

# Fast Implementation of Pairings on Elliptic Curves: Algorithms and Tricks

Mike Scott

Dublin City University

Workshop on Pairings, Essen, May 2009

## First steps...

- ▶ To do pairing-based crypto on elliptic curves we need two things...
- ▶ 1. Efficient Algorithms.
- ▶ 2. Suitable Elliptic Curves (See Freeman-S.-Teske Taxonomy (2006)).
- ▶ We have both!

# The Tate Pairing

- ▶ We start with an elliptic curve  $E$  defined over  $\mathbb{F}_q$  (where  $q = p^m$ ), where the number of points on the curve  $\#E = q + 1 - t$  (where  $t$  is the “trace”) has a large prime divisor  $r$ .
- ▶  $\rho = \lg(q) / \lg(r)$ .
- ▶ The curve is “special” in that  $k$  is smallest positive integer such that  $r | q^k - 1$ , and  $k \leq 50$ .
- ▶ Points  $P$  and  $Q$  are (linearly independent points) in  $E(\mathbb{F}_{q^k})$ , and  $P$  is of order  $r$ .
- ▶ Then the Tate pairing  $e(P, Q)$ , evaluates as element of order  $r$  in  $\mathbb{F}_{q^k}$ .

# Security

- ▶ To be useful as a cryptographic primitive...
- ▶ Group size  $r$  must be big enough to resist Pollard-Rho-like algorithms.
- ▶ Extension field  $\mathbb{F}_{q^k}$  must be big enough to resist index calculus algorithms.
- ▶ For example if  $q$  and  $r$  are 171 bits and  $k = 6$ ,  $q^k$  is approx. 1024 bits. About right.
- ▶ Need for “balanced” security! This more or less “fixes” the optimal  $k$  value.
- ▶ (although it is effected by the best  $\rho$  value available for a pairing-friendly curve at the required level of security)
- ▶ For AES-128,  $q$  and  $r$  should be ideally 256-bits, so need  $q^k$  to be 3072 bits, so  $k = 12$ .

## Where to do it?

- ▶ Three practical possibilities...
- ▶ Supersingular curves over  $\mathbb{F}_p$  ( $k = 2$ )
- ▶ Supersingular curves over  $\mathbb{F}_{2^m}$  or  $\mathbb{F}_{3^m}$  ( $k = 4$  and  $k = 6$  resp.)
- ▶ “Pairing-friendly” ordinary curves over  $\mathbb{F}_p$  (Unlimited  $k$ )
- ▶ Here I will concentrate on the case of ordinary curves.
- ▶ For SS curves over fields of small characteristic, different methods apply (See  $\eta_T$  pairing)

## Starting point – Miller's Algorithm

---

**Algorithm 1** Computation of  $e(P, Q)$  using basic Miller's algorithm

---

INPUT:  $P \in E(\mathbb{F}_{p^k})$ ,  $Q \in E(\mathbb{F}_{p^k})$ , where  $P$  has order  $r$

OUTPUT:  $e(P, Q)$

- 1:  $T \leftarrow P, f \leftarrow 1$
  - 2: **for**  $i \leftarrow \lfloor \lg(r) \rfloor - 1$  **downto** 0 **do**
  - 3:      $f \leftarrow f^2 \cdot I_{T,T}(Q) / v_{2T}(Q)$
  - 4:      $T \leftarrow 2T$
  - 5:     **if**  $r_i = 1$  **then**
  - 6:          $f \leftarrow f \cdot I_{T,P}(Q) / v_{T+P}(Q)$
  - 7:          $T \leftarrow T + P$
  - 8:     **end if**
  - 9: **end for**
  - 10:  $f \leftarrow f^{(p^k-1)/r}$
  - 11: **return**  $f$
-

## Line functions

- ▶ Note that the algorithm is of two parts – the “Miller loop” and the “final exponentiation”.
- ▶ Assume standard Weierstrass representation of the curve with affine coordinates.
- ▶ Observe that  $P$  is being multiplied by its own group order, using a standard double-and-add algorithm, and so ends up at  $O$ .
- ▶ The line and vertical functions  $l_{A,B}(Q)$  and  $v_{A+B}(Q)$  are distances calculated between  $Q$  and the lines that arise when adding  $B$  to  $A$ .
- ▶ If  $A$  is  $(x_j, y_j)$  and  $A + B$  is  $(x_{j+1}, y_{j+1})$ , and  $Q$  is  $(x_Q, y_Q)$ , and the line through  $A$  and  $B$  has slope  $\lambda_j$ , then..
- ▶  $l_{A,B}(Q) \leftarrow (y_Q - y_j) - \lambda_j(x_Q - x_j)$
- ▶  $v_{A+B}(Q) \leftarrow (x_Q - x_{j+1})$

