Some ancillary functions and other tricks for implementing Attribute-Base Encryption.

Luis J. Dominguez Perez

Cinvestav. Departamento de Computación.

ldominguez.computing.dcu.ie.

From a joint-work with Eric Zavattoni, Ana Helena Sanchez Ramirez and Francisco Rodriguez-Henriquez

"Crypto Seminar"

Mexico City.

February 3^{rd} 2012.

(refurbished)

Outline

Notation

Let p-prime > 3. Let $E(\mathbb{F}_p)$ be an elliptic curve and $r \mid \#E(\mathbb{F}_p)$. Let k be the embedding degree of E with respect to r.

Let e be a divisor of k. Let d be the minimal $\frac{k}{e}$ with $e \in [2, 3, 4, 6]$.

A twist curve E' defined over $\mathbb{F}_{p^d}$ is another elliptic curve isomorphic to E defined over $\mathbb{F}_{p^k}$.

... and let t be the trace of the Frobenius.

Discrete logarithm problem

Let $P = (x, y) \in E(\mathbb{F}_p)$ and $n \in \mathbb{Z}$, $n \geqslant 0$. Define $[n]P = P + P + \cdots + P$. The **order of the point** P is the smallest n such that $[n]P = \mathcal{O}$.

Denote $< P >$ the group generated by P. In other words,

$$< P > = \{\mathcal{O}, P, P + P, P + P + P, \ldots\}$$

Let $Q \in < P >$. Given Q, find n such that $Q = [n]P$. This is known as the **Elliptic Curve Discrete Logarithm Problem (ECDLP)**.

Known attacks affect some anomalous curves, P with a small prime order and some weak combinations of parameters.

Cinvestav

Discrete logarithm problem II

Let $\alpha \in \mathbb{F}^*_{p^n}$ and $n \in \mathbb{Z}$, $n > 0$. Define $\alpha^n = \alpha \cdot \alpha \ldots \alpha$. The **order of the element** α is the smallest n such that $\alpha^n = 1$.

Denote $< \alpha >$ the group generated by α. In other words,

$$< \alpha >= \{1, \alpha, \alpha \cdot \alpha, \alpha \cdot \alpha \cdot \alpha, \ldots\}$$

Let $\beta \in < \alpha >$. Given β, find n modulo $|\alpha|$ such that $\beta = \alpha^n$. This is known as the **The Finite Field Discrete Logarithm Problem (DLP)**.

The most efficient methods in the finite field are based on Index Calculus. The most efficient methods in elliptic curves are based on the Pollard's Rho attack.

What is a Pairing?[1]

A **Pairing** is a *bilinear map* on an *Abelian group M* taking values in some other Abelian group R,

$$\langle \cdot, \cdot \rangle : M \times M \longrightarrow R$$

There are many Abelian groups we might consider

- $\mathbb{Z}$, or more generally $\mathbb{Z}^d$
- $\mathbb{Z}/m\mathbb{Z}$, a cyclic group of order m, or generally $(\mathbb{Z}/m\mathbb{Z})^d$
- $\mathbb{F}_p$ with addition as the group law, or generally $\mathbb{F}_p^d$
- $\mathbb{F}_p^*$ with multiplication as the group law
- $E(\mathbb{F}_p)$**, the group of** $\mathbb{F}_p$**-points on an elliptic curve**
- μ_m**, the group of** m^{th}**-roots of unity**
- etc.

[1] this slide is taken from J. H. Silverman's talk in Pairing 2010

Definition of a pairing

Here, we define a **pairing** as a map: $\mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$.

These groups are finite and cyclic. $\mathbb{G}_1$ and $\mathbb{G}_2$ are additively-written and at least one is of prime order r, $\mathbb{G}_1 \subseteq E(\mathbb{F}_p)$, and $\mathbb{G}_2 \subseteq E'(\mathbb{F}_{p^d})$.

