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

Motivation

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

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 $\mathrm{div}(f_R) = r(R) - r(\mathcal{O})$

Weil pairing methodology

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

Tate pairing methodology

$$e(R,S) = f_R(S)^{p^k-1} \in \mathbb{F}_{p^k}$$

The pairing evaluation functions

What do the functions $f_R(S)$ and $f_S(R)$ look like?

- $\mathrm{div}(f_R) = r(R) - r(\mathcal{O})$, i.e. a zero of order r at R, and a pole of order r at infinity $(\mathcal{O})$.
- 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 $r = (r_{\lfloor \log(r) \rfloor}, \ldots, r_0)_2$
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

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: v's are in subfields so discard + projective coords

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** $f \leftarrow f^{(p^k-1)}$

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 $m_{\mathrm{opt}} = (m_{\lfloor \log(m_{\mathrm{opt}}) \rfloor}, ..., m_0)_2$
Output: $f_R(S)$

- $f \leftarrow 1$, $T \leftarrow R$
- **for** i **from** $\lfloor \log(m_{\mathrm{opt}}) \rfloor - 1$ **to** 0 **do**
  1. Compute $g = l$ in the chord-and-tangent doubling of T
  2. $T \leftarrow [2]T$
  3. $f \leftarrow f^2 \cdot g(S)$

end for: **return** $f \leftarrow f^{(p^k-1)}$

Tate vs. ate groups

- $\mathbb{G}_1 = E[r] \cap \ker(\pi_p - [1])$ and $\mathbb{G}_2 = E[r] \cap \ker(\pi_p - [p])$, i.e. $\mathbb{G}_1 \in E(\mathbb{F}_p)$ (base field) and $\mathbb{G}_2 \in E(\mathbb{F}_{p^k})$ (full ext. field)
- Use twisted curve $E' \cong E$ to define $\mathbb{G}_2' \cong \mathbb{G}_2$ but $\mathbb{G}_2' \in E(\mathbb{F}_{p^{k/d}})$ (twisted subfield)

Tate-like pairings

1st argument: $R \in \mathbb{G}_1$ 2nd argument $S \in \mathbb{G}_2'$

Ate-like pairings

1st argument: $R \in \mathbb{G}_2'$ 2nd argument $S \in \mathbb{G}_1$

What else can we do?

Red stuff: Optimized or exhausted or given enough attention

Input: R, S and $m_{\text{opt}} = (m_{\lfloor \log(m_{\text{opt}}) \rfloor}, ..., m_0)_2$
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. $f \leftarrow f^2 \cdot g(S)$

  end for
- **return** $f \leftarrow f^{(p^k-1)}$

A closer look at the Miller update step

Complexity of operations

i.	$f \leftarrow f^2$	$\mathbf{s}_k$
ii.	Evaluate g at S	$2k/d \cdot \mathbf{m}_1$
iii.	$f \leftarrow f \cdot g$	$\mathbf{m}_k$?

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

for $i = \lfloor \log_2(m) \rfloor - 1$ **to** 0 **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

for $i = \lfloor \log_{2^n}(m) \rfloor - 1$ **to** 0 **do**

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

$T \leftarrow [2^n]T$

$f \leftarrow f^{2^n} \cdot g_{\text{prod}}(S)$

end for

- Green comps: was $n\mathbf{s}_k + n\tilde{\mathbf{m}}_k \quad \rightarrow \quad$ now $\quad n\mathbf{s}_k + \mathbf{m}_k$
- 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_{prod} are redeemed by the saving of $(n-1)\mathbf{m}_k$.
- 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_{prod}

Compute $g_{\mathrm{prod}} = g_1^{2^{n-1}} g_2^{2^{n-2}} ... g_{n-1}^{2^1} g_n$ 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
  - $y^2 = x^3 + b$ $\quad d = 2, 6$ twists $\quad$ Homogeneous projective
  - $y^2 = x^3 + ax$ $\quad d = 2, 4$ twists $\quad$ Weight-$(1, 2)$
- Formulas are reduced using Gröbner basis reduction