# First Optimisation

- ▶ Choose  $r$ , a fixed system parameter, to have a low Hamming weight (if possible).
- ▶ (if not modify the algorithm to represent  $r$  as a NAF, do some precomputation, and use a standard windowing algorithm rather than simple double-and-add.)
- ▶ Choose  $P$  as a point on the base field  $E(\mathbb{F}_p)$ . Obviously we can't do this for  $Q$  as well, as then  $P$  and  $Q$  would be linearly dependent, and the pairing degenerate. But by doing it for  $P$  the point doublings and additions will obviously be a lot faster.
- ▶ Restrict  $k = 2d$  to be even. This brings many advantages. Observe that the final exponentiation is now  $f^{(p^d-1)[(p^d+1)/r]}$ .

## Extension field arithmetic

- ▶ Since  $k = 2d$  is even, we can represent extension field elements as  $(a + ib)$ , where  $a, b \in \mathbb{F}_{p^d}$ , and  $i$  is a quadratic non-residue and  $\delta = i^2$ .
- ▶ Multiplication: (Cost = 3M)  
$$(a + ib)(c + id) = ac + \delta bd + i[(a + b)(c + d) - ac - bd]$$
- ▶ Squaring: (Cost = 2M)  
$$(a + ib)(a + ib) = (a + b)(a + \delta b) - ab - \delta ab + i.2ab$$
- ▶ Frobenius: (Cost = free)  $(a + ib)^{p^d} = (a - ib)$
- ▶ Best to implement the full  $k$ -th extension as a “tower” of extensions. So for  $k = 12$  representation could be as a quadratic extension of a cubic extension of a quadratic extension over the base field.

# Improved Algorithm

---

**Algorithm 2** Computation of  $e(P, Q)$  with some optimizations

---

INPUT:  $P \in E(\mathbb{F}_p)$ ,  $Q \in E(\mathbb{F}_{p^k})$ , where  $P$  has order  $r$

OUTPUT:  $e(P, Q)$

```
1:  $T \leftarrow P, f \leftarrow 1$ 
2: for  $i \leftarrow \lfloor \lg(r) \rfloor - 1$  downto 0 do
3:    $f \leftarrow f^2 \cdot l_{T,T}(Q) / v_{2T}(Q)$ 
4:    $T \leftarrow 2T$ 
5:   if  $r_i = 1$  then
6:      $f \leftarrow f \cdot l_{T,P}(Q) / v_{T+P}(Q)$ 
7:      $T \leftarrow T + P$ 
8:   end if
9: end for
10:  $f \leftarrow fp^{d-1}$ 
11:  $f \leftarrow f^{(p^d+1)/r}$ 
12: return  $f$ 
```

---

## The amazing implications of $f \leftarrow f^{(p^d-1)}$

- ▶ Visualise fully “unrolling” the Miller loop and writing down the  $f$  function as the product of multiple line functions, each now individually raised to the power of  $p^d - 1$ .
- ▶ All sub-field components, that is elements of  $\mathbb{F}_{p^d}$ , become equal to 1 when raised to the power of  $p^d - 1$ . So they can be ignored when calculating the line functions. They can also be used to simplify line functions.
- ▶ It is easy (Frobenius) to see that  $(1/(a + ib))^{p^d-1} = (a - ib)^{p^d-1}$
- ▶ Therefore after raising to the power of  $p^d - 1$  inversions and conjugations become identical operations. So all divisions in the Miller loop can be replaced by multiplications by the conjugate of the divisor.
- ▶ Any divisions by small constants if required for optimal extension field arithmetic (e.g Toom-Cook methods) can be ignored.

