An Introduction to Computing Cryptographic Pairings: Part 1

Craig Costello

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

Izmir Yasar University, Turkey

1st July, 2010

Some thoughts to carry with you...

- Pairings are extremely useful in cryptography
- Don't worry if you don't understand because I didn't either
- **Always remember, pairings are just a function**

$$e(P, Q) = f(x_P, y_P, x_Q, y_Q)$$

  of two points (four numbers)
- We can talk for as long as you like after the talk...

Pairing computation speeds: then and now

- **Then**:

  - 1993 Menezes' elliptic curve book : few minutes

...BIG GAP...

- **Now**:

  -2009 Hankerson, Menezes, Scott: 4.01ms
  -April 2010 Naehrig, Niederhagen, Schwabe: 1.80ms
  -June 2010 Beuchat *et al.*: 0.94ms

So what happened in the big gap?

- Heaps of exciting protocol stuff has happened...
  ID-based encryption (IBE), ID-based key agreement, short signatures, group signatures, ring signatures, certificateless encryption, hierarchical encryption, predicate-based encryption, attribute-based encryption, ... and many more!!!
- Heaps of cool pairing optimizations has since 'followed'...
  - *Tate pairing instead of Weil pairing*
  - *denominator elimination*
  - *group choices and twisted curves*
  - *endomorphism rings and loop shortening*
  - *low rho-valued curves*
  - *pairing and towering-friendly fields*
  - *... and many more!!!*
- Today we will just touch on some of the major stuff that has happened

Two parts

1. Elliptic curves (a group of points)
2. Pairings (a function of two points in these elliptic curve groups) with very useful properties

PART 1: Elliptic Curves

The search for "better" groups

- Groups are sets with a binary operation that have the following properties
  1. closure
  2. associativity
  3. identity
  4. inverse
- There may be groups in settings that we are (relatively) familiar with (integers $\mathbb{Z}$, rationals $\mathbb{Q}$, complex numbers $\mathbb{C}$)
- In what follows we will search for a slightly more abstract group (called Elliptic curves)
- These groups offer many advantages in cryptography (and elsewhere)

Some rough motivation on where to start

- **FIRST AND FOREMOST:** We want a setting where combining (operating on) two elements gives another elements
- Very very roughly: a polynomial of degree n has n roots over $\mathbb{C}$
- Therefore, a polynomial of degree 3 has 3 roots
- If we have two of the roots, this implies (allows us to determine) the third root
- None of this is very helpful yet and we're nowhere near a group... but it's a start

A step closer...

Theorem (Bezout)

Two projective curves of degree m and n having no component in common intersect in mn points.

- Let's generalise (in a sense) the statement about roots of polynomials on the previous slide
- The statement on the previous slide: take x-axis as a curve of degree 1... then n degree polynomial intersects x-axis in n places, i.e. has n roots.
- ... forget the x-axis from now on

A step closer...

- There will be 3 intersections of a line (curve of degree 1) and a cubic (curve of degree 3)
- Specifying two of them allows us to find the third, but how can we form a "big" cryptographically useful group out of the intersection of a line and a cubic (three points)???
- Behold the magic of the group definition on elliptic curves....

Elliptic Curve

- We are interested in cubic equations

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j = 0$$

- Any cubic with a rational point can be transformed into

$$y'^2 + px'y' + qy' = x'^3 + rx'^2 + sx' + t$$

Definition

An elliptic curve over $\mathbb{Q}$ is the set of points $(x_i, y_i) \in \mathbb{Q} \times \mathbb{Q}$ satisfying the equation

$$y^2 = x^3 + Ax + B$$

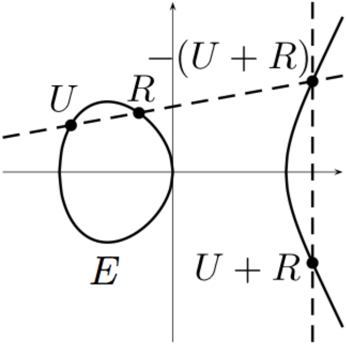
for some $A, B \in \mathbb{Q}$ where $4A^3 + 27B^2 \neq 0$ together with the point at infinity $\mathcal{O}$.

