# Efficient Implementation of Cryptographic pairings

Mike Scott
Dublin City University

DCU
Dublin City University
Ollscoil Chathair Bhaile Átha Cliath

# First Steps

- To do Pairing based Crypto we need two things
  - Efficient algorithms
  - Suitable elliptic curves
- We have got both! (Maybe not quite enough suitable curves?)

# What's a Pairing?

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

# 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 $e(P,Q)^{sab}$

# Why is a pairing useful

- A Trusted Authority has a secret $s$ and generates $P$ and $P_{pub}=sP$. He makes $P$ and $P_{pub}$ public.
- A user approaches the TA, proffers an identity $Q_{id}$, and is issued with a secret $D=sQ_{id}$

# Identity Based Encryption

- To encrypt a message to $Q_{ID}$, encrypt it using as key $e(Q_{ID},P_{pub})^w$ for random $w$ and append $U=wP$ to the ciphertext.
- To decrypt it use as key $e(D,U)$. This is the same key because of bilinearity
- $e(Q_{ID},P_{pub})^w=e(Q_{ID},P)^{sw}$
- $=e(sQ_{ID},wP)=e(D,U)$
- All possible attacks protected by a hard problem!

# Where to Find a Pairing?

- First Stop - Supersingular Elliptic curves $E(F_q)$, $q=p^m$
- The Tate Pairing $e(P,Q)$ has the required properties!
- If $P$ and $Q$ are points on $E(F_{qk})$, then pairing evaluates as element in $F_{qk}$
- If $P$ is of order $r$, so is $e(P,Q)$
- It is bilinear, and $k$ (the *embedding degree*) is of a "reasonable" size {2,4,6}

# Making it secure

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

# Modified Tate Pairing

- $k$ is *smallest* number such that $r|(q^k-1)$
- Supersingular curves support a distortion map, $\Phi(Q)$ which evaluates as a point on $E(F_{qk})$, 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 $\hat{e}(P,Q) = \hat{e}(Q,P)$

# Prove $\hat{e}(P,Q) = \hat{e}(Q,P)$ !

- If P and Q are points of order $r$ on $E(F_q)$, then Q=$c$P for some unknown $c$
- So $\hat{e}(P,Q) = \hat{e}(P,cP) = \hat{e}(P,P)^c$
- $= \hat{e}(cP,P) = \hat{e}(Q,P)$
- Observe the power of bilinearity!

# What choices?

- 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 $q^k \sim 1024$ 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 HATE $F_{3m}$ !
- So what about Hyperelliptic curves…?
- Not very promising in practice…
- Fortunately, we have an alternative choice – certain families of ordinary elliptic curves over $F_p$

# 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 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" $E(F_{pk/w})$, 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 $E(F_{p2})$
- $e(P,Q) \neq e(Q,P)$

# Implementation

- For simplicity (for now)
- Assume $k=2d$, $d=1$, $p=3 \bmod 4$
- Elements in $F_{p2}$ 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 $E(F_{p2})$