## The amazing implications of $f \leftarrow f^{(p^d-1)}$ (Cont.)

- ▶ After raising to the power of  $p^d - 1$ , elements  $a + ib$  have a norm  $a^2 - \delta.b^2 = 1$ . Such an element is called a *unitary* element, and unitary elements have a faster squaring operation (Stam-Lenstra (2002))
- ▶ Squaring: (Cost = 2S)  
 $(a + ib)(a + ib) = 2\delta b^2 + 1 + i[(a + b)^2 - b^2 - \delta b^2 - 1]$
- ▶ ...which can be used to speed up the final exponentiation. Note that a squaring is *much* cheaper than a multiplication when at the top of a tall tower...
- ▶ Unitary elements can be compressed, and exponentiated using Lucas sequence (or XTR methods) (S.-Barreto (2004))

## Further Improved Algorithm

---

**Algorithm 3** Computation of  $e(P, Q)$  with further optimization

---

INPUT:  $P \in E(\mathbb{F}_p)$ ,  $Q \in E(\mathbb{F}_{p^k})$ , where  $P$  has order  $r$

OUTPUT:  $e(P, Q)$

```
1:  $T \leftarrow P, f \leftarrow 1$ 
2: for  $i \leftarrow \lfloor \lg(r) \rfloor - 1$  downto 0 do
3:    $f \leftarrow f^2 \cdot l_{T, T}(Q) \cdot \bar{v}_{2T}(Q)$ 
4:    $T \leftarrow 2T$ 
5:   if  $r_i = 1$  then
6:      $f \leftarrow f \cdot l_{T, P}(Q) \cdot \bar{v}_{T+P}(Q)$ 
7:      $T \leftarrow T + P$ 
8:   end if
9: end for
10:  $f \leftarrow f^{p^d - 1}$ 
11:  $f \leftarrow f^{(p^d + 1)/r}$ 
12: return  $f$ 
```

---

## What about $Q$ ?

- ▶ Currently  $Q$  is of the form  $(x_Q, y_Q)$ , where  $x_Q = a + ib$  and  $y_Q = c + id$ , with  $a, b, c, d \in \mathbb{F}_{p^d}$ .
- ▶ Now fix  $b = c = 0$ . Points in this form remain in this form under point multiplication – there is a group of order  $r$  of this nice form (the Trace-Zero group).
- ▶ Now the vertical line functions are entirely  $\in \mathbb{F}_{p^d} \dots$
- ▶ .. so they get wiped out by  $f^{(p^d-1)}$ . So we can ignore them!
- ▶ Denominator elimination (Barreto-Kim-Lynn-S. (2002)).

## What about Q? (continued)

- ▶ If  $Q(a, id)$  is a point on  $E(\mathbb{F}_{p^k})$ , then it can be mapped to a point on the isomorphic group on the quadratic twist of this curve  $E'(\mathbb{F}_{p^d})$ .
- ▶ So now Q is “smaller”.
- ▶ Its coordinates can be manipulated directly by a slightly modified line function.

## Further Optimized Pairing Algorithm

---

**Algorithm 4** Computation of  $e(P, Q)$  with denominator elimination

---

INPUT:  $P \in E(\mathbb{F}_p)$ ,  $Q \in E'(\mathbb{F}_{p^d})$ , where  $P$  has order  $r$

OUTPUT:  $e(P, Q)$

- 1:  $T \leftarrow P, f \leftarrow 1$
  - 2: **for**  $i \leftarrow \lfloor \lg(r) \rfloor - 1$  **downto** 0 **do**
  - 3:      $f \leftarrow f^2 \cdot l_{T, T}(Q)$
  - 4:      $T \leftarrow 2T$
  - 5:     **if**  $r_i = 1$  **then**
  - 6:          $f \leftarrow f \cdot l_{T, P}(T, Q)$
  - 7:          $T \leftarrow T + P$
  - 8:     **end if**
  - 9: **end for**
  - 10:  $f \leftarrow f^{p^d - 1}$
  - 11:  $f \leftarrow f^{(p^d + 1)/r}$
  - 12: **return**  $f$
-

## Some further tweaks

- ▶ Since  $r$  is odd, the last iteration requires an addition.
- ▶ But it can easily be established that the contribution of the last line function is always in  $\mathbb{F}_{p^d}$ .
- ▶ So it can be ignored (Duursma-Lee).
- ▶ The final exponent can be further broken down into the components  $p^d - 1$ ,  $(p^d + 1)/\Phi_k(p)$  and  $\Phi_k(p)/r$ .
- ▶ (for  $k = 12$ ,  $p^6 - 1$ ,  $p^2 + 1$  and  $(p^4 - p^2 + 1)/r$ ).
- ▶ That last component is “the hard part of the final exponentiation”.