$\mathbb{G}_T$, is multiplicatively-written and of order r, $\mathbb{G}_T \subseteq \mu_r$ or just $\mathbb{F}^*_{p^k}$

Properties:

- Bilinearity
- Non-degeneracy
- Efficiently computable

Cinvestav

Jacobian Coordinate System

An elliptic curve point if represented with two coordinates (x, y) is called to be in Affine coordinates. The group law of a point in such representation requires the use of inversion of elements in a finite field, which are expensive.

Instead, a point can be represented as (X, Y, Z), where $(X, Y, Z) = (x/z^c, y/z^d)$. There is an effort when converting between representations of the same point. (see the "Explicit-Formulas Database")

Jacobian Coordinate System II

The traditional form of the curve is:

$$E : y^2 = x^3 + ax + b$$

In a projective coordinate system, the equation changes. In the case of the Jacobian coordinates ($c = 2, d = 3$), the equation of the curve is now:

$$E : Y^2 = X^3 + axZ^4 + bZ^6.$$

Cinvestav

Pairing function

We skip the details about the Miller function...

Scalar-point multiplication and exponentiation in pairings

We recall that the most important property of a pairing is the bilinearity, denoted as:

$$e([a]Q,[b]P) = e([b]Q,[a]P) = e(Q,[ab]P) = e(Q,P)^{ab}$$

where $Q \in \mathbb{G}_2$, $P \in \mathbb{G}_1$, and the result is in $\mathbb{G}_T$.

A scalar-point multiplication in $\mathbb{G}_2$ is much more expensive than in $\mathbb{G}_1$, it is wise to place such operation in the smaller group, whenever possible.

It is also know that an exponentiation in $\mathbb{G}_T$ is cheaper than a pairing computation, some protocol designers try to exploit this too.

Cinvestav

The scalar-point multiplication

The traditional method for computing the scalar-point multiplication is the Double-and-Add method.

Algorithm 1 Traditional scalar-point multiplication

Input: Positive integer k, $P \in E(\mathbb{F}_{p^m})$

Ouput: kP

```
Q ← O
l ← ⌊log_2(k)⌋
for i = l − 1 downto 0 do
    Q ← [2]Q
    if k_i = 1 then
        Q ← Q + P
    end if
end for
return Q
```

Speeding up

Generic method to speed up the exponentiation in any finite Abelian group.

- Precomputation
- Addition chains whenever the scalar is known
- Windowing techniques
- Simultaneous multiple exponentiation techniques.

Replacing the binary representation of the scalar into one with fewer non-zero terms.

Speeding up II

Elliptic curve specific methods:

- A field defined with a (pseudo-)Mersenne prime.
- Field construction using small irreducible polynomials
- Point representation with fast arithmetic
- EC with special properties.

Multinormalization

In a pairing-based protocol, the most used cryptographic primitive is the scalar-point multiplication. After a scalar-point multiplication, the output point is usually left in Jacobian coordinates.

A bunch of multiplications are performed in a typical protocol, and eventually they are used as inputs for one or several pairing functions.

If the points are expected to be in Affine coordinates, a normalization procedure is required. This operation is expensive.

Cinvestav

Multipairing

In some Pairing-Based protocols a significant number of pairing functions need to be performed.

- If the point in $\mathbb{G}_2$ is known in advance, one can precompute the values of the line function in the pairing.
- If we have several pairings to be performed AND multiplied, one can share the Squaring and the Final Exponentiation steps.
- If we have several pairings to be performed BUT used separately, we still can optimize the operations IFF they share one of the inputs of the pairing.

Cinvestav

w-NAF representation

A non-adjacent form (NAF) of a positive integer k is an expression: $k = \sum_{i=0}^{l-1} k_i 2^i$, where $k_i \in 0, \pm 1$, $k_{l-1} = 0$, and no two consecutive digits k_i are nonzero. The length of the NAF is I.