The Group Law

- We have a set of points (i.e. what an elliptic curve is!).
- Our goal is to form a group
- All we need is a binary operation (our group law).... LET'S DRAW/DERIVE IT!
- ... (it's also got to satisfy those four earlier properties)

"Addition"

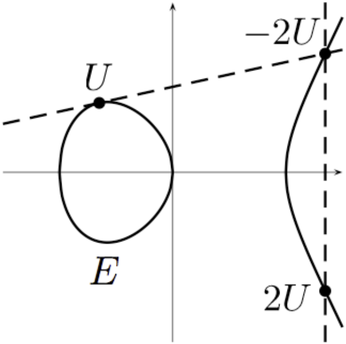
- We just want to perform an operation on two elements so that it gives another element
- We call it "adding"



- It's not traditional "adding"!

Doubling

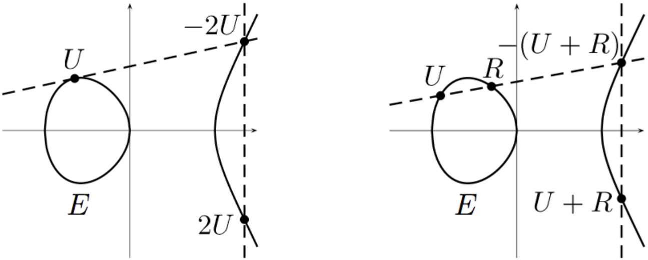
- What about when we want to add something to itself?
- Where's the line between a point and itself?



- Group law often called chord-and-tangent!

Deriving the affine group law

Let's draw and derive the group law (Viette's formula)



Are we right?

ADDITION $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$

$$\lambda = (y_2 - y_1)/(x_2 - x_1)$$
$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

DOUBLING $(x_3, y_3) = 2(x_1, y_1)$

$$\lambda = (3x_1^2 + a)/(2y_1)$$
$$x_3 = \lambda^2 - 2x_1$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

Point Addition

E : $y^2 = x^3 - x + 1$ over $\mathbb{Q}$

- $A = -1$ and $B = 1$.
- $\Delta = -4A^3 - 27B^2 = -23 \neq 0$, E is an elliptic curve.
- The point $P = (1, 1)$ is a rational point on E.
- $Q = P + P = (1, 1) + (1, 1) = (-1, 1)$
- $R = Q + P = (1, -1) + (1, 1) = (0, -1)$

Point Addition

E : $y^2 = x^3 - x + 1$ over $\mathbb{Q}$

- $S = R + P = \left(0, -1\right) + \left(1, 1\right) = \left(3, -5\right)$
- $T = S + P = \left(3, -5\right) + \left(1, 1\right) = \left(5, 11\right)$
- $U = T + P = \left(5, 11\right) + \left(1, 1\right) = \left(1/4, 7/8\right)$
- $V = U + P = \left(1/4, 7/8\right) + \left(1, 1\right) = \left(-11/9, -17/27\right)$
- $Y = V + P = \left(-11/9, -17/27\right) + \left(1, 1\right) = \left(19/25, -103/125\right)$

- Everything is starting to take shape, but the points are growing in size... we need to control this and have some consistency
- Instead of defining elliptic curves over the rationals, lets use finite fields
- For now, we will only consider prime fields $\mathbb{F}_p$ and their extensions

Point Addition: a toy example

Same curve as before $\mathbf{E} : y^2 = x^3 - x + 1$ over $\mathbb{F}_{13}$

- $\Delta = -4A^3 - 27B^2 = -23 \equiv 3 \bmod 13 \neq 0$, E is an elliptic curve.
- 19 Points: $(0,1)$, $(0,12)$, $(1,1)$, $(1,12)$, $(3,5)$, $(3,8)$,$(4,3)$,$(4,10)$,$(5,2)$,$(5,11)$, $(6,4)$,$(6,9)$,$(7,5)$,$(7,8)$,$(10,4)$, $(10,9)$, $(12,1)$, $(12,12)$....., oh, and don't forget $\mathcal{O}$.
- Let $P = (1,12)$ and $Q = (4,10)$ on E.
- $P + Q = (1,12) + (4,10) = (7,5)$.
- $2P = P + P = (1,12) + (1,12) = (12,12)$
- $3P = 2P + P = (12,12) + (1,12) = (0,1)$
- $4P = 3P + P = (0,1) + (1,12) = (3,5)$ ....