## Yet Further Optimized Pairing Algorithm

---

**Algorithm 5** Computation of  $e(P, Q)$  with yet more optimization

---

INPUT:  $P \in E(\mathbb{F}_p)$ ,  $Q \in E'(\mathbb{F}_{p^d})$ , where  $P$  has order  $r$

OUTPUT:  $e(P, Q)$

- 1:  $T \leftarrow P$ ,  $f \leftarrow 1$ ,  $s \leftarrow r - 1$
  - 2: **for**  $i \leftarrow \lfloor \lg(s) \rfloor - 1$  **downto** 0 **do**
  - 3:      $f \leftarrow f^2 \cdot l_{T, T}(Q)$
  - 4:      $T \leftarrow 2T$
  - 5:     **if**  $s_i = 1$  **then**
  - 6:          $f \leftarrow f \cdot l_{T, P}(Q)$
  - 7:          $T \leftarrow T + P$
  - 8:     **end if**
  - 9: **end for**
  - 10:  $f \leftarrow f^{p^d - 1}$
  - 11:  $f \leftarrow f^{(p^d + 1) / \Phi_k(p)}$
  - 12:  $f \leftarrow f^{\Phi_k(p) / r}$
  - 13: **return**  $f$
-

## Some other nice ideas

- ▶ Use projective coordinates for faster point doubling/addition.
- ▶ Integrate the point doubling/addition with the line function calculation (Chatterjee-Sarkar-Barua (2004)).
- ▶ Use the optimal point doubling/addition formulae (Bernstein-Lange (2009)).
- ▶ In some application  $P$  may be a fixed parameter, in which case all of its multiples may be precalculated (In this case revert to affine coordinates).

## Curve-dependent Optimizations

- ▶ Many pairing-friendly families of curves have a simple polynomial form for the prime modulus  $p(x)$ ,  $r(x)$  and  $t(x)$ .
- ▶ First idea – exploit the form of  $p(x)$  for a faster modular arithmetic (Koblitz-Menezes (2005)).
- ▶ Not really demonstrated successfully, (except recently in hardware?). Concerns for security implications for DL problem.
- ▶ The first two parts of the final exponentiation can be calculated at the cost of a few Frobenius operations and a single extension field division
- ▶ Note that the hard part of the final exponent is a constant parameter-dependent value.

## Final Exponentiation

- ▶ For MNT  $k = 6$  prime-order curves,  $p(x) = x^2 + 1$ ,  $r(x) = x^2 - x + 1$ .
- ▶ The “hard exponent” is  $(p^2 - p + 1)/r$ . By simple substitution this becomes  $x^2 + x + 1 = p + x$
- ▶ Therefore the hard part of the exponentiation becomes  $f \leftarrow f^p \cdot f^x$ , which requires one application of the Frobenius and a powering to the power of  $x$ .
- ▶ Observe that  $x$  has just half the number of bits as  $(p^2 - p + 1)/r$ , so this idea has merit!
- ▶ All parameterised families of pairing-friendly curves benefit to some degree (See S.-Benger-Charlemagne-Dominguez-Kachisa (2008)).
- ▶ For many families (e.g. BN curves)  $x$  can be chosen with low Hamming weight, for further speed-ups.
- ▶ Optimal addition chains can be precalculated and used to combine individual components.

## Final Exponentiation (Cont.)

- ▶ Recall that in the hard part of the final exponentiation we can use the Stam-Lenstra (2002) fast squaring formula.
- ▶ For BN curves using standard methods a squaring costs  $36m$ , where  $m$  is an  $\mathbb{F}_p$  multiplication.
- ▶ Using Chung-Hasan (2006) method and the Stam-Lenstra idea, the cost is just  $24m$ .
- ▶ Using an alternative Stam-Lenstra idea, the cost can be as low as  $18m$  (with bad overheads).

