T(P, Q)^M$$

- Computation of the $\eta_T$ pairing
  - via Miller’s algorithm: loop of $(m+1)/2$ iterations
  - result only defined modulo $N$-th powers in $\mathbb{F}_p^{\times}$, with $N = \#E(\mathbb{F}_{p^m})$

- Final exponentiation by $M = (p^{km} - 1)/N$
  - required to obtain a unique value for each congruence class
  - example in characteristic 3 ($k = 6$ and $N = 3^m + 1 \pm 3^{(m+1)/2}$):
    $$M = \frac{3^{6m} - 1}{3^m + 1 \pm 3^{(m+1)/2}} = (3^{3m} - 1)(3^m + 1)\left(3^m + 1 \mp 3^{(m+1)/2}\right)$$
    - exploit the special form of the exponent: *ad-hoc* algorithm

- Two distinct computational requirements
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps
  
  \[ \hat{e}(P, Q) = \eta_T(P, Q)^M \]

- Computation of the $\eta_T$ pairing
  
  ▶ via Miller’s algorithm: loop of $(m+1)/2$ iterations
  ▶ result only defined modulo $N$-th powers in $\mathbb{F}_{p^km}^\times$, with $N = \#E(\mathbb{F}_{p^m})$

- Final exponentiation by $M = (p^{km} - 1)/N$
  
  ▶ required to obtain a unique value for each congruence class
  ▶ example in characteristic 3 ($k = 6$ and $N = 3^m + 1 \pm 3^{(m+1)/2}$):

  \[ M = \frac{3^{6m} - 1}{3^m + 1 \pm 3^{(m+1)/2}} = (3^{3m} - 1)(3^m + 1)\left(3^m + 1 \mp 3^{(m+1)/2}\right) \]

  ▶ exploit the special form of the exponent: ad-hoc algorithm

- Two distinct computational requirements ⇒ use two distinct coprocessors
Reduced Tate pairing

\[ E(F^{3m})[\ell] \]

\[ \mu^\ell \subseteq F \times 3^{6m} \]

Non-reduced pairing (iterative computation) (irregular exponentiation Final algorithm)

Input: two points \( P \) and \( Q \) in \( E(F^{3m})[\ell] \)

Output: an \( \ell \)-th root of unity in the extension \( F \times 3^{6m} \)

Two very different steps

Francisco Rodríguez-Henríquez

Faster Implementation of Pairings
Reduced Tate pairing

\[ E(\mathbb{F}_{3^m})[\ell] \]

**Input:** two points \( P \) and \( Q \) in \( E(\mathbb{F}_{3^m})[\ell] \)
### Reduced Tate pairing

- **Input:** two points $P$ and $Q$ in $E(\mathbb{F}_{3^m})[\ell]$
- **Output:** an $\ell$-th root of unity in the extension $\mathbb{F}_{3^{6m}}^\times$
Reduced Tate pairing

Input: two points $P$ and $Q$ in $E(\mathbb{F}_{3^m})[\ell]$

Output: an $\ell$-th root of unity in the extension $\mathbb{F}_{3^6m}^\times$

Two very different steps
Reduced Tate pairing

- **Input:** two points $P$ and $Q$ in $E(\mathbb{F}_{3^m})[\ell]$
- **Output:** an $\ell$-th root of unity in the extension $\mathbb{F}_{3^{6m}}^\times$
- **Two very different steps**
Reduced Tate pairing

- **Input**: two points $P$ and $Q$ in $E(\mathbb{F}_{3^m})[\ell]$
- **Output**: an $\ell$-th root of unity in the extension $\mathbb{F}_{3^{6m}}^\times$
- **Two very different steps**

![Diagram showing the process of reduced Tate pairing](image)
Reduced Tate pairing

- **Input**: two points $P$ and $Q$ in $E(\mathbb{F}_{3^m})[\ell]
- **Output**: an $\ell$-th root of unity in the extension $\mathbb{F}_{3^6m}^\times$
- **Two very different steps**
Two coprocessors for the $\eta_T$ pairing

- The two operations are purely sequential
- Only one active coprocessor at every moment
Two coprocessors for the $\eta_T$ pairing

- The two operations are purely sequential
- Only one active coprocessor at every moment
- Pipeline the data between the two coprocessors
Two coprocessors for the $\eta_T$ pairing

- The two operations are purely sequential.
- Only one active coprocessor at every moment.
- Pipeline the data between the two coprocessors.
  - Both of them are kept busy.
  - Higher throughput.
Two coprocessors for the $\eta_T$ pairing

- The two operations are purely sequential
- Only one active coprocessor at every moment
- Pipeline the data between the two coprocessors
  - both of them are kept busy
  - higher throughput
- Balance the computation time between the two coprocessors
\( \eta_T \) pairing algorithm

\[ \eta_T : E(\mathbb{F}_{3^m})[\ell] \times E(\mathbb{F}_{3^m})[\ell] \rightarrow \mathbb{F}_{3^{6m}}^\times \]

- **Three tasks** per iteration:
  1. update the coordinates
  2. compute the line equation
  3. accumulate the new factor

- **Total cost**: 17 ×, 4 Frobenius/inverse Frobenius and 30 + over \( \mathbb{F}_{3^m} \)
\( \eta_T \) pairing algorithm

\[ \eta_T : E(\mathbb{F}_3^m)[\ell] \times E(\mathbb{F}_3^m)[\ell] \rightarrow \mathbb{F}_3^{6m} \]

- **Three tasks** per iteration:
  1. update the coordinates
  2. compute the line equation
  3. accumulate the new factor
- **Total cost**: \( 17 \times, 4 \) Frobenius/inverse Frobenius and \( 30 + \) over \( \mathbb{F}_3^m \)
- **Cost of the inverse Frobenius**: Same as the Frobenius

```plaintext
for i ← 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 inv. Frobenius \( (\mathbb{F}_3^m) \)
    2 Frobenius

② \( t \leftarrow x_p + x_Q \); \( u \leftarrow y_p y_Q \)
    \( S \leftarrow -t^2 \pm u\sigma - t\rho - \rho^2 \)
    2 ×, 1 + \( (\mathbb{F}_3^m) \)

③ \( R \leftarrow R \cdot S \)
    15 ×, 29 + \( (\mathbb{F}_3^m) \)

end for
```
Accelerating the $\eta_T$ pairing

- Total cost: $17 \times$, 2 Frobenius and inverse Frobenius and $30 +$ over $\mathbb{F}_{3^m}$ per iteration
  - Frobenius/inverse Frobenius and $+$: cheap and fast operations
Accelerating the $\eta_T$ pairing

- Total cost: $17 \times, 2$ Frobenius and inverse Frobenius and $30 +$ over $\mathbb{F}_{3^m}$ per iteration
  - Frobenius/inverse Frobenius and $+$: cheap and fast operations
  - Critical operation: $\times$

- Need for a fast parallel multiplier: Karatsuba
Accelerating the $\eta_T$ pairing

- Total cost: $17 \times$, 2 Frobenius and inverse Frobenius and $30 +$ over $\mathbb{F}_{3^m}$ per iteration
  - Frobenius/inverse Frobenius and $+$: cheap and fast operations
  - Critical operation: $\times$
- Need for a fast parallel multiplier: Karatsuba

\[
\begin{align*}
A^H B^L + A^L B^H &= (A^H + A^L)(B^H + B^L) - A^H B^H - A^L B^L
\end{align*}
\]
A parallel Karatsuba multiplier

- fully parallel: all sub-products are computed in parallel
- pipelined architecture: higher clock frequency, one product per cycle
A parallel Karatsuba multiplier

- fully parallel: all sub-products are computed in parallel
- pipelined architecture: higher clock frequency, one product per cycle
- sub-products recursively implemented as Karatsuba-Ofman multipliers
A parallel Karatsuba multiplier

- fully parallel: all sub-products are computed in parallel
- pipelined architecture: higher clock frequency, one product per cycle
- sub-products recursively implemented as Karatsuba-Ofman multipliers
- support for other variants: odd-even split, 3-way split, ...
A parallel Karatsuba multiplier