Let $w \geqslant 2$ be a positive integer. A width-w NAF of a positive integer k is also an expression $k = \sum_{i=0}^{l-1} k_i 2^i$, but where each nonzero coefficient k_i is odd, $|ki| < 2^{w-1}$, $k_{l-1} = 0$, and at most one of any w consecutive digits is nonzero. The length of the width-w NAF is l.

Example:

$$k_{i_+} = 100000300T001003000T00T0000T00T$$

w-NAF representation II

Algorithm 2 Getting the w-NAF representation of a scalar

Input: Window width w, positive integer k

Ouput: NAF$w(k)$

```
i ← 0
while k ⩾ 1 do
    if k is odd then
        k_i ← k mods 2^w, k ← k − k_i
    else
        k_i ← 0
    end if
    k ← k/2, i ← i + 1
end while
return (k_{i−1}, k_{i−2}, ..., k_1, k_0)
```

Applying the algorithm

Scalar-point multiplication using the w-NAF expansion.

Algorithm 3 w-NAF multiplication

Input: Window width w, positive integer k, $P \in E(\mathbb{F}_{p^m})$
Ouput: kP

Compute the w-NAF expansion of k
Compute $P_i = iP$ for $i \in \{1, 3, 5, \ldots 2^{w-1} - 1\}$
$Q \leftarrow \mathcal{O}$
$l \leftarrow \lfloor log_2(k) \rfloor$
for $i = l - 1$ downto 0 **do**
    $Q \leftarrow [2]Q$
    if $k_i \neq 0$ **then**
        if $k_i > 0$ **then**
            $Q \leftarrow Q + P_{k_i}$
        else
            $Q \leftarrow Q - P_{k_i}$
        end if
    end if
end for
return Q

A note in the algorithm

Please note that, if the point P is well-known in advance, and if there are plenty of memory available, a larger w-NAF value can be chosen to speed up the multiplication by precomputing in advance the step 2.

Even if the point is only known at running time, but used several times, it may be worth the cost of a large precomputation.

Point multiplication on EC w/ efficient endomorphisms

Paper:*Faster Point Multiplication on Elliptic Curves* by Gallant, Lambert and Vanstone.

The scalar-point multiplication is the additive analogue of the exponentiation operation α^k in a general (multiplicatively-written) finite group.

In other words, we can apply the same concepts in groups defined with different operations, and referring the operation simply as exponentiation in a group.

Cinvestav

Endomorphism

Let E be an elliptic curve defined over the finite field $\mathbb{F}_p$ with the point at infinity denoted by $\mathcal{O}$.

An endomorphism of E is a rational map $\phi : E \rightarrow E$ satisfying $\phi(\mathcal{O}) = \mathcal{O}$. If the rational map is defined over $\mathbb{F}_p$, then the endomorphism ϕ is also said to be defined over $\mathbb{F}_p$. In this case, ϕ is a group homomorphism of $E(\mathbb{F}_p)$, and also of $E(\mathbb{F}_{p^m})$, for all $m \geqslant 1$.

Examples

Example 1. For each $m \in \mathbb{Z}$, the *multiplication by m* map $[m] : E \rightarrow E$ defined by $P \mapsto mP$ is an endomorphism defined over $\mathbb{F}_p$.

Another case is the negation map: $P \mapsto -P$.

Example 2. The p^{th} power map $\phi : E \rightarrow E$ defined by $(x, y) \mapsto (x^p, y^p)$ and $\mathcal{O} \mapsto \mathcal{O}$ is an endomorphism defined over $\mathbb{F}_p$, called the Frobenius endomorphism.

This endomorphism is usually denoted as π, and it is normally quite fast since it is composed by a few multiplications.

Examples II

Example 3. Let $p \equiv 1 \pmod 4$ be a prime, and consider the following elliptic curve

$$E_1 : y^2 = x^3 + ax.$$

defined over $\mathbb{F}_p$. Let $\alpha \in \mathbb{F}_p$. Then, the map $\phi : E_1 \to E_1$ defined by $(x, y) \mapsto (-x, \alpha y)$ and $\mathcal{O} \mapsto \mathcal{O}$ is an endomorphism defined over $\mathbb{F}_p$.

