Avoiding Full Extension Field Arithmetic in Pairing Computations

Craig Costello

craig.costello@qut.edu.au
Queensland University of Technology

AfricaCrypt 2010

Joint work with Colin Boyd, Juanma Gonzalez-Nieto, Kenneth Koon-Ho Wong
Faster pairings mean more efficient...

- ID-based encryption (IBE)
- ID-based key agreement
- short signatures
- group signatures
- ring signatures
- certificateless encryption
- hierarchical encryption
- attribute-based encryption
- ...
# Table of contents

1. **Introduction**
   - Pairings and Miller’s algorithm
   - The evolution of Miller’s algorithm: state-of-the-art pairings

2. **Motivation**

3. **Miller $2^n$-tupling**

4. **Results**

5. **Related Work**
Pairings on ordinary elliptic curves over large prime fields

- Need two linearly independent points $R$ and $S$ of large prime order $r$ on $E(\mathbb{F}_p)$, i.e. need two subgroups of $E[r]$
- $E(\mathbb{F}_{p^k})$ is the smallest extension that contains two such subgroups (all $r+1$ subgroups in fact)
- $k$ is the embedding degree, first value such that $r|p^k - 1$
- Need a function $f_R$ with divisor $\text{div}(f_R) = r(R) - r(O)$

**Weil pairing methodology**

$$e(R, S) = f_R(S)/f_S(R) \in \mathbb{F}_{p^k}$$

**Tate pairing methodology**

$$e(R, S) = f_R(S)^{p^k-1} \in \mathbb{F}_{p^k}$$
What do the functions $f_R(S)$ and $f_S(R)$ look like?

- $\text{div}(f_R) = r(R) - r(O)$, i.e. a zero of order $r$ at $R$, and a pole of order $r$ at infinity ($O$).
- Indeterminate $f_R, f_S$ are of degree $r$ (at least in affine form)
- If $R \in E(\mathbb{F}_p)$ and $S \in E(\mathbb{F}_{p^k})$, then
  - $f_R(S)$ will have coefficients in $\mathbb{F}_p$, evaluated at elements in $\mathbb{F}_{p^k}$
  - $f_S(R)$ will have coefficients in $\mathbb{F}_{p^k}$, evaluated at elements in $\mathbb{F}_p$
- Too much to store $f_R$ explicitly before evaluating at $S$
- Therefore, evaluate at $S$ as you build the function and vice versa.
Miller’s algorithm

Input: $R$, $S$ and $r = (r_{\lfloor \log(r) \rfloor}, ..., r_0)_2$
Output: $f_R(S)$

- $f \leftarrow 1$, $T \leftarrow R$
- for $i$ from $\lfloor \log(r) \rfloor - 1$ to 0 do
  1. Compute $g = l/v$ in the chord-and-tangent doubling of $T$
  2. $T \leftarrow [2]T$
  3. $f \leftarrow f^2 \cdot g(S)$
  4. if $r_i = 1$ then
     i. Compute $g = l/v$ in the chord-and-tangent addition of $T + R$
     ii. $T \leftarrow T + R$
     iii. $f \leftarrow f \cdot g(S)$
  end if
end for: return $f$
Miller’s algorithm for the Weil pairing methodology

Initially: run twice to compute $e(R, S) = f_R(S)/f_S(R)$

Input: $R$, $S$ and $r = (r_{\lfloor \log(r) \rfloor}, \ldots, r_0)_2$

Output: $f_R(S)$ (first time) and $f_S(R)$ (second time)

- $f \leftarrow 1$, $T \leftarrow R$
- for $i$ from $\lfloor \log(r) \rfloor - 1$ to $0$ do
  1. Compute $g = l/v$ in the chord-and-tangent doubling of $T$
  2. $T \leftarrow [2] T$
  3. $f \leftarrow f^2 \cdot g(S)$
  4. if $r_i = 1$ then
     i. Compute $g = l/v$ in the chord-and-tangent addition of $T + R$
     ii. $T \leftarrow T + R$
     iii. $f \leftarrow f \cdot g(S)$
  end if
- end for: return $f$
Miller’s algorithm for the Tate pairing methodology