- fully parallel: all sub-products are computed in parallel
- pipelined architecture: higher clock frequency, one product per cycle
- sub-products recursively implemented as Karatsuba-Ofman multipliers
- support for other variants: odd-even split, 3-way split, ...
- final reduction modulo the irreducible polynomial $f$
Accelerating the $\eta_T$ pairing

- $\eta_T$ coprocessor based on a single large multiplier:
  - parallel Karatsuba architecture
  - 7-stage pipeline
  - one product per cycle

Challenge: keep the multiplier busy at all times
- Careful scheduling to avoid pipeline bubbles (idle cycles):
  - ensure that multiplication operands are always available
  - avoid memory congestion issues

We managed to accomplish that: our processor computes Miller loop in just $17 \cdot (m + 3)/2$ clock cycles (considering the initialization phase)
Accelerating the $\eta_T$ pairing

- $\eta_T$ coprocessor based on a single large multiplier:
  - parallel Karatsuba architecture
  - 7-stage pipeline
  - one product per cycle

- **Challenge**: keep the multiplier busy at all times
Accelerating the $\eta_T$ pairing

- $\eta_T$ coprocessor based on a single large multiplier:
  - parallel Karatsuba architecture
  - 7-stage pipeline
  - one product per cycle

- Challenge: keep the multiplier busy at all times

- Careful scheduling to avoid pipeline bubbles (idle cycles):
  - ensure that multiplication operands are always available
  - avoid memory congestion issues

- We managed to accomplish that: our processor computes Miller loop in just $17 \cdot (m + 3)/2$ clock cycles (considering the initialization phase)
A parallel operator for the $\eta_T$ pairing
The final exponentiation

- Compute $\hat{e}(P, Q)$ as $\eta_T(P, Q)^M$ with $\eta_T(P, Q) \in \mathbb{F}_{36m}^\times$ and

$$M = (3^{3m} - 1)(3^m + 1) \left(3^m + 1 \mp 3^{(m+1)/2}\right)$$
The final exponentiation

- Compute \( \hat{e}(P, Q) \) as \( \eta_T(P, Q)^M \) with \( \eta_T(P, Q) \in \mathbb{F}_{36m}^\times \) and

\[
M = (3^{3m} - 1) (3^m + 1) \left( 3^m + 1 \mp 3^{(m+1)/2} \right)
\]

- Operations over \( \mathbb{F}_{3m} \): \( 73 \times \), \( 3m + 3 \) Frobenius, \( 3m + 175 + \), and \( 1 \) inversion
The final exponentiation

- Compute $\hat{e}(P, Q)$ as $\eta_T(P, Q)^M$ with $\eta_T(P, Q) \in \mathbb{F}^\times_{36m}$ and

$$M = (3^{3m} - 1)(3^m + 1) \left(3^m + 1 \mp 3^{(m+1)/2}\right)$$

- Operations over $\mathbb{F}_{3^m}$: $73 \times$, $3m + 3$ Frobenius, $3m + 175 +$, and $1$ inversion ($\sim \log m \times$ and $m - 1$ Frobenius)
The final exponentiation

- Compute $\hat{e}(P, Q)$ as $\eta_T(P, Q)^M$ with $\eta_T(P, Q) \in \mathbb{F}_{36m}$ and

  $$M = (3^{3m} - 1) (3^m + 1) \left(3^m + 1 \mp 3^{(m+1)/2}\right)$$

- Operations over $\mathbb{F}_{3^m}$: $73 \times$, $3^m + 3$ Frobenius, $3^m + 175 +$, and $1$ inversion ($\sim \log m \times$ and $m - 1$ Frobenius)

- Cost of the $\eta_T$ pairing:
  - $(m + 1)/2$ iterations
  - $17 \times$, $10$ Frobenius and $30 +$ over $\mathbb{F}_{3^m}$ per iteration
The final exponentiation

- Compute \( \hat{e}(P, Q) \) as \( \eta_T(P, Q)^M \) with \( \eta_T(P, Q) \in \mathbb{F}_{36m}^\times \) and

\[
M = (3^{3m} - 1) (3^m + 1) \left(3^m + 1 \mp 3^{(m+1)/2}\right)
\]

- Operations over \( \mathbb{F}_{3m} \): \( 73 \times, 3m + 3 \) Frobenius, \( 3m + 175 +, \) and \( 1 \) inversion (\( \sim \log m \times \) and \( m - 1 \) Frobenius)

- Cost of the \( \eta_T \) pairing:
  - \( (m + 1)/2 \) iterations
  - \( 17 \times, 10 \) Frobenius and \( 30 + \) over \( \mathbb{F}_{3m} \) per iteration

- The final exponentiation is much cheaper than the \( \eta_T \) pairing

- Challenge for the final exponentiation:
  - computation in the same time as the \( \eta_T \) pairing
  - ... using as few resources as possible
The final exponentiation

- Design the **smallest architecture** possible supporting all the required operations over $\mathbb{F}_{3^m}$
- **purely sequential** scheduling
The final exponentiation

- Design the smallest architecture possible supporting all the required operations over $\mathbb{F}_{3^m}$
- purely sequential scheduling
- Although some parallelism is required.
The final exponentiation

- Design the **smallest architecture** possible supporting all the required operations over $\mathbb{F}_{3^m}$
- purely sequential scheduling
- Although some parallelism is required.
- We found out that the usage of the inverse Frobenius operator is advantageous for computing the final exponentiation (as long as the irreducible polynomials are inverse-Frobenius friendly)
The final exponentiation

- Design the smallest architecture possible supporting all the required operations over $\mathbb{F}_{3^m}$
- purely sequential scheduling
- Although some parallelism is required.
- We found out that the usage of the inverse Frobenius operator is advantageous for computing the final exponentiation (as long as the irreducible polynomials are inverse-Frobenius friendly)
- New coprocessor with two arithmetic units:
  - a standalone multiplier, based on a parallel-serial scheme
  - a unified operator supporting addition/subtraction, inverse Frobenius map and inverse double Frobenius map
A coprocessor for the final exponentiation
Agenda

1. Context

2. Hardware accelerator for the Tate pairing over supersingular curves
   - Implementation Results in Hardware

3. Software accelerator for the Tate pairing over supersingular curves
   - Computing the non-reduced pairing
   - Final exponentiation
   - Implementation results

4. Optimal Ate Pairing over Barreto-Naehrig Curves
   - Barreto–Naehrig Curves
Hardware accelerators

<table>
<thead>
<tr>
<th>Security [bits]</th>
<th>Virtex-II Pro</th>
<th>Virtex-4 LX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$6.2 \mu s / F_3^{97}$</td>
<td>$20.9 \mu s / F_3^{397}$</td>
</tr>
<tr>
<td></td>
<td>$12.8 \mu s / F_3^{193}$</td>
<td>$16.9 \mu s / F_3^{313}$</td>
</tr>
<tr>
<td></td>
<td>$20.9 \mu s / F_3^{397}$</td>
<td>$100.8 \mu s / F_2^{457}$</td>
</tr>
<tr>
<td></td>
<td>$675.5 \mu s / F_2^{557}$</td>
<td>$100.8 \mu s / F_2^{457}$</td>
</tr>
</tbody>
</table>

Calculation time [µs]

Francisco Rodríguez-Henríquez
Hardware implementation notes

- Our Xilinx FPGA implementation, significantly improved the computation time of all the hardware pairing coprocessors for supersingular curves previously published.
Hardware implementation notes

- Our Xilinx FPGA implementation, significantly improved the computation time of all the hardware pairing coprocessors for supersingular curves previously published.
- (a bit Surprisingly) our architecture also enjoys the best area/time trade-off performance among supersingular pairing accelerators.

In the design process of our char 2 accelerator we found the following undocumented family of square-root friendly irreducible pentanomials:

\[ f(x) = x^m + x^{m-2}d + x^d + 1. \]

All technical details of these designs can be found in the preprint manuscripts eprint 2009/122 and eprint 2009/398.
Hardware implementation notes

- Our Xilinx FPGA implementation, significantly improved the computation time of all the hardware pairing coprocessors for supersingular curves previously published.
- (a bit Surprisingly) our architecture also enjoys the best area/time trade-off performance among supersingular pairing accelerators.
- However, because we exceeded the FPGA’s capacity, we could only achieve up to 109 bits of security.