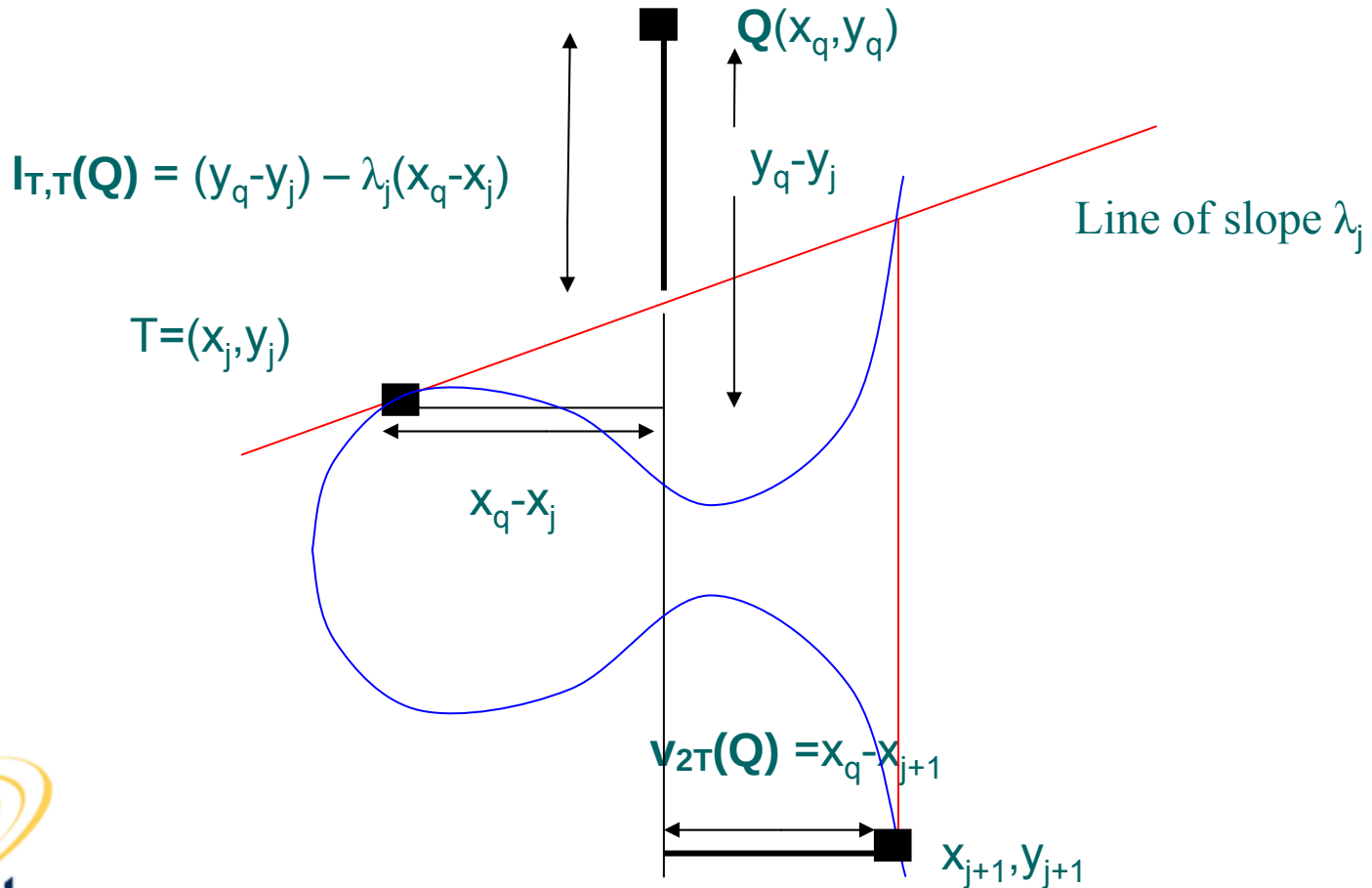
# Basic Algorithm for *e(P,Q)*

$m \leftarrow 1$, $T \leftarrow P$
for $i$=lg($r$)-1 downto 0 do
$m \leftarrow m^2 . l_{T,T}(Q)/v_{2T}(Q)$
$T \leftarrow 2.T$
if $r_i = 1$
$m \leftarrow m . l_{T,P}(Q)/v_{T+P}(Q)$
$T$=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(.)* functions become elements in $F_{pd}$ 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

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

# 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.

# The $\eta_T$ pairing - 1

- For the supersingular curves of low characteristic, the basic algorithm can be drastically simplified by integrating the distortion map, the point multiplication, and the action of the Frobenius directly into the main Miller loop. Also exploits the simple group order.

# The $\eta_T$ pairing - 2

- In characteristic 2, $k$=4.
- $r = 2^m \pm 2^{[(m+1)]/2} + 1$
- Elements in $F_{2^m}$ are represented as a polynomial with $m$ coefficients in $F_2$
- Elements in the *extension field* $F_{2^{4m}}$ are represented as a polynomial with 4 coefficients in $F_{2^m}$
- e.g. $a+bX+cX^2+dX^3$ represented as $[a,b,c,d]$.

# The $\eta_T$ pairing - 3

- Let $s$=[0,1,1,0] and $t$=[0,1,0,0] (derived from distortion map)
- Then on the supersingular curve $y^2+y=x^3+x+b$, where $b$=0 or 1
- And $m$= 3 mod 8
- A pairing $e(P,Q)$, where $P=(x_P,y_P)$ and $Q=(x_Q,y_Q)$, can be calculated as

# The $\eta_T$ pairing - 4

$u \leftarrow x_P+1$

$\boldsymbol{f} \leftarrow u(x_P+x_Q+1)+y_P+y_Q+b+1+(u+x_Q)s+t$

for $i$=1 to $(m+1)/2$ do

$u \leftarrow x_P \quad x_P \leftarrow \sqrt{x_P} \quad y_P \leftarrow \sqrt{y_P}$

$\boldsymbol{g} \leftarrow u(x_P+x_Q)+y_P+y_Q+x_P+(u+x_Q)s+t$

$\boldsymbol{f} \leftarrow \boldsymbol{f}.\boldsymbol{g} \quad x_Q \leftarrow x_Q^2 \quad y_Q \leftarrow y_Q^2$

end for

return $\boldsymbol{f}^{(2^{2m}-1)(2^m-2^{(m+1)/2}+1)}$

# The $\eta_T$ pairing - 5

- This is very fast! <5 seconds on an msp430 wireless sensor network node, with $m$=271 (C – no asm)
- Note truncated loop ($m$+1)/2.
- Final exponentiation very fast using Frobenius.
- Idea in low power, resource constrained environment.

# 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
- Simple generic polynomial representation will be slow, and misses optimization opportunities.

# Towering extensions

- Consider $p=5$ mod 8
- Then a suitable representation for $F_{p2}$ 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 $F_{p4}$ would be $(a+xb)$, where $a,b$ are in $F_{p2}$, $x=(-2)^{1/4}$
- 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.)

# 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

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

# 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…

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

- 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

$$e = e_0 + e_1 p + e_2 p^2 \ldots$$

$$f^e = f^{e_0 + e_1 p + e_2 p^2 \ldots} = f^{e_0} . (f^p)^{e_1} . (f^{p^2})^{e_2} \ldots$$

- 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.

## Implementation – more complex than RSA or ECC!

- There are many choices of curves, and of embedding degrees, and of pairings. 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) $p$~512 bits and $k$=2, or (b) $p$~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

# Some timings – 80-bit security

- 32-bit 3GHz PIV
- Tate Pairing
- k=2, p~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~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~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~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~256 bits MNT curve
- With precomp.  = 7.75 seconds

# Some timings – 80-bit security

- 8-bit 16MHz Atmel128
- $\eta_T$ pairing
- k=4, m=271 bits, supersingular curve
- w/o precomp = 4.6 seconds

# Some timings – 128-bit security

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

Questions Anyone?

DCU
Dublin City University
Ollscoil Chathair Bhaile Átha Cliath