**Idea:** run once and exponentiate \( e(R, S) = f_R(S)^{p^k-1} \)

**Input:** \( R, S \) and \( r = (r_{\lfloor \log(r) \rfloor}, \ldots, r_0)_2 \)

**Output:** \( f_R(S) \)

- \( f \leftarrow 1, T \leftarrow R \)
- **for** \( i \) **from** \( \lfloor \log(r) \rfloor - 1 \) **to** 0 **do**
  - 1. Compute \( g = l/v \) in the chord-and-tangent doubling of \( T \)
  - 2. \( T \leftarrow [2] T \)
  - 3. \( f \leftarrow f^2 \cdot g(S) \)
  - 4. **if** \( r_i = 1 \) **then**
    - i. Compute \( g = l/v \) in the chord-and-tangent addition of \( T + R \)
    - ii. \( T \leftarrow T + R \)
    - iii. \( f \leftarrow f \cdot g(S) \)
  **end if**
- **end for:** return \( f \leftarrow f^{(p^k-1)} \)
### Miller’s algorithm with no inversions

**Ideas:** $\nu$’s are in subfields so discard $+$ projective coords

<table>
<thead>
<tr>
<th>Input: $R, S$ and $r = (r_{\lfloor \log(r) \rfloor}, \ldots, r_0)_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: $f_R(S)$</td>
</tr>
</tbody>
</table>

1. $f \leftarrow 1$, $T \leftarrow R$
2. **for** $i$ **from** $\lfloor \log(r) \rfloor - 1$ **to** 0 **do**
   1. Compute $g = l/\nu$ in the chord-and-tangent doubling of $T$
   2. $T \leftarrow [2]T$
   3. $f \leftarrow f^2 \cdot g(S)$
   4. **if** $r_i = 1$ **then**
      1. Compute $g = l/\nu$ in the chord-and-tangent addition of $T + R$
      2. $T \leftarrow T + R$
      3. $f \leftarrow f \cdot g(S)$
   **end if**
3. **end for:** return $f \leftarrow f(p^k - 1)$
Miller’s algorithm with optimal loop length

Idea: Minimize loop length + low Hamming-weight

Input: $R, S$ and $m_{\text{opt}} = (m_{\lfloor \log(m_{\text{opt}}) \rfloor}, \ldots, m_0)_2$

Output: $f_R(S)$

- $f \leftarrow 1, T \leftarrow R$
- for $i$ from $\lfloor \log(m_{\text{opt}}) \rfloor - 1$ to 0 do
  1. Compute $g = l$ in the chord-and-tangent doubling of $T$
  2. $T \leftarrow [2] T$
  3. $f \leftarrow f^2 \cdot g(S)$
  4. if $r_i = 1$ then
     i. Compute $g = l$ in the chord-and-tangent addition of $T + R$
     ii. $T \leftarrow T + R$
     iii. $f \leftarrow f \cdot g(S)$
  end if

end for: return $f \leftarrow f^{p^k-1}$
The state-of-the-art

Input: \( R, S \) and \( m_{\text{opt}} = (m_{\lfloor \log(m_{\text{opt}}) \rfloor}, \ldots, m_0)_2 \)

Output: \( f_R(S) \)

- \( f \leftarrow 1, \quad T \leftarrow R \)
- \( \text{for } i \text{ from } \lfloor \log(m_{\text{opt}}) \rfloor - 1 \text{ to } 0 \text{ do} \)
  - Compute \( g = l \) in the chord-and-tangent doubling of \( T \)
  - \( T \leftarrow [2] T \)
  - \( f \leftarrow f^2 \cdot g(S) \)
- \( \text{end for: } \quad \text{return } f \leftarrow f^{(p^k - 1)} \)
Tate vs. ate groups