In the design process of our char 2 accelerator we found the following undocumented family of square-root friendly irreducible pentanomials:

\[ f(x) = x^m + x^m - d + x^m - 2d + xd + 1. \]

All technical details of these designs can be found in the preprint manuscripts eprint 2009/122 and eprint 2009/398.
Hardware implementation notes

- Our Xilinx FPGA implementation, significantly improved the computation time of all the hardware pairing coprocessors for supersingular curves previously published.

- (a bit Surprisingly) our architecture also enjoys the best area/time trade-off performance among supersingular pairing accelerators.

- However, because we exceeded the FPGA’s capacity, we could only achieve up to 109 bits of security.

- Although it was not discussed here, we also implemented the Tate pairing over char 2. Experimentally, we observed that our char 2 and char 3 accelerators achieve almost the same time performance.
Hardware implementation notes

- Our Xilinx FPGA implementation, significantly improved the computation time of all the hardware pairing coprocessors for supersingular curves previously published.
- (a bit Surprisingly) our architecture also enjoys the best area/time trade-off performance among supersingular pairing accelerators.
- However, because we exceeded the FPGA’s capacity, we could only achieve up to 109 bits of security.
- Although it was not discussed here, we also implemented the Tate pairing over char 2. Experimentally, we observed that our char 2 and char 3 accelerators achieve almost the same time performance.
- In the design process of our char 2 accelerator we found the following undocumented family of square-root friendly irreducible pentanomials: 
  \[ f(x) = x^m + x^{m-d} + x^{m-2d} + x^d + 1. \]
- All technical details of these designs can be found in the preprint manuscripts eprint 2009/122 and eprint 2009/398.
Agenda

1. Context

2. Hardware accelerator for the Tate pairing over supersingular curves
   - Implementation Results in Hardware

3. Software accelerator for the Tate pairing over supersingular curves
   - Computing the non-reduced pairing
   - Final exponentiation
   - Implementation results

4. Optimal Ate Pairing over Barreto-Naehrig Curves
   - Barreto–Naehrig Curves
Computing the non-reduced pairing

- $\eta_T$ pairing: shorter loop

```
for i ← 0 to (m − 1)/2 do
  end for
```
Computing the non-reduced pairing

- $\eta_T$ pairing: shorter loop
- Based on Miller’s algorithm:

\[
\text{for } i \leftarrow 0 \text{ to } (m - 1)/2 \text{ do} \\
\quad x_P \leftarrow \sqrt[3]{x_P} \quad ; \quad y_P \leftarrow \sqrt[3]{y_P} \\
\quad x_Q \leftarrow x_Q^3 \quad ; \quad y_Q \leftarrow y_Q^3 \\
\quad t \leftarrow x_P + x_Q \quad u \leftarrow y_P y_Q \\
\quad S \leftarrow -t^2 \pm u\sigma - t\rho - \rho^2 \\
\quad R \leftarrow R \cdot S \\
\text{end for}
\]
Computing the non-reduced pairing

- $\eta_T$ pairing: shorter loop
- Based on Miller’s algorithm:
  - 1. update of point coordinates

```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
```
Computing the non-reduced pairing

- **$\eta_T$ pairing**: shorter loop
- Based on **Miller’s algorithm**:
  1. update of point coordinates
  2. computation of line equation

```
for i ← 0 to (m − 1)/2 do
    \[\begin{align*}
    x_P &\leftarrow \sqrt[3]{x_P} ; \quad y_P &\leftarrow \sqrt[3]{y_P} \\
    x_Q &\leftarrow x_Q^3 ; \quad y_Q &\leftarrow y_Q^3
    \end{align*}\]

    \[\begin{align*}
    t &\leftarrow x_P + x_Q \quad ; \quad u &\leftarrow y_P y_Q \\
    S &\leftarrow -t^2 \pm u \sigma - t \rho - \rho^2
    \end{align*}\]

        \[R \leftarrow R \cdot S\]
end for
```
Computing the non-reduced pairing

- $\eta_T$ pairing: shorter loop
- Based on Miller's algorithm:
  1. update of point coordinates
  2. computation of line equation
  3. accumulation of the new factor

\[
\text{for } i \leftarrow 0 \text{ to } (m - 1)/2 \text{ do}
\]
\[
\begin{align*}
x_P & \leftarrow 3\sqrt{x_P} & y_P & \leftarrow 3\sqrt{y_P} & 2 \cdot 3^{\frac{1}{2}} \\
x_Q & \leftarrow x_Q^3 & y_Q & \leftarrow y_Q^3 & 2 \cdot (\cdot)^3 \\
t & \leftarrow x_P + x_Q & u & \leftarrow y_P y_Q & 2 \times 2^{\frac{1}{2}} + \\
S & \leftarrow -t^2 \pm u\sigma - t\rho - \rho^2 & 1 \times (\mathbb{F}_{3^m}) \\
R & \leftarrow R \cdot S & \end{align*}
\]
end for
Computing the non-reduced pairing

- $\eta_T$ 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
- Comb right-to-left multiplier over $\mathbb{F}_{3^m}$

\begin{verbatim}
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 ← $-t^2 \pm u \sigma - t \rho - \rho^2$
    R ← R · S
end for
\end{verbatim}
Computing the non-reduced pairing

- $\eta_T$ 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
- Comb right-to-left multiplier over $\mathbb{F}_{3^m}$
- Sparse multiplication over $\mathbb{F}_{3^6m}$

```
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
```
Computing the non-reduced pairing

- $\eta_T$ 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
- Comb right-to-left multiplier over $\mathbb{F}_{3^m}$
- Sparse multiplication over $\mathbb{F}_{3^{6m}}$
  - 15 $\times$ and 29 $+$ over $\mathbb{F}_{3^m}$ (Beuchat et al., ARITH 18)

```plaintext
for i ← 0 to (m − 1)/2 do
    $x_P \leftarrow \sqrt[3]{x_P}$; $y_P \leftarrow \sqrt[3]{y_P}$
    $x_Q \leftarrow x_Q^3$; $y_Q \leftarrow y_Q^3$

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

    $R \leftarrow R \cdot S$
end for
```

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

- 15 $\times$ and 29 $+$ over $\mathbb{F}_{3^m}$ (Beuchat et al., ARITH 18)
Computing the non-reduced pairing

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

```plaintext
for i ← 0 to (m − 1)/2 do
    \( x_P \leftarrow 3^{\sqrt{2}} x_P \); \( y_P \leftarrow 3^{\sqrt{2}} y_P \)
    \( x_Q \leftarrow x_Q^3 \); \( y_Q \leftarrow y_Q^3 \)
    \( t \leftarrow x_P + x_Q \); \( u \leftarrow y_P y_Q \)
    \( S \leftarrow -t^2 \pm u \sigma - t \rho - \rho^2 \)
    \( R \leftarrow R \cdot S \)
end for
```
Computing the non-reduced pairing

First core

\[
\text{for } i \leftarrow 1 \text{ to } (m - 1)/2 \text{ do}
\]

\[
\begin{align*}
\text{(1) } & x_P[i] \leftarrow \sqrt[3]{x_P[i-1]} \ , \ y_P[i] \leftarrow \sqrt[3]{y_P[i-1]} \quad 2 \sqrt[3]{\cdot} \\
\text{(2) } & x_Q[i] \leftarrow x_Q[i-1]^3 \ , \ y_Q[i] \leftarrow y_Q[i-1]^3 \quad 2 (\cdot)^3 \\
\text{end for}
\end{align*}
\]

\[
\text{for } i \leftarrow 1 \text{ to } (m - 1)/2 \text{ do}
\]

\[
\begin{align*}
\text{(1) } & t \leftarrow x_P[i] + x_Q[i] \\
\text{(2) } & u \leftarrow y_P[i]y_Q[i] \\
\text{S} & \leftarrow -t^2 \pm u\sigma - t\rho - \rho^2 \\
\text{(3) } & R \leftarrow R \cdot S \quad 12 \times, 59 +
\end{align*}
\]

end for
Computing the non-reduced pairing

**First core**

```markdown
for i ← 1 to (m − 1)/2 do
  \( x_P[i] ← \sqrt{x_P[i - 1]} \); \( y_P[i] ← \sqrt{y_P[i - 1]} \)
  \( x_Q[i] ← x_Q[i - 1]^3 \); \( y_Q[i] ← y_Q[i - 1]^3 \)
end for
```

```markdown
for i ← 1 to (m − 1)/4 do
  \( t ← x_P[i] + x_Q[i] \)
  \( u ← y_P[i] y_Q[i] \)
  \( S ← -t^2 ± u\sigma - t\rho - \rho^2 \)
  \( R_0 ← R_0 \cdot S \)
end for
```