If $P \in E(\mathbb{F}_p)$ is a point of prime order r, then ϕ acts on $\langle P \rangle$ as a multiplication map $[\lambda]$, in essence: $\phi(Q) = \lambda Q$, $\forall Q \in \langle P \rangle$, with $\lambda^2 \equiv -1 \pmod r$

Examples III

Example 3. Let $p \equiv 1 \pmod 3$ be a prime, and consider the following elliptic curve

$$E_2 : y^2 = x^3 + b.$$

defined over $\mathbb{F}_p$. Let $\beta \in \mathbb{F}_p$. Then, the map $\phi : E_2 \to E_2$ defined by $(x, y) \mapsto (\beta x, y)$ and $\mathcal{O} \mapsto \mathcal{O}$ is an endomorphism defined over $\mathbb{F}_p$.

If $P \in E(\mathbb{F}_p)$ is a point of prime order r, then ϕ acts on $\langle P \rangle$ as a multiplication map $[\lambda]$, in essence: $\phi(Q) = \lambda Q, \forall Q \in \langle P \rangle$, with $\lambda^2 + \lambda \equiv -1 \pmod r$

The method

In esence, there exists β such that:

$$[k]P = [k_0]P + [k_1]\lambda P,$$

for some k_0, k_1, and $\lambda P = (\beta x, y)$

where $\beta = -(18x^3 + 18x^2 + 9x + 2)$ for the BN curves, and x is the parameter of the curve. For the Beuchat et al. curve, β is negative, for the Aranha et al. curve, β is positive.

To get the scalar expansion, one can use the extended Euclidean algorithm.

Cinvestav

Applying the algorithm

Simultaneous scalar-point multiplication + w-NAF.

Algorithm 4 Simultaneous w-NAF multiplication

Input: Window width w, $k, l \in \mathbb{Z}$, $P, Q \in E(\mathbb{F}_{p^m})$
Ouput: $R \leftarrow kP + lQ$

Compute the w-NAF expansion of k
Compute $(P_i = iP), (Q_i = iQ)$ for $i \in \{1, 3, 5, \dots 2^{w-1} - 1\}$
$R \leftarrow \mathcal{O}$
$n \leftarrow \sup(\lfloor log_2(k) \rfloor, \lfloor log_2(l) \rfloor)$
for $i = n - 1$ downto 0 **do**
    $R \leftarrow [2]R$

    if $k_i \neq 0$ **then**
        if $k_i > 0$ **then**
            $R \leftarrow R + P_{k_i}$
        else
            $R \leftarrow R - P_{k_i}$
        end if
    end if

    if $l_i \neq 0$ **then**
        if $l_i > 0$ **then**
            $R \leftarrow R + Q_{l_i}$
        else
            $R \leftarrow R - Q_{l_i}$
        end if
    end if

end for
return Q

Introduction

Paper: *Exponentiation in pairing-friendly groups using homomorphisms* by Galbraith and Scott.

Galbraith and Scott, showed a technique for generalizing the GLV method for higher powers of the endomorphism for the groups $\mathbb{G}_2$ and $\mathbb{G}_T$.

To get an m-dimensional expansion

$$n \equiv n_0 + n_1\lambda + \cdots + n_{m-1}\lambda^{m-1} \pmod{r}$$

of $[n]P$, one must decompose n with powers of λ sufficiently different and modulo r.

Decomposition

The method solves a closest vector problem in a lattice using Babai's rounding off method. A reduced lattice basis, however, must be precomputed in order to get an efficient implementation.

For a pairing friendly elliptic curve family, it is possible to get a "natural" m-dimensional expansion with $m = \varphi(k)$, where $\varphi(k)$ is the Euler totient function on k, the embedding degree of the family [2].