Point Addition

Same curve as before $\mathbf{E} : y^2 = x^3 - x + 1$ over $\mathbb{F}_{13}$

- ... $10 * P = P + P + ...P = (4, 10) = Q$
- Given P and Q, the discrete log problem to base P involves finding s such that $sP = Q$ and in this case $s = 10$
- It is simple to find the discrete logarithm (using brute force attack) when curve is defined over such a small field...
- But what if we increase the field size like we did before...

A cryptographically suitable curve

Same curve as before $\mathbf{E} : y^2 = x^3 - x + 1$ over $\mathbb{F}_p$

- $p = 1461501637330902918203684832716283019655932542983$
- $\#E = 1461501637330902918203686004385807989344528195053$
- $P = (1321554781015706068290537639827905592412509913620,$
  $1136877326354697828904160020005825111410953389610)$
- $r = 1156413885967954566959797563242567816342019303 88$
- $Q = rP =$
  $(715875109644815085946717311816604681845099700277,$
  $1450877329524262790654657764775612031321288027789)$

A cryptographically suitable curve

- It was easy for me (my computer) to multiply P by r to get Q (milliseconds)...
- ... but to get r from Q and P...
- A lazy brute force loop...
  $T = P$
  while $T \neq Q$ **do**
  $T = T + P$
  end while
- Loop will have to do $r \approx p$ additions before terminating (impossible)
- So r is buried inside the elliptic curve discrete logarithm problem (ECDLP)
- Attacks are much better than brute force (as always), but in this context they are much slower than what we may be used to

Elliptic curves in cryptography

- To put it simply, elliptic curves (on their own) just provide an alternative discrete logarithm problem (the ECDLP)
- Arithmetic in the forward direction, i.e. multiplying a point by an integer (finding rP from r and P) is very fast (double-and-add techniques)
- Finding r from P and rP is computationally “hard”
- In fact, for identical field sizes, the ECDLP is much harder than the DLP (somewhat intuitive)
- This is the beauty of elliptic curves - we can use much much smaller fields for discrete log based cryptosystems
- Actually, this is just the beginning...

DEEP BREATHE....

Questions so far?

PART 2: Pairings on elliptic curves

Don't forget...

- **Pairings are just a function**

$$e(P, Q) = f(x_P, y_P, x_Q, y_Q)$$

of two points (four numbers)

What complicates things

- The groups the points P and Q come from
- The degree and nature of this function
- More than one setting for the discrete log problem... attack complexity varies
- The need for speed...

What are pairings?

- A bilinear pairing e is just a mapping where 2 inputs "pair" to result in a value

$$e(P, Q) = t$$

- The bilinearity property is as follows:

$$e(aP, bQ) = t^{ab} = e(bP, aQ)$$

- Behold the magic of bilinearity (a quick key exchange). TA holds master secret s. Alice's identity is A. Bob's is B. TA issues Alice sA. TA issues Bob sB.

$$\text{Alice} \rightarrow e(sA, B) = e(A, B)^s = e(A, sB) \leftarrow \text{Bob}$$

Why Elliptic Curves?

- There is only one known mathematical setting where desirable bilinear pairings exist: (hyper)elliptic curves
- Therefore in $e(P, Q)$, P and Q are points on an elliptic curve
- Attacks on elliptic curves are much slower than on finite fields (160 bit group order for elliptic curves comparable to 1024 bit security in finite fields)

Lucky for elliptic curves

The fact that elliptic curves just happen to be more efficient than finite fields is a happy coincidence... we need them to pair with regardless

The General Pairing Definition

General Pairings

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$$

- $\mathbb{G}_1$ is almost always a subgroup of $E(\mathbb{F}_q)$.
- $\mathbb{G}_2$ is almost always a subgroup of $E(\mathbb{F}_{q^k})$.
- $\mathbb{G}_T$ is the multiplication group of a finite field $\mathbb{F}_{q^k}$ (k is called the embedding degree).

The embedding degree

k is called the embedding degree and plays a big role in pairing-based crypto...