**Second core**

```markdown
for i ← (m − 1)/4 + 1 to (m − 1)/2 do
  \( t ← x_P[i] + x_Q[i] \)
  \( u ← y_P[i] y_Q[i] \)
  \( S ← -t^2 ± u\sigma - t\rho - \rho^2 \)
  \( R_1 ← R_1 \cdot S \)
end for
```
Computing the non-reduced pairing

First core

for $i \leftarrow 1 \text{ to } (m - 1)/2$ do

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

end for

for $i \leftarrow 1 \text{ to } (m - 1)/4$ do

2. $t \leftarrow x_P[i] + x_Q[i]$
3. $u \leftarrow y_P[i]y_Q[i]$
4. $S \leftarrow -t^2 \pm u\sigma - t\rho - \rho^2$
5. $R_0 \leftarrow R_0 \cdot S$

end for

$R \leftarrow R_0 \cdot R_1$

Second core

for $i \leftarrow (m - 1)/4 + 1 \text{ to } (m - 1)/2$ do

2. $t \leftarrow x_P[i] + x_Q[i]$
3. $u \leftarrow y_P[i]y_Q[i]$
4. $S \leftarrow -t^2 \pm u\sigma - t\rho - \rho^2$
5. $R_1 \leftarrow R_1 \cdot S$

end for

Francisco Rodríguez-Henríquez

Faster Implementation of Pairings
Computing the non-reduced pairing

**First core**

| i | $x_P[i] \leftarrow \sqrt[3]{x_P[i - 1]}$; $y_P[i] \leftarrow \sqrt[3]{y_P[i - 1]}$; $x_Q[i] \leftarrow x_Q[i - 1]^3$; $y_Q[i] \leftarrow y_Q[i - 1]^3$ |
|---|---|---|---|
| 1 | $R \leftarrow R \cdot S$ | $12 \times, 59 +$ |

**Second core**

| i | $t \leftarrow x_P[2i - 1] + x_Q[2i - 1]$; $u \leftarrow y_P[2i - 1]y_Q[2i - 1]$; $S \leftarrow -t^2 \pm u \sigma - t \rho - \rho^2$ |
|---|---|---|---|
| 1 | $R \leftarrow R \cdot S$; $R_0 \leftarrow R_0 \cdot S$; $R_1 \leftarrow R_1 \cdot S$ | $12 \times, 59 +$ |

$R \leftarrow R_0 \cdot R_1$ $15 \times, 67 +$
Computing the non-reduced pairing

**First core**

\[
\begin{align*}
&\text{for } i \leftarrow 1 \text{ to } (m - 1)/2 \text{ do} \\
&\quad \begin{array}{l}
\; 1 \ x_p[i] \leftarrow \sqrt[x_p[i - 1]]{\sqrt[x_p[i - 1]]{y_p[i]}} \\
\; 2 \ \sqrt[3]{y_p[i]} \leftarrow y_p[i - 1] \\
\; 3 \ x_q[i] \leftarrow x_q[i - 1] \\
\; 1 \ y_q[i] \leftarrow y_q[i - 1] \\
\end{array}
\end{align*}
\]

end for

**Second core**

\[
\begin{align*}
&\text{for } i \leftarrow 1 \text{ to } (m - 1)/4 \text{ do} \\
&\quad \begin{array}{l}
\; 1 \ x_p[i] \leftarrow x_p[2i - 1] + x_q[2i - 1] \\
\; 1 \ y_p[i] \leftarrow y_p[2i - 1]y_q[2i - 1] \\
\; 1 \ x_q[i] \leftarrow x_q[2i] \\
\; 1 \ y_q[i] \leftarrow y_q[2i] \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\; 1 \ t_0 \leftarrow x_p[2i - 1] + x_q[2i - 1] \\
\; 1 \ u_0 \leftarrow y_p[2i - 1]y_q[2i - 1] \\
\; 1 \ t_1 \leftarrow x_p[2i] + x_q[2i] \\
\; 1 \ u_1 \leftarrow y_p[2i]y_q[2i] \\
\end{align*}
\]

\[
\begin{align*}
\; 2 \ S \leftarrow (-t_0^2 \pm u_0 \sigma - t_0 \rho - \rho^2), \\
\quad (t_1^2 \pm u_1 \sigma - t_1 \rho - \rho^2) \\
\end{align*}
\]

\[
\begin{align*}
\; 3 \ R_0 \leftarrow R_0 \cdot S \\
\; 15 \times, 67 + \\
\end{align*}
\]

end for

\[
\begin{align*}
\; 1 \ t_0 \leftarrow x_p[2i - 1] + x_q[2i - 1] \\
\; 1 \ u_0 \leftarrow y_p[2i - 1]y_q[2i - 1] \\
\; 1 \ t_1 \leftarrow x_p[2i] + x_q[2i] \\
\; 1 \ u_1 \leftarrow y_p[2i]y_q[2i] \\
\end{align*}
\]

\[
\begin{align*}
\; 2 \ S \leftarrow (-t_0^2 \pm u_0 \sigma - t_0 \rho - \rho^2), \\
\quad (t_1^2 \pm u_1 \sigma - t_1 \rho - \rho^2) \\
\end{align*}
\]

\[
\begin{align*}
\; 3 \ R_1 \leftarrow R_1 \cdot S \\
\; 15 \times, 67 + \\
\end{align*}
\]

end for

\[
\begin{align*}
\; R \leftarrow R_0 \cdot R_1 \\
\; 15 \times, 67 + \\
\end{align*}
\]
Agenda

1. Context

2. Hardware accelerator for the Tate pairing over supersingular curves
   - Implementation Results in Hardware

3. Software accelerator for the Tate pairing over supersingular curves
   - Computing the non-reduced pairing
   - Final exponentiation
   - Implementation results

4. Optimal Ate Pairing over Barreto-Naehrig Curves
   - Barreto–Naehrig Curves
Final exponentiation

- Final exponentiation consists of raising $\hat{e}(P, Q)$ to the exponent,

$$M = \frac{2^{4m} - 1}{N} = (2^{2m} - 1) \cdot (2^m + 1 - \nu 2^{(m+1)/2}),$$

where $\nu = (-1)^b$ when $m \equiv 1, 7 \pmod{8}$ and $\nu = (-1)^{1-b}$ in all other cases.

- Highly sequential computation, Very heterogeneous
Final exponentiation

- Final exponentiation consists of raising \( \hat{e}(P, Q) \) to the exponent,

\[
M = \frac{2^{4m} - 1}{N} = (2^{2m} - 1) \cdot (2^m + 1 - \nu 2^{(m+1)/2}),
\]

where \( \nu = (-1)^b \) when \( m \equiv 1, 7 \pmod{8} \) and \( \nu = (-1)^{1-b} \) in all other cases.

- Highly sequential computation, Very heterogeneous

- We perform this operation according to a slightly optimized version:
  - Raising to the \((2^m + 1)\)-th power. Raising the outcome of Miller’s algorithm to the \((2^{2m} - 1)\)-th power produces an element \( U \in \mathbb{F}_{2^{4m}} \) of order \( 2^{2m} + 1 \). This property allows one to save a multiplication over \( \mathbb{F}_{2^{4m}} \) when raising \( U \) to the \((2^m + 1)\)-th power.
Final exponentiation

- Final exponentiation consists of raising $\hat{e}(P, Q)$ to the exponent,

$$M = \frac{2^{4m} - 1}{N} = (2^{2m} - 1) \cdot (2^m + 1 - \nu 2^{(m+1)/2}),$$

where $\nu = (-1)^b$ when $m \equiv 1, 7 \pmod{8}$ and $\nu = (-1)^{1-b}$ in all other cases.

