



Efficient Implementation of Cryptographic pairings

Mike Scott
Dublin City University



First Steps

- To do Pairing based Crypto we need three things

- Efficient algorithms
- Suitable elliptic curves
- Hard problems

- We got them all!



Implementations

- On PCs and workstations
- Smartcards
- Wireless Sensor Network nodes
- FPGAs and Hardware..



What's a Pairing?

- $e(P, Q)$ where P and Q are points on an elliptic curve.
- It has the property of *bilinearity*
- $e(aP, bQ) = e(bP, aQ) = e(P, Q)^{ab}$
- Apologies for “Retro” additive notation!



Hard problems...

1. Given aP and P , its hard to find a
2. Given $e(P,Q)^a$ and $e(P,Q)$ its hard to find a .
3. Given $\{P, sP, aP, Q, sQ, bQ\}$ its hard to find $e(P,Q)^{sab}$
4. *Etc Etc Etc!!!*



Making it secure

- Recall that on a pairing friendly elliptic curve $E(F_q)$, the curve order has a large prime divisor r , and where k is the smallest integer such that $r|q^k-1$
- k is the embedding degree, a.k.a. the security multiplier



Making it secure

- If r is 160-bits, then Pohlig-Hellman attacks will take $\sim 2^{80}$ steps
- If $k.\lg(q) \sim 1024$ bits, Index calculus discrete Log attacks will also take $\sim 2^{80}$ steps
- So we can achieve appropriate levels of cryptographic security



Supersingular Curves

- Supersingular curves have small k and support a distortion map, $\phi(Q)$ which evaluates as a point on $E(F_{q^k})$, if Q is on $E(F_q)$,

- So choose P and Q on $E(F_q)$, then

$$\hat{e}(P, Q) = e(P, \phi(Q))$$

- Is an alternative, nicer pairing, with the extra property $\hat{e}(P, Q) = \hat{e}(Q, P)$



What choices for Supersingular?

- If $q=p$ a prime, maximum $k=2$
- If $q=2^m$, maximum $k=4$
- If $q=3^m$, maximum $k=6$

- We need group size $r \geq 160$ bits
- We need $q^k \geq 1024$ bits
- We know $r \mid q+1-t$
- (t is trace of the Frobenius $\leq 2 \sqrt{q}$)



Constrained...

- These constraints are... well... constraining!
- I have an irrational aversion to F_{3^m} !
- So what about Hyperelliptic curves...?
- Not very promising in practice...???
- Fortunately, we have an alternative choice – certain families of ordinary elliptic curves over F_p



Pairing Friendly Ordinary Elliptic Curves

- There are the MNT curves, with $k = \{3, 4, 6\}$
- There are Freeman curves with $k = 10$
- There are Barreto-Naehrig curves with $k = 12$



Ordinary Elliptic Curves

- These curves all have $r \sim p$, which is nice, as it means P can be over the smallest possible field for given level of security
- If we relax this, many more families can be found (e.g. Brezing-Weng)
- If we allow $\lg(r) \leq 2.\lg(p)$ then curves for any k are (relatively) plentiful (Cocks-Pinch)



The bad news..

- No distortion map ☹️
- In $e(P, Q)$, while P can be in $E(F_p)$, Q cannot
- The best we can do is to put Q on a lower order “twist” $E(F_{p^k/w})$, where always $w=2$, (but $w=4$ and $w=6$ are possible).
- For example for BN curves we can use $w=6$ and put Q on $E(F_{p^2})$
- $e(P, Q) \neq e(Q, P)$



Pairing types

- Galbraith, Paterson & Smart
“Pairings for Cryptographers”
- Very important paper!
- Type 1 - $G_1 \times G_1 \Rightarrow G_T$ (supersingular)
- Type 2 - $G_2 \times G_2 \Rightarrow G_T$
- Type 3 - $G_1 \times G_2 \Rightarrow G_T$
- Types 1 & 2 can be built on top of Type 3. Here we consider Type 3.