Two Discrete Log Problems

- Recall from a couple of slides ago: TA issues Alice sA. Alice computes $e(sA, B) = e(A, B)^s$. Alice can compute $t = e(A, B)$ from identities.
- Andrew has $sA, A \in E(\mathbb{F}_q)$, as well as $t^s, t \in \mathbb{F}_{q^k}$.
- To find master secret s, Alice can attack whichever discrete log problem is easiest
- Pairing based cryptography is a balancing act
  1. Hard ECDLP in $\mathbb{G}_1$, $\mathbb{G}_2$
  2. Hard DLP in $\mathbb{G}_T$
  3. Efficient algorithms across all groups
- We can achieve “good balance” if we can be flexible with k

What else do we want in a pairing

General Pairings

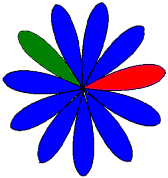
$$e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$$

1. We want to be able to efficiently hash random strings to $\mathbb{G}_1$ and $\mathbb{G}_2$
2. We (usually) want an efficiently computable isomorphism $\psi : \mathbb{G}_2 \longrightarrow \mathbb{G}_1$
3. We want to be flexible in our choice of the embedding degree k

- Unfortunately, achieving all three of these properties simultaneously is not currently possible
- Prior to this being well known, cryptographers often made incorrect assumptions

The r-torsion

- The points P and Q in the pairing come from the r-torsion $E[r] = \mathbb{Z}_r \times \mathbb{Z}_r$.

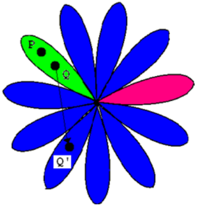


- The green subgroup is usually chosen as $\mathbb{G}_1$ for efficiency (base field)... all we need is another subgroup $\mathbb{G}_2$
- We can get out of the blue subgroups (trace map), but we can't hash into them
- Ironically, the only other subgroup we can hash to (the red subgroup), is the only one we can't map back out of.
- **Cryptographers must make a choice....**

Supersingular Curves

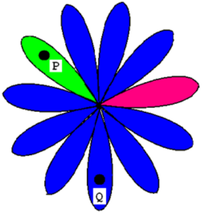
- The only time we can simultaneously do these two things is unfortunately on supersingular curves where our embedding degree $k = 2, 3, 6$ is restricted.
- This inability to satisfy all desired properties forces us to define different types of pairings, each with its own pros and cons

Type 1 Pairings



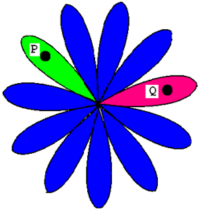
- Can efficiently hash both P and Q onto the base field subgroup
- Use the distortion map to send Q into a linearly independent subgroup
- Pairing defined over same group so isomorphism exists
- BUT... Supersingular curves only ($k = 2$ for large characteristic)

Type 2 Pairings



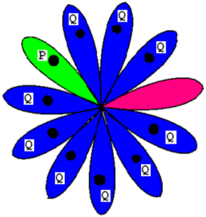
- Can efficiently hash P onto the base field subgroup
- The trace map will map Q back to the base field subgroup
- Available over all curves and embedding degrees
- BUT... cannot randomly sample from this blue group without knowing the discrete logarithm

Type 3 Pairings



- Can hash P and Q to their subgroups
- Available over all curves and embedding degrees
- BUT... no map from this Q's group back to P's group

Type 4 Pairings



- Can hash both P and Q onto their subgroups
- Available over all curves and embedding degrees
- There will always be a map back (the trace map)
- Cannot hash points into the same subgroup (no discrete log between two Q's)

Pairings in Protocols

- There have been schemes published that **incorrectly** assume that all properties of pairings can be utilised simultaneously
- Cryptographers must be careful when developing protocols that the pairings they need actually exist

Common group choices

$$\mathbb{G}_1 = E[r] \cap \ker(\pi_q - [1]) = E(\mathbb{F}_q)[r], \qquad \text{(the base field)}$$

$$\mathbb{G}_2 = E[r] \cap \ker(\pi_q - [q]) \subset E(\mathbb{F}_{q^k})[r], \qquad \text{(the full extension field)}$$