- Highly sequential computation, Very heterogeneous

We perform this operation according to a slightly optimized version:

- **Raising to the $(2^m + 1)$-th power.** Raising the outcome of Miller’s algorithm to the $(2^{2m} - 1)$-th power produces an element $U \in \mathbb{F}_{2^m}$ of order $2^{2m} + 1$. This property allows one to save a multiplication over $\mathbb{F}_{2^m}$ when raising $U$ to the $(2^m + 1)$-th power.

- **Raising to the $2^{\frac{m+1}{2}}$-th power.** Raising an element of $\mathbb{F}_{2^m}$ to the $2^i$-th power involves $4i$ squarings and at most four additions over $\mathbb{F}_{2^m}$.
Final exponentiation

- Final exponentiation consists of raising \( \hat{\epsilon}(P, Q) \) to the exponent,

\[
M = \frac{2^{4m} - 1}{N} = (2^{2m} - 1) \cdot (2^m + 1 - \nu 2^{(m+1)/2}),
\]

where \( \nu = (-1)^b \) when \( m \equiv 1, 7 \pmod{8} \) and \( \nu = (-1)^{1-b} \) in all other cases.

- Highly sequential computation, Very heterogeneous

- We perform this operation according to a slightly optimized version:
  - **Raising to the \((2^m + 1)\)-th power.** Raising the outcome of Miller’s algorithm to the \((2^{2m} - 1)\)-th power produces an element \( U \in \mathbb{F}_{2^{4m}} \) of order \( 2^{2m} + 1 \). This property allows one to save a multiplication over \( \mathbb{F}_{2^{4m}} \) when raising \( U \) to the \((2^m + 1)\)-th power.
  - **Raising to the \(2^\frac{m+1}{2}\)-th power.** Raising an element of \( \mathbb{F}_{2^{4m}} \) to the \(2^i\)-th power involves \( 4i \) squarings and at most four additions over \( \mathbb{F}_{2^m} \)
Finite field arithmetic

- Target: multi-core architectures
Finite field arithmetic

- Target: multi-core architectures
- Arithmetic over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$: SSE instruction set
Finite field arithmetic

- Target: multi-core architectures
- Arithmetic over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$: SSE instruction set
- Timings are given in clock cycles and were measured on an Intel Core 2 processor working at 2.4 GHz.

<table>
<thead>
<tr>
<th>Field</th>
<th>$x^p$</th>
<th>$\sqrt[p]{x}$</th>
<th>Mult</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aranha et al. CT-RSA’10</strong></td>
<td>$\mathbb{F}_{2^{1223}}$</td>
<td>160</td>
<td>166</td>
</tr>
<tr>
<td><strong>Our work CANS’10</strong></td>
<td>$\mathbb{F}_{2^{1223}}$</td>
<td>480</td>
<td>749</td>
</tr>
<tr>
<td></td>
<td>$\mathbb{F}_{3^{509}}$</td>
<td>900</td>
<td>974</td>
</tr>
</tbody>
</table>
Agenda

1. Context

2. Hardware accelerator for the Tate pairing over supersingular curves
   - Implementation Results in Hardware

3. Software accelerator for the Tate pairing over supersingular curves
   - Computing the non-reduced pairing
   - Final exponentiation
   - Implementation results

4. Optimal Ate Pairing over Barreto-Naehrig Curves
   - Barreto–Naehrig Curves
Implementation results

- Timings achieved on an Intel Core2 are given in millions of clock cycles
- Windows XP 64-bit SP2 environment

<table>
<thead>
<tr>
<th>Curve</th>
<th>Security</th>
<th># of cores</th>
<th>Freq. [GHz]</th>
<th>Calc. time [Mcycles]</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(\text{F}^2_{1223})</td>
<td>128</td>
<td>1</td>
<td>2.4</td>
<td>18.76</td>
</tr>
<tr>
<td>Aranha et al.</td>
<td>E(\text{F}^2_{1223})</td>
<td>128</td>
<td>2</td>
<td>2.4</td>
</tr>
<tr>
<td>CT-RSA'10</td>
<td>E(\text{F}^2_{1223})</td>
<td>128</td>
<td>4</td>
<td>2.4</td>
</tr>
<tr>
<td>Our work</td>
<td>E(\text{F}^3_{509})</td>
<td>128</td>
<td>1</td>
<td>2.4</td>
</tr>
<tr>
<td>CANS'10</td>
<td>E(\text{F}^3_{509})</td>
<td>128</td>
<td>2</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td>E(\text{F}^3_{509})</td>
<td>128</td>
<td>4</td>
<td>2.4</td>
</tr>
</tbody>
</table>
Timings achieved on an **Intel Core2** are given in millions of clock cycles

**Windows XP 64-bit SP2 environment**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aranha et al.</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CT-RSA’10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E(F_{2^{1223}})$</td>
<td>128</td>
<td>1</td>
<td>2.4</td>
<td>18.76</td>
</tr>
<tr>
<td>$E(F_{2^{1223}})$</td>
<td>128</td>
<td>2</td>
<td>2.4</td>
<td>10.08</td>
</tr>
<tr>
<td>$E(F_{2^{1223}})$</td>
<td>128</td>
<td>4</td>
<td>2.4</td>
<td>5.72</td>
</tr>
<tr>
<td><strong>Our work</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANS’10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E(F_{3^{509}})$</td>
<td>128</td>
<td>1</td>
<td>2.4</td>
<td>18.2</td>
</tr>
<tr>
<td>$E(F_{3^{509}})$</td>
<td>128</td>
<td>2</td>
<td>2.4</td>
<td>10.34</td>
</tr>
<tr>
<td>$E(F_{3^{509}})$</td>
<td>128</td>
<td>4</td>
<td>2.4</td>
<td>7.06</td>
</tr>
</tbody>
</table>
Software implementation notes: The supersingular case

- Significantly faster implementation (for a while)
Software implementation notes: The supersingular case

- Significantly faster implementation (for a while)
- How many cores?
Software implementation notes: The supersingular case

- Significantly faster implementation (for a while)
- How many cores?
  - acceleration always less than the ideal speedup factor
  - best choice: dual-core implementation
Software implementation notes: The supersingular case

- Significantly faster implementation (for a while)
- How many cores?
  - acceleration always less than the ideal speedup factor
  - best choice: dual-core implementation
- Characteristic 3 performs better than characteristic 2
Software implementation notes: The supersingular case

- Significantly faster implementation (for a while)
- How many cores?
  - acceleration always less than the ideal speedup factor
  - best choice: dual-core implementation
- Characteristic 3 performs better than characteristic 2
  - at least on Intel Core2 and Intel Core i7
Software implementation notes: The supersingular case

- Significantly faster implementation (for a while)
- How many cores?
  - acceleration always less than the ideal speedup factor
  - best choice: dual-core implementation
- Characteristic 3 performs better than characteristic 2
  - at least on Intel Core2 and Intel Core i7
  - next generation of processors: built-in carry-less 64-bit multiplier
  - the battle is not over!
Agenda

1. Context

2. Hardware accelerator for the Tate pairing over supersingular curves
   - Implementation Results in Hardware

3. Software accelerator for the Tate pairing over supersingular curves
   - Computing the non-reduced pairing
   - Final exponentiation
   - Implementation results

4. Optimal Ate Pairing over Barreto-Naehrig Curves
   - Barreto–Naehrig Curves
Barreto–Naehrig Curves

Defined by the equation $E : y^2 = x^3 + b$, where $b \neq 0$. Their embedding degree $k$ is equal to 12. The characteristic $p$ of the prime field, the group order $r$, and the trace of Frobenius $t_r$ of the curve are parametrized as follows:

$$
\begin{align*}
p(t) &= 36t^4 + 36t^3 + 24t^2 + 6t + 1, \\
r(t) &= 36t^4 + 36t^3 + 18t^2 + 6t + 1, \\
tr(t) &= 6t^2 + 1,
\end{align*}
$$

where $t \in \mathbb{Z}$ is an arbitrary integer such that $p = p(t)$ and $r = r(t)$ are both prime numbers.