An example: Quadrupling on $y^2 = x^3 + b$

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

$$g^* = \alpha \cdot (L_{1,0} \cdot x_S + L_{2,0} \cdot x_S^2 + L_{0,1} \cdot y_S + L_{1,1} \cdot x_S y_S + L_{0,0}),$$

$$\begin{aligned}
L_{2,0} &= -6X_1^2 Z_1 (5Y_1^4 + 54bY_1^2 Z_1^2 - 27b^2 Z_1^4),\\
L_{0,1} &= 8X_1 Y_1 Z_1 (5Y_1^4 + 27b^2 Z_1^4),\\
L_{1,1} &= 8Y_1 Z_1^2 (Y_1^4 + 18bY_1^2 Z_1^2 - 27b^2 Z_1^4),\\
L_{0,0} &= 2X_1 (Y_1^6 - 75bY_1^4 Z_1^2 + 27b^2 Y_1^2 Z_1^4 - 81b^3 Z_1^6),\\
L_{1,0} &= -4Z_1 (5Y_1^6 - 75bZ_1^2 Y_1^4 + 135Y_1^2 b^2 Z_1^4 - 81b^3 Z_1^6).
\end{aligned}$$

First Argument Computations

$$X_{D^1} = 4X_1 Y_1 (Y_1^2 - 9bZ_1^2), \quad Y_{D^1} = 2Y_1^4 + 36bY_1^2 Z_1^2 - 54b^2 Z_1^4, \quad Z_{D^1} = 16Y_1^3 Z_1$$

$$(X_{D^2} : Y_{D^2} : Z_{D^2}) = [2](X_{D^1} : Y_{D^1} : Z_{D^1})$$

Quadrupling on $y^2 = x^3 + b$ cont.

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

The above sequence of operations costs $14\mathbf{m} + 16\mathbf{s} + 4e\mathbf{m}_1$.

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 $\mathrm{div}(f^+) = w(R) + ([v]R) - ([v]R + [w]R) - w(\mathcal{O})$
  - $f^+ = \prod_{i=0}^{w-1} g_{[v]R+[i]R,R}$ ...BAD
  - $f^+ = f_{w,R} \cdot g_{[v]R,[w]R}$ ...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_{prod} in the 2^n-tupling of T
- $T \leftarrow [2^n]T$
- $f \leftarrow f^{2^n} \cdot g_{\text{prod}}$
- **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

Results

- $j(E) = 0$: Curves of the form $y^2 = x^3 + b$
- $j(E) = 1728$: Curves of the form $y^2 = x^3 + ax$

$j(E)$	Doubling: $n = 1$ (6 loops)	Quadrupling: $n = 2$ (3 loops)	Octupling: $n = 3$ (2 loops)
0	$12\mathbf{m} + 42\mathbf{s} + 12e\mathbf{m}_1 + 6\mathbf{M} + 6\mathbf{S}$	$42\mathbf{m} + 48\mathbf{s} + 12e\mathbf{m}_1 + 3\mathbf{M} + 6\mathbf{S}$	$80\mathbf{m} + 64\mathbf{s} + 16e\mathbf{m}_1 + 2\mathbf{M} + 6\mathbf{S}$
1728	$12\mathbf{m} + 48\mathbf{s} + 12e\mathbf{m}_1 + 6\mathbf{M} + 6\mathbf{S}$	$33\mathbf{m} + 60\mathbf{s} + 12e\mathbf{m}_1 + 3\mathbf{M} + 6\mathbf{S}$	$64\mathbf{m} + 114\mathbf{s} + 16e\mathbf{m}_1 + 2\mathbf{M} + 6\mathbf{S}$

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

Results cont...

		Pairings on $\mathbb{G}_1 \times \mathbb{G}_2$ (Tate, twisted ate)			Pairings on $\mathbb{G}_2 \times \mathbb{G}_1$ (ate, R-ate)		
k	$j(E)$	$n = 1$ (6 loops)	$n = 2$ (3 loops)	$n = 3$ (2 loops)	$n = 1$ (6 loops)	$n = 2$ (3 loops)	$n = 3$ (2 loops)
4	1728	159.6	163.2	232.4	159.6	163.2	232.4
6	0	219.6	209.4	249.2	219.6	209.4	249.2
8	1728	366	315.6	370.8	466.8	477.6	681.2
12	0	555.6	455.4	469.2	646.8	616.2	731.6
16	1728	973.2	760.8	770	1376.4	1408.8	2011.6
18	0	891.6	701.4	689.2	1074	1023	1214
24	0	1551.6	1181.4	1113.2	1916.4	1824.6	2162.8
32	1728	2770.8	2072.4	1935.6	4081.2	4178.4	5970.8
36	0	2547.6	1907.4	1757.6	3186	3033	3594
48	0	4515.6	3335.4	3013.2	5701.2	5425.8	6424.4

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?