- $G_1 = E[r] \cap \ker(\pi_p - [1])$ and $G_2 = E[r] \cap \ker(\pi_p - [p])$, i.e. $G_1 \in E(\mathbb{F}_p)$ (base field) and $G_2 \in E(\mathbb{F}_{p^k})$ (full ext. field)
- Use twisted curve $E' \cong E$ to define $G'_2 \cong G_2$ but $G'_2 \in E(\mathbb{F}_{p^k/d})$ (twisted subfield)

<table>
<thead>
<tr>
<th>Tate-like pairings</th>
<th>Ate-like pairings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st argument: $R \in G_1$</td>
<td>1st argument: $R \in G'_2$</td>
</tr>
<tr>
<td>2nd argument $S \in G'_2$</td>
<td>2nd argument $S \in G_1$</td>
</tr>
</tbody>
</table>
## What else can we do?

Red stuff: Optimized or exhausted or given enough attention

| Input: $R$, $S$ and $m_{\text{opt}} = (m_{\lfloor \log(m_{\text{opt}}) \rfloor}, \ldots, m_0)_2$ |
| Output: $f_R(S)$ |

1. $f \leftarrow 1, T \leftarrow R$
2. **for** $i$ **from** $\lfloor \log(m_{\text{opt}}) \rfloor - 1$ **to** 0 **do**
   1. Compute $g = l$ in the chord-and-tangent doubling of $T$
   2. $T \leftarrow [2] T$
   3. $f \leftarrow f^2 \cdot g(S)$
3. **end for**
4. **return** $f \leftarrow f(p^k - 1)$
A closer look at the Miller update step

Complexity of operations

i. $f \leftarrow f^2$

ii. Evaluate $g$ at $S$

iii. $f \leftarrow f \cdot g$

i. $f$ is a general element of $\mathbb{F}_{p^k}$ (can't do much here)

ii. Indeterminate $g$ takes form $g(x, y) = g_x \cdot x + g_y \cdot y + g_0$, and is evaluated as $g(S_x, S_y)$
   - **ate:** $g_x, g_y, g_0 \in \mathbb{F}_{p^k/d}$ and $S_x, S_y \in \mathbb{F}_p$
   - **Tate:** $g_x, g_y, g_0 \in \mathbb{F}_p$ and $S_x, S_y \in \mathbb{F}_{p^k/d}$

iii. **KEY:** If degree of twist $d = 4$ or $d = 6$, then $g(S)$ is not a general element of $\mathbb{F}_{p^k/d}$ (i.e. $f \cdot g$ is not a full extension field multiplication!)
The multiplication $f \cdot g$

- An example of $f \cdot g$ (sextic twist)
  
  $$f = (f_{2,1} \cdot \alpha + f_{2,0}) \cdot \beta^2 + (f_{1,1} \cdot \alpha + f_{1,0}) \cdot \beta + (f_{0,1} \cdot \alpha + f_{0,0}) \in \mathbb{F}_{p^k},$$
  
  $$g(S_x, S_y) = (g_x \hat{S}_x) \cdot \beta + (g_y \hat{S}_y) \cdot \alpha + g_0 \in \mathbb{F}_{p^k},$$

  where the $f_{i,j}$'s and both $g_x \hat{S}_x$ and $g_y \hat{S}_y$ are contained in $\mathbb{F}_{p^e}$.

- NOT a full extension field multiplication!

- Repetitively multiplying full elements (the $f$'s) by sparse elements (the $g$'s) is potentially bad, because
  
  - We’re not making full use of finite field optimizations (Karatsuba, Toom-Cook multiplication etc)
  
  - We’re “touching” the full extension field element before we need to

- ... what can we do instead?
Keeping the $f$'s and $g$'s separate

\[
\text{for } i = \lceil \log_2(m) \rceil - 1 \text{ to } 0 \text{ do }
\]