Implementation

- For simplicity (for now)
- Assume $k=2d$, $d=1$, $p=3 \pmod{4}$
- Elements in F_{p^2} can be represented as $(a+ib)$, where a and b are in F_p and $i=\sqrt{-1}$ because -1 is a quadratic non-residue (think “imaginary number”)
- Assume P is in $E(F_p)$, Q in $E(F_{p^2})$

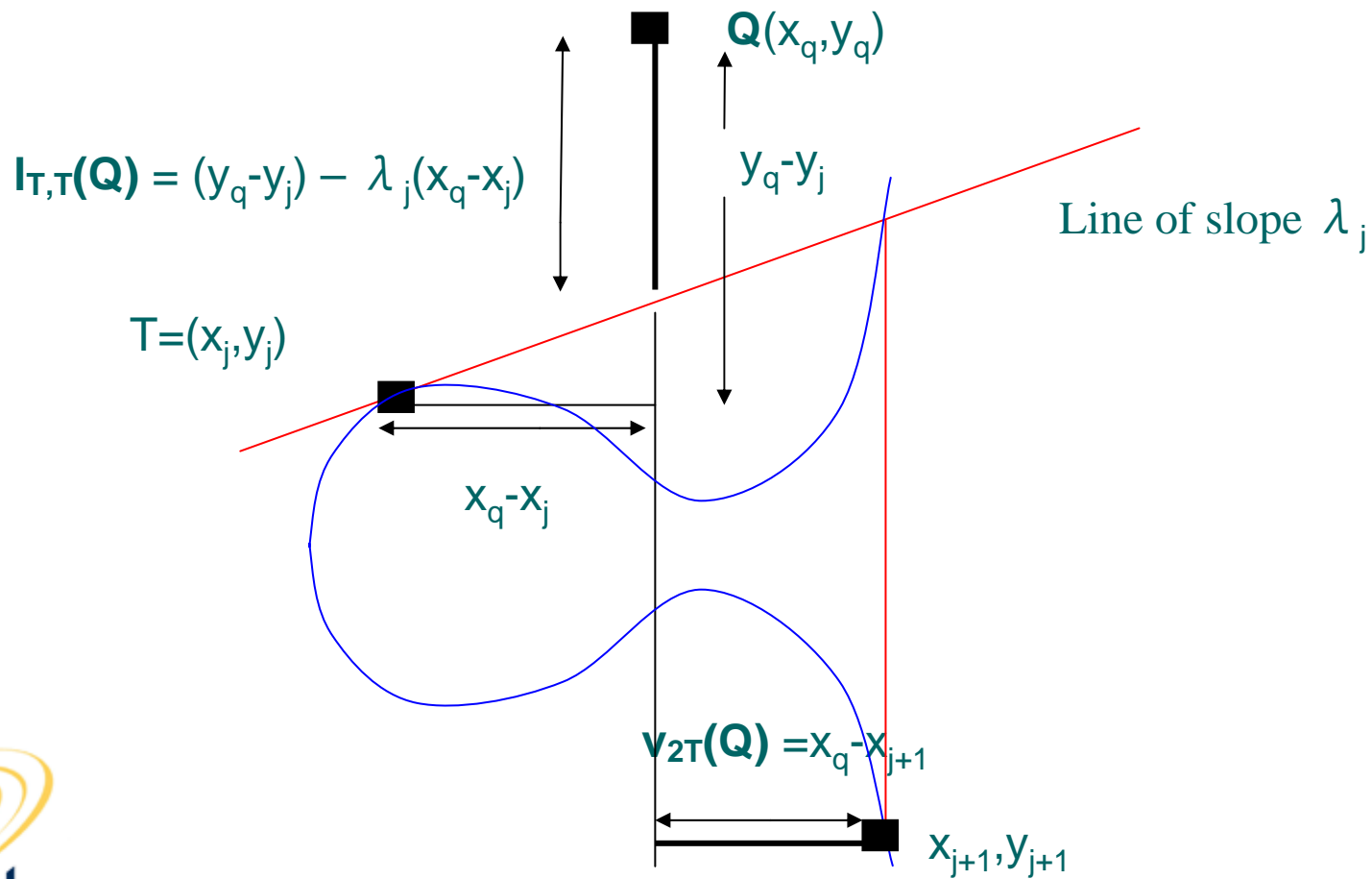
Basic Algorithm for $e(P, Q)$

```
 $m \leftarrow 1, T \leftarrow P$   
for  $i = \lg(r) - 1$  downto 0 do  
     $m \leftarrow m^2 \cdot l_{T,T}(Q) / v_{2T}(Q)$   
     $T \leftarrow 2 \cdot T$   
    if  $r_i = 1$   
         $m \leftarrow m \cdot l_{T,P}(Q) / v_{T+P}(Q)$   
         $T = T + P$   
    end if  
end for  
 $m \leftarrow m^{(p-1)}$   
return  $m^{(p+1)/r}$ 
```