For efficiency purposes, $t$ must have a low Hamming weight. In this work we used,

$$
t = 2^{62} - 2^{54} + 2^{44}
$$
Barreto–Naehrig Curves

Let $E[r]$ denote the $r$-torsion subgroup of $E$ and $\pi_p$ be the Frobenius endomorphism $\pi_p : E \to E$ given by $\pi_p(x, y) = (x^p, y^p)$. We define,

- $G_1 = E(\mathbb{F}_p)[r]$,
- $G_2 \subseteq E(\mathbb{F}_{p^{12}})[r]$,
- $G_\tau = \mu_r \subseteq \mathbb{F}_{p^{12}}^*$ (i.e. the group of $r$-th roots of unity).
- The optimal ate pairing on the BN curve $E$ is given as,

$$a_{opt} : G_2 \times G_1 \longrightarrow G_3$$

$$(Q, P) \longmapsto (f_{6t+2} Q(P) \cdot l_{[6t+2]} Q, \pi_p(Q)(P) \cdot l_{[6t+2]} Q + \pi_p(Q), -\pi_p^2(Q)(P))^\frac{p^{12}-1}{r}.$$ 

- In practice, pairing computations can be restricted to points $P$ and $Q'$ that belong to $E(\mathbb{F}_p)$ and $E'(\mathbb{F}_{p^2})$, respectively, where, $E'/\mathbb{F}_{p^2} : y^2 = x^3 + b/\xi$. 

Francisco Rodríguez-Henríquez
Faster Implementation of Pairings (34 / 49)
Optimal ate pairing algorithm

Input: $P \in \mathbb{G}_1$ y $Q \in \mathbb{G}_2$.
Output: $a_{\text{opt}}(Q, P)$.

1. Write $s = 6t + 2$ as $s = \sum_{i=0}^{L-1} s_i 2^i$, where $s_i \in \{-1, 0, 1\}$;
2. $T \leftarrow Q$, $f \leftarrow 1$;
3. for $i = L - 2$ to 0 do
   4. $f \leftarrow f^2 \cdot l_{T,T}(P)$; $T \leftarrow 2T$;
   5. if $s_i = -1$ then
      6. $f \leftarrow f \cdot l_{T,-Q}(P)$; $T \leftarrow T - Q$;
   7. else if $s_i = 1$ then
      8. $f \leftarrow f \cdot l_{T,Q}(P)$; $T \leftarrow T + Q$;
   9. end if
10. end for
11. $Q_1 \leftarrow \pi_p(Q)$; $Q_2 \leftarrow \pi_{p^2}(Q)$;
12. $f \leftarrow f \cdot l_{T,Q_1}(P)$; $T \leftarrow T + Q_1$;
13. $f \leftarrow f \cdot l_{T,-Q_2}(P)$; $T \leftarrow T - Q_2$;
14. $f \leftarrow f(p^{12} - 1)/r$;
15. return $f$;
Since \( p \mod 12 \equiv 1 \) we can build the towering up to the twelfth extension by adjoining irreducible binomial only.

\[
F_{p^{12}} = \frac{F_{p^6}[w]}{(w^2 - \gamma)}
\]

\[
F_{p^6} = \frac{F_{p^2}[v]}{(v^3 - \xi)}
\]

\[
F_{p^2} = \frac{F_p[u]}{(u^2 - \beta)}
\]
Since $p \mod 12 \equiv 1$ we can build the towering up to the twelfth extension by adjoining irreducible binomial only.
Since \( p \mod 12 \equiv 1 \) we can build the tower up to the twelfth extension by adjoining irreducible binomial only.

\[ \beta = -5 \]
Since \( p \mod 12 \equiv 1 \) we can build the towering up to the twelfth extension by adjoining irreducible binomial only.

\[
\begin{align*}
\xi &= u \\
\beta &= -5
\end{align*}
\]
Since $p \mod 12 \equiv 1$ we can build the tower up to the twelfth extension by adjoining irreducible binomial only.

\[
\begin{align*}
\beta &= -5 \\
\xi &= u \\
\gamma &= \nu
\end{align*}
\]
Since $p \mod 12 \equiv 1$ we can build the tower up to the twelfth extension by adjoining irreducible binomial only.

\[
\begin{align*}
F_{p^{12}} & = F_{p^6}[w]/(w^2 - \gamma) \\
F_{p^6} & = F_{p^2}[v]/(v^3 - \xi) \\
F_{p^2} & = F_p[u]/(u^2 - \beta)
\end{align*}
\]

\[
f = g + hw \in F_{p^{12}},
\]
with $g, h \in F_{p^6}$.

but also
\[
\begin{align*}
g & = g_0 + g_1 v + g_2 v^2, \\
h & = h_0 + h_1 v + h_2 v^2,
\end{align*}
\]
where $g_i, h_i \in F_{p^2}$, for $i = 1, 2, 3$.  

Since \( p \mod 12 \equiv 1 \) we can build the towering up to the twelfth extension by adjoining irreducible binomial only.

hence, we can write \( f \in \mathbb{F}_{p^{12}} \) as

\[
f = g + hw
= g + hw
= g_0 + h_0 W + g_1 W^2 + h_1 W^3 + g_2 W^4 + h_2 W^5.
\]
Let \((a, m, s, i), (\tilde{a}, \tilde{m}, \tilde{s}, \tilde{i}),\) and \((A, M, S, I)\) denote the cost of field addition, multiplication, squaring, and inversion in \(F_p, F_{p^2},\) and \(F_{p^6},\) respectively.

- We sometimes need to compute the multiplication in the base field by the constant coefficient \(\beta \in F_p\) of the irreducible binomial \(f(u) = u^2 - \beta.\) We refer to this operation as \(m_\beta\)

- We sometimes need to compute the multiplication of an arbitrary element in \(F_{p^2}\) times the constant \(\xi = u \in F_p\) at a cost of one multiplication by the constant \(\beta.\) We refer to this operation as \(m_\xi,\) but it is noticed that the cost of \(m_\xi\) is essentially the same of that of \(m_\beta.\)
Computational costs of the tower extension field arithmetic

<table>
<thead>
<tr>
<th>Field</th>
<th>Add/Sub</th>
<th>Mult</th>
<th>Squaring</th>
<th>Inversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{p^2}$</td>
<td>$\tilde{a} = 2a$</td>
<td>$\tilde{m} = 3m + 3a + m_\beta$</td>
<td>$\tilde{s} = 2m + 3a + m_\beta$</td>
<td>$\tilde{i} = 4m + m_\beta + 2a + i$</td>
</tr>
<tr>
<td>$F_{p^6}$</td>
<td>$3\tilde{a}$</td>
<td>$6\tilde{m} + 2m_\xi + 15\tilde{a}$</td>
<td>$2\tilde{m} + 3\tilde{s} + 2m_\xi + 8\tilde{a}$</td>
<td>$9\tilde{m} + 3\tilde{s} + 4m_\xi + 4\tilde{a} + \tilde{i}$</td>
</tr>
<tr>
<td>$F_{p^{12}}$</td>
<td>$6\tilde{a}$</td>
<td>$18\tilde{m} + 6m_\xi + 60\tilde{a}$</td>
<td>$12\tilde{m} + 4m_\xi + 45\tilde{a}$</td>
<td>$25\tilde{m} + 9\tilde{s} + 12m_\xi + 61\tilde{a} + \tilde{i}$</td>
</tr>
<tr>
<td>$G_{\Phi_6}(F_{p^2})$</td>
<td>$6\tilde{a}$</td>
<td>$18\tilde{m} + 6m_\xi + 60\tilde{a}$</td>
<td>$9\tilde{s} + 4m_\xi + 30\tilde{a}$</td>
<td>Conjugate</td>
</tr>
</tbody>
</table>
We took advantage of the following design decisions,

- The bit-length of $6t + 2$ is $L = 65$

This implies that we require 64 point doubling in the Miller loop.

The Hamming weight of $6t + 2$ is 7

This implies that we require 6 point addition/subtraction in the Miller loop.

The low Hamming weight of $t$ allows us to save arithmetic operations in the hard part of the final exponentiation.
We took advantage of the following design decisions,

- The bit-length of $6t + 2$ is $L = 65$
  - This implies that we require 64 point doubling in the Miller loop.