- Compute $g = l$ in the chord-and-tangent doubling of $T$
- $T \leftarrow [2] T$
- $f \leftarrow f^2 \cdot g(S)$

end for

- What happens if we keep the $f$'s and $g$'s separate for $n$ iterations in a row?
- $T$ would be doubled $n$ times
- The $f$ would be squared $n$ times in a row
- The $n$ consecutive $g$'s would no longer be absorbed into $f$
Combining \( n \) iterations: Miller \( 2^n \)-tupling

\[
\begin{align*}
\text{for } i &= \lfloor \log_2(n) \rfloor - 1 \text{ to } 0 \text{ do} \\
&\quad \text{Compute } g_{\text{prod}} = g_1^{2^n-1} g_2^{2^n-2} \ldots g_{n-1}^{2^1} g_n \text{ in the } 2^n\text{-tupling of } T \\
&\quad T \leftarrow [2^n] T \\
&\quad f \leftarrow f^{2^n} \cdot g_{\text{prod}}(S) \\
\text{end for}
\end{align*}
\]

- **Green comps:** was \( ns_k + n\tilde{m}_k \rightarrow \) now \( ns_k + m_k \)
- **Red comps:** Used to be \( n \) degree 1 functions, now is one (much more complicated) \( 2^n \)-degree function
- **How can we win?** if the extra computations incurred computing \( g_{\text{prod}} \) are redeemed by the saving of \((n - 1)m_k\).
- Will win if \( \mathbb{F}_{p^k} \) is much bigger than \( \mathbb{F}_p \) (Tate) or \( \mathbb{F}_{p^{k/d}} \) (ate)
How to get $g_{\text{prod}}$

Compute $g_{\text{prod}} = g_1^{2^n-1} g_2^{2^n-2} \ldots g_{n-1}^{2^1} g_n$ in the $2^n$-tupling of $T$

$T \leftarrow [2^n] T$

- $T_n = [2] T_{n-1} = ... = [2^{n-1}] T$
- Degrees of formulas for $T_n$ and $g_n$ in terms of $T = (x_1, y_1)$ grow exponentially in $n$
- Paper explores $n = 2$ (quadrupling) and $n = 3$ (octupling)
- Paper explores two curve shapes
  - $y^2 = x^3 + b$  \hspace{1cm} $d = 2, 6$ twists \hspace{1cm} Homogeneous projective
  - $y^2 = x^3 + ax$  \hspace{1cm} $d = 2, 4$ twists \hspace{1cm} Weight-$(1, 2)$
- Formulas are reduced using Gröbner basis reduction
An example: Quadrupling on \( y^2 = x^3 + b \)

\[
g_{\text{prod}} = \prod_{i=1}^{2} (g[2i-1] T, [2i-1] T)^{2^2 - i} = (g T, T)^{2} \cdot (g[2] T, [2] T),
\]

\[
g^{\ast} = \alpha \cdot (L_{1,0} \cdot x_S + L_{2,0} \cdot x_S^2 + L_{0,1} \cdot y_S + L_{1,1} \cdot x_S y_S + L_{0,0}),
\]

First

\[
L_{2,0} = -6X_1^2 Z_1 (5Y_1^4 + 54b Y_1^2 Z_1^2 - 27b^2 Z_1^4),
\]

\[
L_{0,1} = 8X_1 Y_1 Z_1 (5Y_1^4 + 27b^2 Z_1^4),
\]

\[
L_{1,1} = 8Y_1 Z_1^2 (Y_1^4 + 18b Y_1^2 Z_1^2 - 27b^2 Z_1^4),
\]

Argument

\[
L_{0,0} = 2X_1 (Y_1^6 - 75b Y_1^4 Z_1^2 + 27b^2 Y_1^2 Z_1^4 - 81b^3 Z_1^6),
\]

Computations

\[
L_{1,0} = -4Z_1 (5Y_1^6 - 75b Z_1^2 Y_1^4 + 135Y_1^2 b^2 Z_1^4 - 81b^3 Z_1^6).
\]

\[
X_{D_1} = 4X_1 Y_1 (Y_1^2 - 9b Z_1^2), \quad Y_{D_1} = 2Y_1^4 + 36b Y_1^2 Z_1^2 - 54b^2 Z_1^4, \quad Z_{D_1} = 16Y_1^3 Z_1
\]

\[
(X_{D_2} : Y_{D_2} : Z_{D_2}) = [2](X_{D_1} : Y_{D_1} : Z_{D_1})
\]
Quadrupling on $y^2 = x^3 + b$ cont.