## Truncated Loop Pairings

- ▶ The  $\eta_T$  pairing algorithm for supersingular curves of small characteristic demonstrated the possibility of a “truncated loop pairing” (building on work of Duursma and Lee).
- ▶ A bilinear pairing, closely related to the Tate pairing, but with a shorter Miller loop.
- ▶ Hess-Smart-Vercauteren (2006) demonstrated a similar idea for ordinary curves. The ate pairing.
- ▶ Key idea: “Swap”  $P$  and  $Q$ . Use  $t - 1$  as the loop multiplier, rather than  $r$ .
- ▶ Its still bilinear!

## Truncated Loop Pairings (cont.)

- ▶ Note that  $\log_2(t - 1)$  can be as small as  $\log_2 r/\varphi(k)$ .
- ▶ ...and even if its not the rather more complex R-ate pairing (Lee-Lee-Park (2008)) can achieve basically the same thing.
- ▶ We will however assume that our pairing friendly curve is “ate-friendly” and achieves this lower bound.
- ▶ This is potentially a big saving (for  $k = 12$  the loop with be one-quarter size)
- ▶ ...but now  $P$  is over the bigger twisted field, and so point doublings/additions much more expensive ☹.

## ate pairing

---

**Algorithm 6** Computation of  $e(P, Q)$  with ate pairing

---

INPUT:  $P \in E'(\mathbb{F}_{p^d})$ ,  $Q \in E(\mathbb{F}_p)$ ,  $P$  has order  $r$

OUTPUT:  $e(P, Q)$

- 1:  $T \leftarrow P$ ,  $f \leftarrow 1$ ,  $s \leftarrow t - 1$
  - 2: **for**  $i \leftarrow \lfloor \lg(s) \rfloor - 1$  **downto** 0 **do**
  - 3:      $f \leftarrow f^2 \cdot l_{T, T}(Q)$
  - 4:      $T \leftarrow 2T$
  - 5:     **if**  $s_i = 1$  **then**
  - 6:          $f \leftarrow f \cdot l_{T, P}(Q)$
  - 7:          $T \leftarrow T + P$
  - 8:     **end if**
  - 9: **end for**
  - 10:  $f \leftarrow f^{p^d - 1}$
  - 11:  $f \leftarrow f^{(p^d + 1) / \Phi_k(p)}$
  - 12:  $f \leftarrow f^{\Phi_k(p) / r}$
  - 13: **return**  $f$
-

## Higher Order Twists

- ▶ Many pairing-friendly families of curves have a low CM discriminant of  $D = 1$  or  $D = 3$
- ▶ These have quartic and sextic twists respectively.
- ▶ Now it is possible (if  $4|k$  or  $6|k$  resp.) to use these higher order twists for  $P$
- ▶ So for  $k = 12$ ,  $P \in \mathbb{F}_{p^2}$ . This is much better
- ▶ As a bonus the line functions  $l_{A,B}(Q)$  when mapped from the twisted curve are quite sparse over the full extension field, which can be further exploited.

## Related Computations

- ▶ Fast point multiplication over extension fields – see Galbraith-S. (2008) for fast method exploiting Gallant-Lambert-Vanstone (2001) idea.
- ▶ Fast hashing to points on curves over extension fields – see S.-Benger-Charlemagne-Dominguez-Kachisa (2008)

## Further Work

- ▶ Faster extension field arithmetic? Should be possible.
- ▶ Further loop reduction? For  $k = 2$  NSS curves (S. (2005)) a shorter loop is possible (exploiting an endomorphism).
- ▶ Better and more pairing-friendly curve representations (Edwards curves?).
- ▶ Q. What is the optimal curve representation for *combined* point addition/doubling PLUS line function calculation?
- ▶ What about a tool that reads is pairing friendly polynomials for  $p(x)$ ,  $r(x)$  and  $t(x)$ , and emits optimal pairing code?

## Example timing - BN $k = 12$ curves (AES-128)

- ▶ On an Intel Core 2 Quad Q9400 processor, 2.66GHz, using one core only.
- ▶ C++, no assembly language, Microsoft 64-bit compiler.
- ▶ 256-bit BN curve, variable  $P$ , requires 7499  $\mathbb{F}_p$  modmuls for the Miller loop, and 7156  $\mathbb{F}_p$  modmuls for the final exponentiation.
- ▶ R-ate pairing calculation requires 3.9ms.

## Question Time

- ▶ Thank you for your attention