Computing the Matrix of Frobenius for $E[3]$

Simon Spicer
0939537
Math 581B/D Project
Spring 2010, University of Washington

December 16, 2010

Abstract

In the world of elliptic curves, computing the trace of Frobenius is an important step in determining the size of a curve E over a finite field, as well as helping determine the modular curve attached to E. In this project we develop an algorithm in Sage for explicitly computing the matrix representing the action of Frobenius on the ℓ-torsion of an elliptic curve E over the rationals for the case of $\ell = 3$.

This algorithm is parameterized by the choice of basis of $E[3]$, the 3-torsion elements over E, as well as the choice of reduction of E to the finite field case. We develop a practical way of choosing such parameters, so that all computations can be carried out in feasible time.

1 Problem Outline and Motivation

For the course of this project we fix the following:

- E, an elliptic curve over the rationals;
- p, a prime of good reduction for E;
- ℓ, a small prime $\neq 2$ (the second half of this project will consider an algorithm for the case $\ell = 3$, however the theory discussed initially holds for any odd prime ℓ);
- $K = \mathbb{Q}(E[\ell])$, the number field obtained by adjoining all the coordinates of the ℓ-torsion elements in $E(\overline{\mathbb{Q}})$.

Note that K is a Galois extension of $\mathbb{Q}$, and $E(K)$ contains $E[\ell] = \{P \in E(\overline{\mathbb{Q}}) : \ell P = \mathcal{O}\}$.

We wish to make sense of the mod-ℓ Galois representation $\rho_\ell : \mathrm{Gal}(K/\mathbb{Q}) \to \mathrm{GL}_2(\mathbb{F}_\ell)$. This project specifically considers the element $\mathrm{Frob}_\mathfrak{p} \in \mathrm{Gal}(K/\mathbb{Q})$, the Frobenius element with respect to p; we wish to compute the 2×2 matrix of its image $\rho_\ell(\mathrm{Frob}_\mathfrak{p}) \in \mathrm{GL}_2(\mathbb{F}_\ell)$.

Why is this important? Recall that the group of points of E over the finite field of size p has cardinality 'approximately equal to $p + 1$'. Specifically:

Theorem 1.1. *If we let $a_p = p + 1 - \#E(\mathbb{F}_p)$, then $|a_p| \leq 2\sqrt{p}$ and $a_p \equiv \mathrm{tr}(\rho_\ell(\mathrm{Frob}_\mathfrak{p})) \pmod{\ell}$.*

Thus computing the trace of the matrix of $\mathrm{Frob}_\mathfrak{p}$ in $\mathrm{GL}_2(\mathbb{F}_\ell)$ allows us to find the value of a_p modulo ℓ. Doing this for a number of small ℓ gives us sufficient information to reconstruct a_p in its entirety via the Chinese Remainder Theorem, and hence determine the size of $E(\mathbb{F}_p)$. This is in fact the tactic used by Schoof's Algorithm to determine the cardinality of groups of elliptic curves over finite fields, an algorithm that is integral to the success of elliptic curve cryptography.

However, Schoof's Algorithm uses a number of shortcuts to obtain $\mathrm{tr}(\rho_\ell(\mathrm{Frob}_\mathfrak{p}))$ without computing the matrix $\rho_\ell(\mathrm{Frob}_\mathfrak{p})$ directly. In this project we provide an algorithm in Sage for computing $\rho_\ell(\mathrm{Frob}_\mathfrak{p})$ in its entirety in a meaningful way. We do this by parameterizing two morphisms whose composition is equal to ρ_l on $\mathrm{Frob}_\mathfrak{p}$, allowing us to compute the Frobenius matrix with respect to these parameters.

2 Specifying Morphisms

Recall that $E[\ell] \approx \mathbb{Z}/\ell\mathbb{Z} \times \mathbb{Z}/\ell\mathbb{Z} \simeq (\mathbb{F}_\ell)^2$. To determine the image of Frobenius in $\mathrm{GL}_2(\mathbb{F}_\ell)$, we must nail down the representation

$$\rho_\ell : \mathrm{Gal}(K/\mathbb{Q}) \to \mathrm{GL}_2(\mathbb{F}_\ell),$$

where ρ_ℓ is defined as the action of $\mathrm{Gal}(K/\mathbb{Q})$ on the ℓ-torsion subgroup $E[\ell]$, interpreted as a two-dimensional F_ℓ-vector space.

However, specifying ρ_ℓ in its entirety constitutes doing more work than is necessary, since we are only interested in the image of the Frobenius element. Thus we will consider the restricted map

$$\rho_\ell|_{\langle \mathrm{Frob}_\mathfrak{p} \rangle} : \langle \mathrm{Frob}_\mathfrak{p} \rangle \to \mathrm{GL}_2(\mathbb{F}_\ell).$$

For ease of notation let $\phi_\ell = \rho_\ell|_{\langle \mathrm{Frob}_\mathfrak{p} \rangle}$.

Theorem 2.1. *For prime $\mathfrak{p}$ lying above p in K, ϕ_ℓ factors through the Galois group of $\mathcal{O}_K/(\mathfrak{p})$ over $\mathbb{F}_p$, where the map $\langle \mathrm{Frob}_\mathfrak{p} \rangle \to \mathrm{Gal}(\frac{\mathcal{O}_K}{\mathfrak{p}}/\mathbb{F}_p)$ is just the canonical map induced by reducing elements in K modulo $\mathfrak{p}$.*

Recall that for $\mathfrak{p}$ a prime lying over p in K, then $\mathcal{O}_K/(\mathfrak{p}) \simeq \mathbb{F}_{p^m}$ for some $m \in \mathbb{N}$. Furthermore, by the definition of Frobenius endomorphism, Frobenius acting on fields of characteristic p is just the map $x \mapsto x^p$. Hence we can compute $\rho_\ell(\mathrm{Frob}_\mathfrak{p})$ by computing the action of the map $(x, y) \to (x^p, y^p)$ in the reduced ℓ-torsion group $\bar{E}[\ell] \subset E(\mathcal{O}_K/(\mathfrak{p}))$.