- The low Hamming weight of $t$ allows us to save arithmetic operations in the hard part of the final exponentiation.
We took advantage of the following design decisions,

- The bit-length of $6t + 2$ is $L = 65$
  - This implies that we require 64 point doubling in the Miller loop.
- The Hamming weight of $6t + 2$ is 7
We took advantage of the following design decisions,

- The bit-length of $6t + 2$ is $L = 65$
  - This implies that we require 64 point doubling in the Miller loop.

- The Hamming weight of $6t + 2$ is 7
  - This implies that we require 6 point addition/subtraction in the Miller loop.
We took advantage of the following design decisions,

- The bit-length of $6t + 2$ is $L = 65$
  - This implies that we require 64 point doubling in the Miller loop.
- The Hamming weight of $6t + 2$ is 7
  - This implies that we require 6 point addition/subtraction in the Miller loop.
- The low Hamming weight of $t$ allows us to save arithmetic operations in the hard part of the final exponentiation.
Mille Loop Cost

\[
\text{Miller Loop} = 64 \cdot (28\tilde{m} + 8\tilde{s} + 100\tilde{a} + 4m + 6m_\beta) + \\
6 \cdot (20\tilde{m} + 7\tilde{s} + 64\tilde{a} + 4m + 2m_\beta) + \\
40\tilde{m} + 14\tilde{s} + 128\tilde{a} + 14m + 4m_\beta \\
= 1952\tilde{m} + 568\tilde{s} + 6912\tilde{a} + 294m + 400m_\beta.
\]
Calculating the final Exponentiation

We must compute $f \in \mathbb{F}_{p^{12}}$ raised to the power $e = (p^{12} - 1)/r$. 

Raising to $f^{(p^6 - 1)}$ costs one conjugation, one inversion and one multiplication over $\mathbb{F}_{p^{12}}$. After this step, $f$ becomes an element of the cyclotomic group $G_{\Phi_6}(\mathbb{F}_p^2)$. Raising to the power $p^2 + 1$ costs 5 multiplications over $\mathbb{F}_{p^{12}}$, and one multiplication over $\mathbb{F}_{p^{12}}$. Raising to the power $m^{(p^4 - p^2 + 1)}/r$ is referred as the hard part of the final exponentiation.
Calculating the final Exponentiation

We must compute \( f \in \mathbb{F}_{p^{12}} \) raised to the power \( e = (p^{12} - 1)/r \)

\[
e = \frac{p^{12} - 1}{r} = (p^6 - 1) \cdot (p^2 + 1) \cdot \frac{p^4 - p^2 + 1}{r}.
\]
Calculating the final Exponentiation

We must compute \( f \in \mathbb{F}_{p^{12}} \) raised to the power \( e = (p^{12} - 1)/r \)

\[
e = \frac{p^{12} - 1}{r} = (p^6 - 1) \cdot (p^2 + 1) \cdot \frac{p^4 - p^2 + 1}{r}.
\]

- Raising to \( f(p^6-1) = \bar{f} \cdot f^{-1} \) costs one conjugation, one inversion and one multiplication over \( \mathbb{F}_{p^{12}} \).
- After this step, \( f \) becomes an element of the cyclotomic group \( \mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2}) \).
Calculating the final Exponentiation

We must compute $f \in \mathbb{F}_{p^{12}}$ raised to the power $e = (p^{12} - 1)/r$

$$e = \frac{p^{12} - 1}{r} = (p^6 - 1) \cdot (p^2 + 1) \cdot \frac{p^4 - p^2 + 1}{r}.$$

- Raising to $f^{(p^6-1)} = \bar{f} \cdot f^{-1}$ costs one conjugation, one inversion and one multiplication over $\mathbb{F}_{p^{12}}$.
- After this step, $f$ becomes an element of the cyclotomic group $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$.
- Raising to the power $p^2 + 1$ costs 5 multiplications over $\mathbb{F}_p$, and one multiplication over $\mathbb{F}_{p^{12}}$. 
Calculating the final Exponentiation

We must compute \( f \in \mathbb{F}_{p^{12}} \) raised to the power \( e = (p^{12} - 1)/r \)

\[
e = \frac{p^{12} - 1}{r} = (p^6 - 1) \cdot (p^2 + 1) \cdot \frac{p^4 - p^2 + 1}{r}.
\]

- Raising to \( f^{(p^6-1)} = \bar{f} \cdot f^{-1} \) costs one conjugation, one inversion and one multiplication over \( \mathbb{F}_{p^{12}} \).
- After this step, \( f \) becomes an element of the cyclotomic group \( \mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2}) \).
- Raising to the power \( p^2 + 1 \) costs 5 multiplications over \( \mathbb{F}_p \), and one multiplication over \( \mathbb{F}_{p^{12}} \).
- Raising to the power \( m(p^4-p^2+1)/r \) is referred as the hard part of the final exponentiation.
Hard part of the final exponentiation

We used the addition chain proposed by Scott et al. at Pairing’09

\[ m^t, m^{t^2}, m^{t^3}, m^p, m^{p^2}, m^{p^3}, m^{(tp)}, m^{(t^2p)}, m^{(t^3p)}, m^{(t^2p^2)}, \]
Hard part of the final exponentiation

We used the addition chain proposed by Scott et al. at Pairing’09

\[ m^t, m^{t^2}, m^{t^3}, m^p, m^{p^2}, m^{p^3}, m^{(tp)}, m^{(t^2p)}, m^{(t^3p)}, m^{(t^2p^2)}, \]

- Taking advantage of the Frobenius, we can easily compute, \( m^p, m^{p^2}, m^{p^3}, m^{(tp)}, m^{(t^2p)}, m^{(t^3p)}, m^{(t^2p^2)} \) at a cost of 35 multiplications in the base field \( \mathbb{F}_p \).
Hard part of the final exponentiation

We used the addition chain proposed by Scott et al. at Pairing’09

\[ m^t, \ m^{t^2}, \ m^{t^3}, \ m^p, \ m^{p^2}, \ m^{p^3}, \ m^{(tp)}, \ m^{(t^2p)}, \ m^{(t^3p)}, \ m^{(t^2p^2)}, \]

- Taking advantage of the Frobenius, we can easily compute, \( m^p, \ m^{p^2}, \ m^{p^3}, \ m^{(tp)}, \ m^{(t^2p)}, \ m^{(t^3p)}, \ y \ m^{(t^2p^2)} \) at a cost of 35 multiplications in the base field \( \mathbb{F}_p \).
- The most costly part of this procedure consists on the computation of \( m^t, \ m^{t^2} = (m^t)^t, \ m^{t^3} = (m^{t^2})^t. \)
- Since \( t = 2^{62} - 2^{54} + 2^{44} \), these exponentiations can be computed at a cost of \( 62 \cdot 3 = 186 \) cyclotomic squarings plus \( 2 \cdot 3 = 6 \) multiplications over \( \mathbb{F}_{p^{12}} \).
Final exponentiation computational cost

\[
\text{Exp. Final} = (25\tilde{m} + 9\tilde{s} + 12m_\beta + 61\tilde{a} + \tilde{i}) + (18\tilde{m} + 6m_\beta + 60\tilde{a}) +
\]
\[
(18\tilde{m} + 6m_\beta + 60\tilde{a}) + 10m +
\]
\[
13 \cdot (18\tilde{m} + 6m_\beta + 60\tilde{a}) + 4 \cdot (9\tilde{s} + 4m_\beta + 30\tilde{a}) + 70m +
\]
\[
186 \cdot (9\tilde{s} + 4m_\beta + 30\tilde{a}) + 6 \cdot (18\tilde{m} + 6m_\beta + 60\tilde{a})
\]
\[
= 403\tilde{m} + 1719\tilde{s} + 7021\tilde{a} + 80m + 898m_\beta + \tilde{i}.
\]
A Comparison of arithmetic operations required by the computation of the ate pairing variants.