The elements of $\mathbb{G}_2$ are much bigger than the elements of $\mathbb{G}_1$ (e.g. $k = 12$)

$$\mathbb{F}_{q^{12}} = \mathbb{F}_{q^4}(\alpha) = \mathbb{F}_{q^2}(\gamma) = \mathbb{F}_q(\beta)$$

$P \in \mathbb{G}_1$: [341746248540, 710032105147]

$Q \in \mathbb{G}_2$:

$[((502478767360 \cdot \beta + 1034075074191) \cdot \gamma + 342970860051 \cdot \beta + 225764301423) \cdot \alpha^2 + ((205398279920 \cdot \beta + 182600014119) \cdot \gamma + 860891557473 \cdot \beta + 435210764901) \cdot \alpha + (1043922075477 \cdot \beta + 566889113793) \cdot \gamma + 150949917087 \cdot \beta + 21392569319,$

$((654337640030 \cdot \beta + 744622505639) \cdot \gamma + 1092264803801 \cdot \beta + 895826335783) \cdot \alpha^2 + ((529466169391 \cdot \beta + 550511036767) \cdot \gamma + 985244799144 \cdot \beta + 554170865706) \cdot \alpha + (194564971321 \cdot \beta + 969736450831) \cdot \gamma + (579122687888 \cdot \beta + 581111086076)]$

The twisted curve

- Original curve is $E(\mathbb{F}_q) : y^2 = x^3 + ax + b$
- Twisted curve is $E'(\mathbb{F}_{q^{k/d}})$: $y^2 = x^3 + a\omega^4 x + b\omega^6$, $\omega \in \mathbb{F}_{q^k}$
- Possible degrees of twists are $d \in \{2, 3, 4, 6\}$
- $d > 2$ requires $a = 0$ or $b = 0$
- Twist $\Psi : E' \to E : (x', y') \to (x'/\omega^2, y'/\omega^3)$ induces $\mathbb{G}_2' = E'(\mathbb{F}_{q^{k/d}})[r]$ so that $\Psi : \mathbb{G}_2' \to \mathbb{G}_2$
- Instead of working with $Q \in \mathbb{G}_2$, a lot of work can be done with $Q' \in \mathbb{G}_2'$ defined over subfield $\mathbb{F}_{q^e} = \mathbb{F}_{q^{k/d}}$

$P \in \mathbb{G}_1$: $(341746248540, 710032105147)$
$Q' \in \mathbb{G}_2' = \Psi^{-1}(\mathbb{G}_2)$:

$((917087150949\beta + 25693192139) \cdot \omega^2, (878885791226\beta + 860765811110) \cdot \omega^3)$

Theory behind Miller's algorithm

- **Recall: pairings are just a function**

$$e(P, Q) = f(x_P, y_P, x_Q, y_Q)$$

of two points (four numbers)

- The theory behind how this function is constructed and why it's bilinear is too in depth for today's discussion
- We will take Miller's algorithm for granted (for now)
- The pairing is computed as $e(P, Q) = f_{r,P}(Q)^{(q^k-1)/r}$, where $f_{r,P}(Q)$ would expand explicitly as

$$f_{r,P}(Q) = \sum_{i=0}^{r}\sum_{j=0}^{i} c_{i,j} \cdot x_Q^{i-j} y_Q^j,$$

where the $c_{i,j}$'s are entirely P dependent...

Miller's algorithm for $e(P, Q) = f_{r,P}(Q)^{(q^k-1)/r}$

Input: P, Q and $r = (r_{\lfloor \log(r) \rfloor}, ..., r_0)_2$
Output: $f_{r,P}(Q)^{(q^k-1)/r}$

- $f \leftarrow 1,\ T \leftarrow P$
- **for** i **from** $\lfloor \log(r) \rfloor - 1$ **to** 0 **do**
  1. Compute $g = l$ in the chord-and-tangent doubling of T
  2. $T \leftarrow [2]T$
  3. $f \leftarrow f^2 \cdot g(Q)$
  4. **if** $r_i = 1$ **then**
     i. Compute $g = l$ in the chord-and-tangent addition of $T + P$
     ii. $T \leftarrow T + P$
     iii. $f \leftarrow f \cdot g(Q)$

     end if

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

A good place to stop...

The next talk will be entirely about optimizing Miller's algorithm
(over 200 papers contributing)