[2] For the BN cuvres, $m = 4$

Decomposition II

The modular lattice basis is defined as, by:

$$L = \left\{ x \in \mathbb{Z}^m : \sum_{i=0}^{m-1} x_i \lambda^i \equiv 0 \pmod{r} \right\}$$

where $\lambda = T = t - 1$. This m-dimensional modular lattice L will be used to construct a $m \times m$ matrix. Then, one can fill the matrix with any linear combination of $\lambda : L_{i,j} \equiv 0 \pmod{r}$.

Cinvestav

The lattice

One way to get the lattice, is with the use of the LLL function, on the another hand, one can use the Weak Popov transformation of the matrice.

The matrice, however, can be represented in polynomial form, hence, one only needs to compute it once:

$$L = \begin{pmatrix} 2x & 1+x & -x & x \\ 1+x & -1-3x & -1+x & 1 \\ -1 & 2+6x & 2 & -1 \\ 2+6x & 1 & -1 & 1 \end{pmatrix}$$

This matrix is for the BN curves only.

Applying the decomposition

To apply de decomposition, we perform:

Algorithm 5 Decomposition

Input: The L matrix, the scalar $n \in \mathbb{Z}_r$
Ouput: Vector $u = (n_0, n_1, n_2, n_3)$

$w \leftarrow (n, 0, 0, 0)$
$l \leftarrow wL^{-1}$
$m \leftarrow (0, 0, 0, 0)$
for $i \leftarrow 1$ to 4 **do**
  $m_i = \lfloor l_i \rfloor$
end for
$u \leftarrow w - mL$

Applying the decomposition II

Another way to perform the decomposition is as follows:

Algorithm 6 Decomposition

Input: The L matrix, the W vector, the scalar $n \in \mathbb{Z}_r$
Ouput: Vector $u = (n_0, n_1, n_2, n_3)$

$v \leftarrow (0, 0, 0, 0)$
$u \leftarrow (0, 0, 0, 0)$
for $i \leftarrow 0$ to 4 **do**
  $v \leftarrow nW[i]/r$
end for
$u_0 \leftarrow n$
for $i \leftarrow 1$ to 4 **do**
  for $j \leftarrow 1$ to 4 **do**
    $u_i = u_i - v_j L_{j,i}$
  end for
end for

return u

where the vector for the BN cuves is:
$W = (-(6x^2 + 6x + 2), -1, x, 1 + 3x + 6x^2 + 6x^3)$.

The endomorphism

In the case of this method, the efficiently computable endomorphism is:

$$\psi^i = \phi \pi^i \psi^{-1} \tag{1}$$

where $\phi : E' \rightarrow E$ is the endomorphism used to take a point from the elliptic curve of the twist to a curve defined over an extension field (and viceversa), and π^i is the p-power Frobenius map.

In essence, the cost of ψ is about 2 multiplications in $\mathbb{F}_{p^2}$ by a constant element in $\mathbb{F}_p$, and two cheap Frobenius map applications.

More about the endomorphism

From Hess, Smart and Vercauteren, if $p \geqslant 5$, and $j(E) \in [0, 1728]$ then:

$$\phi : Aut(E) : \xi \mapsto [\xi] with [\xi](x, y) = (\xi^2 x, \xi^3 y)$$

Let π_p the p-power Frobenius map on E.

Then $\psi = \phi^{-1}\pi_p\phi$ is an endomorphism of E' s.t. $\psi : G_2 \rightarrow G_2$.

For $Q \in G_2$, $\psi^k(Q) = Q$, $\psi(Q) = pQ$, and $\Phi_k(\psi)(Q) = \mathcal{O}$, where $\Phi_k(x)$ is the k-th cyclotomic polynomial.

even more about the endomorphism

ψ is an endomorphism from $E' \to E'$ which fixes the point at infinity. Hence, ψ is an endomorphism on E'.