Millers Algorithm

Final Exponentiation

Explaining the Algorithm





Optimizations

- Choose r to have a low Hamming weight
- By cunning choice of Q as a point on the twisted curve and using only even $k=2d$, the $v(\cdot)$ functions become elements in F_{p^d} and hence get “wiped out” by the final exponentiation, which always includes p^d-1 as a factor of the exponent.
- Now the algorithm simplifies to...

Improved Algorithm

```
 $m \leftarrow 1, T \leftarrow P$   
for  $i = \lg(r) - 1$  downto 0 do  
     $m \leftarrow m^2 \cdot l_{T,T}(Q)$   
     $T \leftarrow 2 \cdot T$   
    if  $r_i = 1$   
         $m \leftarrow m \cdot l_{T,P}(Q)$   
         $T = T + P$   
    end if  
end for  
 $m \leftarrow m^{(p-1)}$   
return  $m^{(p+1)/r}$ 
```



A useful Observation..

- Observe the line $m \leftarrow m^{(p-1)}$
- Part of the final exponentiation - raising to the power of $(p^k-1)/r$
- Now for any c in F_p , $c^{(p-1)} = 1 \pmod p$ (Fermat)
- Therefore m can be multiplied by any constant at any time in the Miller loop without effecting the final result!



Further optimization ideas

- Truncate the loop in Miller's algorithm, and still get a viable pairing.
- Optimize the final exponentiation
- Exploit the *Frobenius* – an element of any extension field F_{q^k} can easily be raised to any power of q . For example in F_{p^2}

$$(a+ib)^p = (a-ib)$$



Further optimization ideas

- Precomputation!
- If P is fixed, all the T values can be precomputed and stored – with significant savings.
- P may be a fixed public value or a fixed secret key – depends on the protocol.



Ate Pairing for ordinary curves $E(F_p)$

- Truncated Loop pairing, related to Tate pairing.
- Number of iterations in Miller loop may be much shorter – $\lg(t-1)$ instead of $\lg(r)$, and for some families of curves t can be much less than r
- Parameters “change sides”, now P is on the twisted curve and Q is on the curve over the base field.
- Works particularly well with curves that allow a higher order (sextic) twist.



Extension Field Arithmetic

- For non-supersingular curves over F_{pk} there is a need to implement very efficient extension field arithmetic.
- A new challenge for cryptographers (although XTR and OEFs require it)
- Simple generic polynomial representation will be slow, and misses optimization opportunities.



Towering extensions

- Consider $p \equiv 5 \pmod{8}$
- Then a suitable representation for F_{p^2} would be $(a+xb)$, where a, b are in F_p , $x = (-2)^{1/2}$, as -2 will be a QNR.
- Then a suitable representation for F_{p^4} would be $(a+xb)$, where a, b are in F_{p^2} , $x = (-2)^{1/4}$
- Etc!