<table>
<thead>
<tr>
<th></th>
<th>Miller Loop</th>
<th>˜m</th>
<th>˜s</th>
<th>˜a</th>
<th>˜i</th>
<th>˜mξ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hankerson et al.</strong></td>
<td></td>
<td>2277</td>
<td>356</td>
<td>6712</td>
<td>1</td>
<td>412</td>
</tr>
<tr>
<td>R-ate pairing</td>
<td></td>
<td>1616</td>
<td>1197</td>
<td>8977</td>
<td>1</td>
<td>1062</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td>3893</td>
<td>1553</td>
<td>15689</td>
<td>2</td>
<td>1474</td>
</tr>
<tr>
<td><strong>Naehrig et al.</strong></td>
<td></td>
<td>2022</td>
<td>590</td>
<td>7140</td>
<td>1</td>
<td>410</td>
</tr>
<tr>
<td>Optimal ate pairing</td>
<td></td>
<td>678</td>
<td>1719</td>
<td>7921</td>
<td>1</td>
<td>988</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td>2700</td>
<td>2309</td>
<td>15061</td>
<td>1</td>
<td>1398</td>
</tr>
<tr>
<td><strong>This work</strong></td>
<td></td>
<td>1954</td>
<td>568</td>
<td>6912</td>
<td>1</td>
<td>400</td>
</tr>
<tr>
<td>Optimal ate pairing</td>
<td></td>
<td>443</td>
<td>1719</td>
<td>7021</td>
<td>1</td>
<td>898</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td>2397</td>
<td>2287</td>
<td>13933</td>
<td>1</td>
<td>1298</td>
</tr>
</tbody>
</table>
Library Implementation

- We use the mul operation included in the x86-64 instruction set. It multiplies two 64-bit unsigned integers in about 3 clock cycles on Intel Core i7 and AMD Opteron processors.
- An element $x \in \mathbb{F}_p$ is represented as $x = (x_3, x_2, x_1, x_0)$, where $x_i, 0 \leq i \leq 3$, are 64-bit integers.
- Multiplication and inversion over $\mathbb{F}_p$ are accomplished according to the well-known Montgomery multiplication and Montgomery inversion algorithms, respectively.
- The 256-bit integer multiplication and Montgomery reduction are computed in 55 and 100 clock cycles, respectively.
Cycle counts of multiplication over $\mathbb{F}_{p^2}$, squaring over $\mathbb{F}_{p^2}$, and optimal ate pairing on different machines

<table>
<thead>
<tr>
<th></th>
<th>Core i7$^a$</th>
<th>Opteron$^b$</th>
<th>Core 2 Duo$^c$</th>
<th>Athlon 64 X2$^d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication over $\mathbb{F}_{p^2}$</td>
<td>435</td>
<td>443</td>
<td>558</td>
<td>473</td>
</tr>
<tr>
<td>Squaring over $\mathbb{F}_{p^2}$</td>
<td>342</td>
<td>355</td>
<td>445</td>
<td>376</td>
</tr>
<tr>
<td>Miller loop</td>
<td>1,330,000</td>
<td>1,360,000</td>
<td>1,680,000</td>
<td>1,480,000</td>
</tr>
<tr>
<td>Final exponentiation</td>
<td>1,000,000</td>
<td>1,040,000</td>
<td>1,270,000</td>
<td>1,150,000</td>
</tr>
<tr>
<td>Optimal ate pairing</td>
<td>2,330,000</td>
<td>2,400,000</td>
<td>2,950,000</td>
<td>2,630,000</td>
</tr>
</tbody>
</table>

$^a$ Intel Core i7 860 (2.8GHz), Windows 7, Visual Studio 2008 Professional
$^b$ Quad-Core AMD Opteron 2376 (2.3GHz), Linux 2.6.18, gcc 4.4.1
$^c$ Intel Core 2 Duo T7100 (1.8GHz), Windows 7, Visual Studio 2008 Professional
$^d$ Athlon 64 X2 Dual Core 6000+(3GHz), Linux 2.6.23, gcc 4.1.2
$^e$ Intel Core 2 Quad Q6600 (2394MHz), Linux 2.6.28, gcc 4.3.3
Comparison Table

A comparison of cycles and timings required by the computation of the ate pairing variants. The frequency is given in GHz and the timings are in milliseconds.

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Architecture</th>
<th>Cycles</th>
<th>Freq.</th>
<th>Calc. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aranha et al. [CT-RSA 2010]</td>
<td>$\eta T$ Intel Xeon 45nm (1 core)</td>
<td>17,400,000</td>
<td>2.0</td>
<td>8.70</td>
</tr>
<tr>
<td></td>
<td>$\eta T$ Intel Xeon 45nm (8 cores)</td>
<td>3,020,000</td>
<td>1.51</td>
<td></td>
</tr>
<tr>
<td>Beuchat et al. [CANS 2009]</td>
<td>$\eta T$ Intel Core i7 (1 core)</td>
<td>15,138,000</td>
<td>2.9</td>
<td>5.22</td>
</tr>
<tr>
<td></td>
<td>$\eta T$ Intel Core i7 (8 cores)</td>
<td>5,423,000</td>
<td></td>
<td>1.87</td>
</tr>
<tr>
<td>Hankerson et al.</td>
<td>R-ate Intel Core 2</td>
<td>10,000,000</td>
<td>2.4</td>
<td>4.10</td>
</tr>
<tr>
<td>Naehrig et al. eprint 2010/526, April.6.2010</td>
<td>$a_{opt}$ Intel Core 2 Quad Q6600</td>
<td>4,470,000</td>
<td>2.4</td>
<td>1.80</td>
</tr>
<tr>
<td>Fan et al. CHES'09</td>
<td>“R-ate” 130 nm ASIC</td>
<td>59,976</td>
<td>.204</td>
<td>2.91</td>
</tr>
<tr>
<td>This Work eprint 2010/526, jun.17.2010</td>
<td>$a_{opt}$ Intel Core i7</td>
<td>2,330,000</td>
<td>2.8</td>
<td>0.83</td>
</tr>
<tr>
<td>Aranha et al. eprint 2010/526, oct.19.2010</td>
<td>$a_{opt}$ Intel Core i7</td>
<td>1,703,000</td>
<td>2.8</td>
<td>0.608</td>
</tr>
</tbody>
</table>
Software Implementation notes on ordinary curves

- records do not last long in software!
Software Implementation notes on ordinary curves

- records do not last long in software!
- However, in hardware the performance records hold longer: Fan et al. CHES’09 and Beuchat et al. CHES’09 are still the fastest hardware accelerators for ordinary and supersingular curves, respectively.

- Future projects/open problems,
  ▶ To target higher security levels in software implementation of pairings (e.g., 192 bits of security)
  ▶ To design a hardware accelerator faster than any software library for asymmetric pairings over BN curves at 128-bit of security
to implement efficient pairing-based protocols in software and/or hardware
Software Implementation notes on ordinary curves

- records do not last long in software!

- However, in hardware the performance records hold longer: Fan et al. CHES’09 and Beuchat et al. CHES’09 are still the fastest hardware accelerators for ordinary and supersingular curves, respectively.

- Future projects/open problems,
  ▶ To target higher security levels in software implementation of pairings (e.g, 192 bits of security)
Software Implementation notes on ordinary curves

- records do not last long in software!
- However, in hardware the performance records hold longer: Fan et al. CHES’09 and Beuchat et al. CHES’09 are still the fastest hardware accelerators for ordinary and supersingular curves, respectively.
- Future projects/open problems,
  - To target higher security levels in software implementation of pairings (e.g., 192 bits of security)
  - To design a hardware accelerator faster than any software library for asymmetric pairings over BN curves at 128-bit of security
records do not last long in software!

However, in hardware the performance records hold longer: Fan et al. CHES’09 and Beuchat et al. CHES’09 are still the fastest hardware accelerators for ordinary and supersingular curves, respectively.

Future projects/open problems,

- To target higher security levels in software implementation of pairings (e.g., 192 bits of security)
- To design a hardware accelerator faster than any software library for asymmetric pairings over BN curves at 128-bit of security
- to implement efficient pairing-based protocols in software and/or hardware
Software Implementation notes on ordinary curves

- records do not last long in software!

- However, in hardware the performance records hold longer: Fan et al. CHES’09 and Beuchat et al. CHES’09 are still the fastest hardware accelerators for ordinary and supersingular curves, respectively.

- Future projects/open problems,
  - To target higher security levels in software implementation of pairings (e.g., 192 bits of security)
  - To design a hardware accelerator faster than any software library for asymmetric pairings over BN curves at 128-bit of security
  - to implement efficient pairing-based protocols in software and/or hardware
Thank you for your attention

Questions?