\[
A = Y_1^2, \quad B = Z_1^2, \quad C = A^2, \quad D = B^2, \quad E = (Y_1 + Z_1)^2 - A - B, \quad F = E^2, \quad G = X_1^2, \quad H = (X_1 + Y_1)^2 - A - G, \\
l = (X_1 + E)^2 - F - G, \quad J = (A + E)^2 - C - F, \quad K = (Y_1 + B)^2 - A - D, \quad L = 27b^2D, \quad M = 9bF, \quad N = A \cdot C, \\
R = A \cdot L, \quad S = bB, \quad T = S \cdot L, \quad U = S \cdot C, \quad X_{D_1} = 2H \cdot (A - 9S), \quad Y_{D_1} = 2C + M - 2L, \quad Z_{D_1} = 4J, \\
L_{1,0} = -4Z_1 \cdot (5N + 5R - 3T - 75U), \quad L_{2,0} = -3G \cdot Z_1 \cdot (10C + 3M - 2L), \quad L_{0,1} = 2I \cdot (5C + L), \\
L_{1,1} = 2K \cdot Y_{D1}, \quad L_{0,0} = 2X_1 \cdot (N + R - 3T - 75U). \\
F^* = L_{1,0} \cdot x_5 + L_{2,0} \cdot x_5^2 + L_{0,1} \cdot y_5 + L_{1,1} \cdot x_5 y_5 + L_{0,0}, \quad A_2 = Y_{D_1}^2, \quad B_2 = Z_{D_1}^2, \quad C_2 = 3bB_2, \\
D_2 = 2X_{D_1} \cdot Y_{D1}, \quad E_2 = (Y_{D_1} + Z_{D_1})^2 - A_2 - B_2, \quad F_2 = 3C_2, \quad X_{D_2} = D_2 \cdot (A_2 - F_2), \\
Y_{D_2} = (A_2 + F_2)^2 - 12C_2, \quad Z_{D2} = 4A_2 \cdot E_2.
\]

The above sequence of operations costs $14m + 16s + 4em_1$. 
Addition in Miller $2^n$-tupling

- We are now writing the loop parameter in base $2^n$.
- Instead of $T \leftarrow T + R$ in standard routine, we must now account for $T \leftarrow T + [w]R$, where $w < 2^n$.
- Precompute and store the (small number of) values $[w]R$ in the $2^n$-ary expansion of $m$.
- Must now multiply Miller function with addition update $f^+$, where $\text{div}(f^+) = w(R) + ([v]R) - ([v]R + [w]R) - w(O)$
  - $f^+ = \prod_{i=0}^{w-1} g_{[v]R+[i]R,R}$  \hfill ...BAD
  - $f^+ = f_{w,R} \cdot g_{[v]R,[w]R}$  \hfill ...GOOD
- Since $[w]R$ is precomputed, and $f_{w,R}$ can also be precomputed, this is at most two multiplications.
- ... also possible that less addition steps occur in $2^n$-ary implementation.
Algorithm summary: a typical iteration

- Compute function $g_{\text{prod}}$ in the $2^n$-tupling of $T$
- $T \leftarrow [2^n]T$
- $f \leftarrow f^{2n} \cdot g_{\text{prod}}$
- if $m_i \neq 0$ then
  - Compute function $f^+ = f_{w,R} \cdot g_T, [m_i]R$
  - $T \leftarrow T + [m_i]R$
  - $f \leftarrow f \cdot f^+$
- end if
j(E) = 0: Curves of the form $y^2 = x^3 + b$

j(E) = 1728: Curves of the form $y^2 = x^3 + ax$

<table>
<thead>
<tr>
<th>j(E)</th>
<th>Doubling: $n = 1$ (6 loops)</th>
<th>Quadrupling: $n = 2$ (3 loops)</th>
<th>Octupling: $n = 3$ (2 loops)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$12m + 42s + 12em_1$</td>
<td>$42m + 48s + 12em_1$</td>
<td>$80m + 64s + 16em_1$</td>
</tr>
<tr>
<td></td>
<td>$+6M + 6S$</td>
<td>$+3M + 6S$</td>
<td>$+2M + 6S$</td>
</tr>
<tr>
<td>1728</td>
<td>$12m + 48s + 12em_1$</td>
<td>$33m + 60s + 12em_1$</td>
<td>$64m + 114s + 16em_1$</td>
</tr>
<tr>
<td></td>
<td>$+6M + 6S$</td>
<td>$+3M + 6S$</td>
<td>$+2M + 6S$</td>
</tr>
</tbody>
</table>