Towering extensions

- In practise it may be sufficient to restrict $k=2^i3^j$ for $i \geq 1, j \geq 0$, as this covers most useful cases.
- So only need to deal with cubic and quadratic towering.
- These need only be efficiently developed once (using Karatsuba, fast squaring, inversion, square roots etc.)

Multiplication & Squaring (quadratic extension)

- Using Karatsuba..
- $(a+ib)(c+id) = ac - bd + i[(a+b)(c+d) - ac - bd]$
- Requires 3 modmuls...
- OR 3 multiplications and 2 modular reductions ("lazy" reduction)
- $(a+ib)(a+ib) = (a+b)(a-b) + i.2ab$



Multiplication & Squaring (cubic extension)

- Toom-Cook for multiplication?
- Chung-Hasan for squaring?
- A problem with both methods is the requirement for division by small constants...
- Not a problem thanks to the “useful observation”!



Choice of irreducible polynomial

- A binomial is the simplest, $x^n + \delta$, and the easiest to tower over.
- For example for $k=12$ BN curves
- $X^6 + (1 + \sqrt{-2})$ as a sextic tower over $x^2 + 2$, where $(1 + \sqrt{-2})$ is neither a cube nor a square.
- ..and so the sextic extension can be constructed as quadratic over a cubic



Choice of irreducible polynomial

- In general the k -th extension can often be constructed as $x^{k/2} + (a + \sqrt{\beta})$ towered over $x^2 + \sqrt{\beta}$, where a, β in $\{-1, +1, -2, +2\}$
- In practise this seems to work well, and the small values for a, β lead to useful speed-ups.
- Not too restrictive..



The Final Exponentiation - 1

- Note that the exponent is $(p^k-1)/r$
- This is a number dependent only on fixed, system parameters
- So maybe we can choose p , k and r to make it easier (Low Hamming Weight?)
- If $k=2d$ is even then

$$(p^k-1)/r = (p^d-1) \cdot [(p^d+1)/r]$$



The Final Exponentiation - 2

- We know that r divides (p^d+1) and not (p^d-1) from the definition of k .
- Exponentiation to the power of p^d is “for free” using the Frobenius, so exponentiation to the power of p^d-1 costs just a Frobenius and a single extension field division – cheap!



The Final Exponentiation - 3

- In fact we know that the factorisation of (p^k-1) always includes $\Phi_k(p)$, where $\Phi_k(.)$ is the k -th cyclotomic polynomial, and that $r|\Phi_k(p)$.

- For example

$$p^6-1 = (p^3-1)(p+1)(p^2-p+1)$$

- Where $\Phi_6(p) = p^2-p+1$



The Final Exponentiation - 4

- So the final exponent is general breaks down as...

$$(p^d - 1) \cdot [(p^d + 1) / \Phi_k(p)] \cdot \Phi_k(p) / r$$

- All except the final $\Phi_k(p) / r$ part can be easily dealt with using the Frobenius.



The Final Exponentiation - 5

- However this “hard” exponent e can always be represented to base p as

$$e = e_0 + e_1p + e_2p^2 \dots$$

$$f^e = f^{e_0 + e_1p + e_2p^2 \dots} = f^{e_0} \cdot (f^p)^{e_1} \cdot (f^{p^2})^{e_2} \dots$$

- Which can be calculated using the Frobenius and the well known method of multi-exponentiation.



The Final Exponentiation - 6

- Another idea is to exploit the special form of the “hard part” of the final exponentiation for a particular curve
- If k is divisible by 2 the pairing value can be “compressed” times 2 and Lucas exponentiation used.
- If k is divisible by 3 the pairing value can be “compressed” times 3 and XTR exponentiation used.



Case study – $k=6$ MNT curves

- Assuming a prime order curve, then the hard part of the final exponentiation is $(p^2-p+1)/r$, where $r=p+1-t$
- Then the exponent is $p \pm \sigma$, where $\sigma \sim t$
- So the final exponentiation is $f^p \cdot f^{\pm\sigma}$
- Which just costs a Frobenius and one half-length exponentiation!