Let $Q \in E'(\mathbb{F}_{p^d})[r]$. Then, $\phi(Q) \in E(\mathbb{F}_{p^k})$ and, since the image of $E'(\mathbb{F}_{p^d})[r]$ under ϕ does lie in the eigenspace of the p-power Frobenius map on $E(\mathbb{F}_{p^k})$ with eigenvalue p (the characteristic of the base field), $\pi_p(\phi(Q)) = p\phi(Q)$. Hence, $Q' = \pi(\phi(Q))$ does lies in the image of $E'(\mathbb{F}_{p^d})$ under ϕ, and so $Q'' = \phi^{-1}(Q') \in E'(\mathbb{F}_{p^d})$.

Then, $\psi^k = \phi^{-1}\pi_p^k\phi = \phi^{-1}\pi_{p^k}\phi$. Since $\pi_p^k = 1$ on $E(\mathbb{F}_{p^k})$ it follows that $\psi^k(Q) = Q$. Recalling $\pi_p(\phi(Q)) = p\phi(Q)$, so...

yes, even more about the endomorphism

$$\psi = \phi^{-1}\pi_p\phi(Q) = \phi^{-1}p\phi(Q) = pQ.$$

Since $Q[r]$, and $r|\Phi_k(p)$, it follows that $\Phi(\psi)(Q) = \Phi_k(p)Q = \mathcal{O}$.

Now, we proceed to show the scalar-point multiplication algorithm.

The algorithm

Algorithm 7 Multi w-NAF multiplication

Input: Window width w, positive integer matrix k of dimmension $n \times l$ (n vectors of l-bits), $P \in E(\mathbb{F}_{p^m})$

Ouput: kP

Compute the w-NAF expansion of each scalar in k
Compute $P_i^n = iP^n$ for $i \in \{1, 3, 5, \ldots 2^{w-1} - 1\}$
$Q \leftarrow \mathcal{O}$
for $i = l - 1$ downto 0 **do**
    $Q \leftarrow [2]Q$
    for $j = 0 \rightarrow n - 1$ **do**
        if $k_i^j > 0$ **then**
            $Q \leftarrow Q + P^j_{k_i^n}$
        else
            $Q \leftarrow Q - P^j_{k_i^n}$
        end if
    end for
end for
return Q

GS on G_T

In the case of this group, the efficiently computable endomorphism is the Frobenius endomorphism, this is because:

$$e^p \equiv e^{t-1} \equiv e^r$$

Hence,

$$e^k = e^{k_0} \cdot e^{k_1^p} \cdot e^{k_2^{p^2}} \cdots e^{k_1^{p^{m-1}}}$$

where $e \in G_T$, $k \in \mathbb{Z}_r$, m is the degree of the decomposition, and the exponentiation to the p is done using the Frobenius endomorphism.

We can use the same method for decomposing the exponent (using square-and-multiply for this case), and applying the corresponding endomorphism (the Frobenius exponentiation).

Cinvestav

GS decomposition in $\mathbb{G}_1$

The decomposition method can also be applied to the GLV method.

The corresponding matrices are:

$$L_{GLV} = \begin{pmatrix} 2x + 6x^2 & -1 - 2x \\ -1 - 2x & 1 + 4x + 6x^2 \end{pmatrix}$$

And $W_{GLV} = (2 + 4x + 6x^2, -1 - 2x)$. Obviously, algorithms 5 and 6 would need to be adapted to match the matrix and vector dimension.

Multipairing

As it was the case of the scalar-point multiplication and exponentiation, we can share the squaring part. This is the only part that needs to be computed regardless of the input values.

In the case of a negative x-parameter, there is an inversion in $\mathbb{F}_{p^{12}}$, which again is independent of the input points.

First, let's analize the basic pairing function.