Table: Operation counts for the equivalent number of iterations of $2^n$-tuple and add for $n = 1, 2, 3$. 
### Results cont...

<table>
<thead>
<tr>
<th>$k$</th>
<th>$j(E)$</th>
<th>Pairings on $G_1 \times G_2$</th>
<th>Pairings on $G_2 \times G_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(Tate, twisted ate)</td>
<td>(ate, R-ate)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$n = 1$ (6 loops)</td>
<td>$n = 1$ (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$n = 2$ (3 loops)</td>
<td>$n = 2$ (3 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$n = 3$ (2 loops)</td>
<td>$n = 3$ (2 loops)</td>
</tr>
<tr>
<td>4</td>
<td>1728</td>
<td>159.6 (6 loops)</td>
<td>159.6 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>163.2 (3 loops)</td>
<td>163.2 (3 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>232.4 (2 loops)</td>
<td>232.4 (2 loops)</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>219.6 (6 loops)</td>
<td>219.6 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>209.4 (3 loops)</td>
<td>209.4 (3 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>249.2 (2 loops)</td>
<td>249.2 (2 loops)</td>
</tr>
<tr>
<td>8</td>
<td>1728</td>
<td>366 (6 loops)</td>
<td>466.8 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>315.6 (3 loops)</td>
<td>477.6 (3 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>370.8 (2 loops)</td>
<td>681.2 (2 loops)</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>555.6 (6 loops)</td>
<td>646.8 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>455.4 (3 loops)</td>
<td>616.2 (3 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>469.2 (2 loops)</td>
<td>731.6 (2 loops)</td>
</tr>
<tr>
<td>16</td>
<td>1728</td>
<td>973.2 (6 loops)</td>
<td>1376.4 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>760.8 (3 loops)</td>
<td>1408.8 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>770 (2 loops)</td>
<td>2011.6 (6 loops)</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>891.6 (6 loops)</td>
<td>1023 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>701.4 (3 loops)</td>
<td>1214 (6 loops)</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>1551.6 (6 loops)</td>
<td>1916.4 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1181.4 (3 loops)</td>
<td>1824.6 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1113.2 (2 loops)</td>
<td>2162.8 (6 loops)</td>
</tr>
<tr>
<td>32</td>
<td>1728</td>
<td>2770.8 (6 loops)</td>
<td>4081.2 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2072.4 (3 loops)</td>
<td>4178.4 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1935.6 (2 loops)</td>
<td>5970.8 (2 loops)</td>
</tr>
<tr>
<td>36</td>
<td>0</td>
<td>2547.6 (6 loops)</td>
<td>3186 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1907.4 (3 loops)</td>
<td>3033 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1757.6 (2 loops)</td>
<td>3594 (6 loops)</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>4515.6 (6 loops)</td>
<td>5701.2 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3335.4 (3 loops)</td>
<td>5425.8 (6 loops)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3013.2 (2 loops)</td>
<td>6424.4 (6 loops)</td>
</tr>
</tbody>
</table>

**Table:** Total base field operation count for the equivalent of 6 standard double-and-add loops.
Related Work

- **WAIFI2010 paper**
  - Higher integrability into existing pairing code
  - Only slightly slower than these techniques
  - No cumbersome explicit formulas

- **Other paper (to appear soon on ePrint archive)**
  - Many pairing-based protocols have one argument fixed (long term key etc)
  - A heap of precomputation can be done
  - Much faster implementations possible here
QUESTIONS?