Products of pairings

- Arises in many protocols
- $e(P,Q).e(R,S) \Rightarrow 3$ ideas
 - The multiplication of P and R by r occur in “lock-step”, so use Montgomery’s trick, affine coordinates, only one modular inversion
 - Share the “Miller variable” m (so only one squaring of m in the Miller loop)
 - Share the final exponentiation



Implementation – more complex than RSA or ECC!

- There are many choices of curves, pairings (Type 1, 2, or 3?) and of embedding degrees. It is not at all obvious which is “best” for any given application. The optimal pairing to use depends not just on the security level, but also on the protocol to be implemented.



Implementation – more complex than RSA or ECC!

- For example (a) $p \sim 512$ bits and $k=2$, or (b) $p \sim 170$ bits and $k=6$ MNT curve?
- On the face of it same security.
- Smaller p size means faster base field point multiplications – so (b) looks better
- Which is important only if point multiplications are required by the protocol.
- (a) pairing is much faster if precomputation is possible
- (b) must be used for short signatures
- (b) requires Q on the twist $E'(F_{p^3})$ which is more complicated than (a) for which Q can be on $E'(F_p)$
- The (b) curves are hard to find, whereas (a) types are plentiful.
- (a) is much simpler to implement with the smaller extension.. Smaller code



Implementation – more complex than RSA or ECC!

- For maximum efficiency each implementation must be highly specialised according to its parameters.
- A $k=2$ Cocks-Pinch implementation will be quite different from a $k=6$ MNT implementation.



The not-so-good news

- Keys sizes and computation time scale as RSA not ECC
- Not *quite* that bad – for example a 4096 bit element in F_{p^8} where p is a 512 bit prime is easier to work with than a 4096 bit element in Z_n
- Karatsuba easier to exploit, etc.



Scaling security

- RSA – use a bigger modulus, same program
- ECC – use a bigger modulus and a different curve, same program (perhaps with specialised modular reduction code?)
- PBC – use a bigger modulus (and/or a bigger embedding degree), and a different curve, and a radically different program. Or just scale up the modulus?



Some timings – 80-bit security

- 32-bit 3GHz PIV
- Tate Pairing
- $k=2$, $p \sim 512$ bits Cocks-Pinch
- w/o precomp. = 6.7ms
- With precomp. = 3.0ms
- Point mul. = 2.9ms



Some timings – 80-bit security

- 32-bit 3GHz PIV
- Tate Pairing
- $k=2$, $p \sim 512$ bits with Efficient Endomorphism (Scott '05)
- w/o precomp. = 5.1ms
- With precomp. = 3.0ms
- Point mul. = 1.9ms



Some timings – 80-bit security

- 32-bit 3GHz PIV
- Ate pairing
- $k=4$, $p \sim 256$ bits FST curve
- w/o precomp. = 9.1ms
- With precomp. = 3.1ms
- Point mul. = 1.1ms



Some timings – 80-bit security

- 32-bit 3GHz PIV
- Tate pairing
- $k=6$, $p \sim 160$ bits MNT curve
- w/o precomp. = 6.2ms
- With precomp. = 4.5ms
- Point mul. = 0.6ms



Some timings – 80-bit security

- 8-bit 16MHz Atmel128
- Tate pairing
- $k=4$, $p \sim 256$ bits MNT curve
- With precomp. = 7.75 seconds



Some timings – 80-bit security

- 8-bit 16MHz Atmel128
- η_T pairing
- $k=4$, $m=271$ bits, supersingular curve
- w/o precomp = 4.2 seconds
- (Sanity check – an ECC point multiplication at approx. the same security level takes 0.25 seconds!)



Some timings – 128-bit security

- 3.4GHz PIV 32-bit
- Tate pairing
- $k=12$, $p \sim 256$ bits BN curve
- w/o precomp. = 46.1ms
- Ate pairing
- w/o precomp. = 39.3ms



Questions Anyone?