Pairing algorithm/Multipairing

Basic Miller loop + final exponentiation

Input: $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$
Ouput: $f \in \mathbb{G}_T$

$f \leftarrow 1$, $T \leftarrow P$, $i \leftarrow \lfloor \mathrm{Log}_2(r) \rfloor - 1$
while $i \geqslant 0$ **do**
  $f \leftarrow f^2 \cdot L_{T,T}(Q)$
  $T \leftarrow 2T$
  if $s_i[i+1] = 1$ **then**
    $f \leftarrow f \cdot L_{T,P}(Q)$
    $T \leftarrow T + P$
  end if
  $i \leftarrow i - 1$
end while
$f^{(p^k-1)/r}$
return f

Pairing algorithm/Multipairing

Basic Miller loop + final exponentiation

Input: $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$
Ouput: $f \in \mathbb{G}_T$

$f \leftarrow 1$, $T \leftarrow P$, $i \leftarrow \lfloor \mathrm{Log}_2(r) \rfloor - 1$
while $i \geqslant 0$ **do**
  $f \leftarrow f^2 \cdot L_{T,T}(Q)$
  $T \leftarrow 2T$
  if $s_i[i+1] = 1$ **then**
    $f \leftarrow f \cdot L_{T,P}(Q)$
    $T \leftarrow T + P$
  end if
  $i \leftarrow i - 1$
end while
$f^{(p^k-1)/r}$
return f

Pairing algorithm/Multipairing

Basic Miller loop + final exponentiation

Input: $[P_1 \dots P_n]$ with $P_j \in \mathbb{G}_1$, $[Q_1 \dots Q_n]$ with $Q_j \in \mathbb{G}_2$
Ouput: $f \in \mathbb{G}_T$

$f \leftarrow 1$, $T_j \leftarrow P_j$, $i \leftarrow \lfloor \mathrm{Log}_2(r) \rfloor - 1$
while $i \geqslant 0$ **do**
  $f \leftarrow f^2 \cdot L_{T_j,T_j}(Q)$
  $T_j \leftarrow 2T_j$
  if $s_i[i+1] = 1$ **then**
    $f \leftarrow f \cdot L_{T_j,P_j}(Q)$
    $T_j \leftarrow T_j + P_j$
  end if
  $i \leftarrow i - 1$
end while
$f^{(p^k-1)/r}$
return f

Pairing algorithm/Multipairing II

The used Miller loop + final exponentiation

Input: $[P_1 \dots P_n]$ with $P_j \in \mathbb{G}_1$, $[Q_1 \dots Q_n]$ with $Q_j \in \mathbb{G}_2$
Ouput: $f \in \mathbb{G}_T$

$f \leftarrow 1,\ T_j \leftarrow Q_j,\ s \leftarrow |6x+2|$
for $i \leftarrow 2$ to $\leftarrow \lfloor \mathrm{Log}_2(s) \rfloor$ **do**
  $f \leftarrow f^2 \cdot L_{T_j,T_j}(P)$
  $T_j \leftarrow 2T_j$
  if $s_i[i] = 1$ **then**
    $f \leftarrow f \cdot L_{T_j,Q_j}(P_j)$
    $T_j \leftarrow T_j + P_j$
  end if
  $i \leftarrow i - 1$
end for
$f \leftarrow f^{p^6}$
$R \leftarrow \phi(Q_j)$ $f \leftarrow f \cdot L_{-T_j,R_j}(P_j)$
$R \leftarrow \phi^2(Q_j)$ $f \leftarrow f \cdot L_{-T_j,-R_j}(P_j)$
$f^{(p^k-1)/r}$
return f

Pairing algorithm/Multipairing II

The used Miller loop + final exponentiation

Input: $[P_1 \dots P_n]$ with $P_j \in \mathbb{G}_1$, $[Q_1 \dots Q_n]$ with $Q_j \in \mathbb{G}_2$
Ouput: $f \in \mathbb{G}_T$

$f \leftarrow 1,\ T_j \leftarrow Q_j,\ s \leftarrow |6x + 2|$
for $i \leftarrow 2$ to $\leftarrow \lfloor \mathrm{Log}_2(s) \rfloor$ **do**
$f \leftarrow f^2 \cdot L_{T_j,T_j}(P)$
$T_j \leftarrow 2T_j$
if $s_i[i] = 1$ **then**
$f \leftarrow f \cdot L_{T_j,Q_j}(P_j)$
$T_j \leftarrow T_j + P_j$
end if
$i \leftarrow i - 1$
end for
$f \leftarrow f^{p^6}$
$R \leftarrow \phi(Q_j)\ f \leftarrow f \cdot L_{-T_j,R_j}(P_j)$
$R \leftarrow \phi^2(Q_j)\ f \leftarrow f \cdot L_{-T_j,-R_j}(P_j)$
$f^{(p^k-1)/r}$
return f

Pairing algorithm/Multipairing II

The used Miller loop + final exponentiation

Input: $[P_1 \dots P_n]$ with $P_j \in \mathbb{G}_1$, $[Q_1 \dots Q_n]$ with $Q_j \in \mathbb{G}_2$
Ouput: $f \in \mathbb{G}_T$

$f \leftarrow 1,\ T_j \leftarrow Q_j,\ s \leftarrow |6x+2|$
for $i \leftarrow 2$ to $\leftarrow \lfloor \mathrm{Log}_2(s) \rfloor$ **do**
  $f \leftarrow f^2 \cdot L_{T_j,T_j}(P)$
  $T_j \leftarrow 2T_j$
  if $s_i[i] = 1$ **then**
    $f \leftarrow f \cdot L_{T_j,Q_j}(P_j)$
    $T_j \leftarrow T_j + P_j$
  end if
  $i \leftarrow i - 1$
end for
$f \leftarrow f^{p^6}$
$R \leftarrow \phi(Q_j)$ $f \leftarrow f \cdot L_{-T_j,R_j}(P_j)$
$R \leftarrow \phi^2(Q_j)$ $f \leftarrow f \cdot L_{-T_j,-R_j}(P_j)$
$f^{(p^k-1)/r}$
return f

Multinormalize

In a multipairing, a lot of set of points are used as the input, must of them, come after a Scalar-point multiplication, which means, they are in Jacobian coordinates.

Converting from Jacobian coordinates to Affine, implies a division (one for each coordinate).

We can optimize this by converting the whole set of coordinates at once, using simultaneous Montgomery inversion, which uses the following "trick":

$$bc/abc = 1/a \quad ac/abc = 1/b \quad ab/abc = 1/c$$

Using a Intel Core i7 2600K, Sandy Bridge.

Using LSSS_U repository

Operation	Clock cycles
RegularPairing	2108 Kclk
Multipairing 8	8410 Kclk
Pairing w/Precom.	1790 Kclk
G1mul K	232.89Kclk
G1mul U	304.44Kclk
G2mul K	378.26Kclk
G2mul U	535.69Kclk
GTexpo K	617.32Kclk
GTexpo U	931.98Kclk

Cinvestav

More timings

Using LSSS_U repository

Operation	Clock cycles
G1 Add JJA	1.92Kclk
G1 Add JJJ	2.44Kclk
G1 Dbl A	1.20Kclk
G1 Dbl J	1.44Kclk
G2 Add JJA	5.11Kclk
G2 Add JJJ	6.70Kclk
G2 Dbl A	3.03Kclk
G2 Dbl J	2.92Kclk
GT Sqr	3.78Kclk
GT Mul	9.55Kclk

More to come...

The Karabina's compressed squaring is only useful when the exponent has not only a low-Hamming weight, but also has plenty of zeros in a row between the poles of the scalar.

In our scenario, we need one exponentiation in $\mathbb{G}_T$ with a known base. Since it is a known value, we precomputed it with w-NAF=7, hence, we expect to have a significant speed up by using the Karabina's exponentiation against the Granger-Scott method, although we have not yet designed the algorithm.