This raises two issues: to compute $\phi_\ell(\mathrm{Frob}_\mathfrak{p})$ we must specify the prime $\mathfrak{p}$ lying above p with which we factor in order to get to the finite field $(\mathcal{O}_K/(\mathfrak{p}))$. There might be many primes lying over p, and we will get a different presentation of the finite field $\mathbb{F}_{p^m}$ – and hence potentially a different matrix of Frobenius in $\mathrm{GL}_2(\mathbb{F}_l)$ – for each $\mathfrak{p}$.

Secondly, the final matrix depends on the non-canonical isomorphism $E[\ell] \approx (\mathbb{F}_\ell)^2$ i.e., on a choice of basis $(P, Q) \subset E[\ell]$. Again, different choices of basis will result in different matrices of Frobenius.

Therefore the representation $\phi_\ell : \langle \mathrm{Frob}_\mathfrak{p} \rangle \to \mathrm{GL}_2(\mathbb{F}_\ell)$ is parameterized by the choice of $\mathfrak{p}$ lying over p, and the choice of basis $(P, Q) \subset E[\ell]$.

3 The Naïve Approach

Here is an algorithm for computing the matrix $\phi_\ell(\mathrm{Frob}_\mathfrak{p})$ that can be implemented quite easily in Sage. Given elliptic curve E, a prime p of good reduction over E, and a small odd prime ℓ:

1. Precomputation:
   - Compute all elements in $E[\ell] \subset E(\overline{\mathbb{Q}})$.
   - Construct $K = \mathbb{Q}(E[\ell])$, which is de facto Galois.
   - Factor the ideal (p) over K.
2. Specifying choices:
   - Choose a basis $(P, Q) \subset E[\ell]$.
   - choose a prime $\mathfrak{p}$ lying over p.
3. Computing the Frobenius matrix with respect to these choices:
   - Compute finite field $\mathcal{O}_K/(\mathfrak{p})$.
   - Compute the reductions $\overline{P}, \overline{Q}$ via the quotient map $\alpha \mapsto \overline{\alpha}$, where α is the generator of K, and $\overline{\alpha}$ that of $\mathcal{O}_K/(\mathfrak{p})$.
   - Compute $\mathrm{Frob}_\mathfrak{p}(\overline{P})$ and $\mathrm{Frob}_\mathfrak{p}(\overline{Q})$ via $(x, y) \to (x^p, y^p)$.
   - Express the above as linear combinations of $\overline{P}$ and $\overline{Q}$,
     i.e. $\mathrm{Frob}_\ell(\overline{P}) = a \cdot \overline{P} + b \cdot \overline{Q}$ and $\mathrm{Frob}_\ell(\overline{Q}) = c \cdot \overline{P} + d \cdot \overline{Q}$ for some $a, b, c, d \in \mathbb{F}_3$.
   - The matrix representing the action of Frobenius is then $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

However, this approach isn't practical: we are stymied at the outset. Even for $\ell = 3$, $K = \mathbb{Q}(E[\ell])$ is an extension over $\mathbb{Q}$ of degree dividing 48. Thus in most cases performing operations like factoring polynomials over K to compute $E[\ell]$ explicitly takes an inordinate amount of time. We will need to be a bit more clever than this in order compute Frobenius matrices in practical amounts of time.

4 A Better Idea

In this section we describe an algorithm developed in conjunction with William Stein in November 2010 in the attempt to circumvent the ornery issue of computing all of $E[\ell]$. The algorithm effectively halves the degree of the extension over $\mathbb{Q}$ that needs to be computed, allowing us to efficiently compute Frobenius matrices for the case $\ell = 3$.

Thus from here on we consider the specific case $\ell = 3$, for which the algorithm demonstrably works in practice (see the code at the end of the project).

4.1 A Choice of Basis

We have mentioned that $\mathbb{Q}(E[3])$ is typically a degree 48 extension over $\mathbb{Q}$. However, observe that $\mathbb{Q}(x(E[3]))$, the field obtained by adjoining just the x-coordinates of the points in $E[3]$ to $\mathbb{Q}$, is an extension of degree at most 24. This is because the 3-division polynomial for E is always of degree 4; hence adjoining all the roots of this polynomial to $\mathbb{Q}$ results in a Galois extension of degree dividing 24.

Suppose then that instead of finding a basis for $E[3]$ via computing $\mathbb{Q}(E[3])$, we instead compute xP and xQ, the x-coordinates of two points in $E[3]$, which lie in the smaller extension $\mathbb{Q}(x(E[3]))$. Halving the degree of the extension means this will take much less time.

Do (xP, xQ) define a unique choice of basis for $E[3]$? The answer is no: Because the Weierstrass equation for E is quadratic in y, each x-value satisfying said equation is associated with two y-values. For example, if (α, β) is a point on the curve defined by $y^2 = x^3 + ax + b$, then so is $(\alpha, -\beta)$). Thus given xP and xQ, the choices of $(P, Q), (-P, -Q), (-P, Q)$ and $(P, -Q)$ will all be bases of $E[3]$.

How do we differentiate between the choices that have exactly the same (xP, xQ)?

4.2 The Issue of Sign

Suppose we have picked a basis $(P, Q) \equiv (\left(\begin{smallmatrix}1\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right))$ for $E[3]$, and we now compute the action of Frobenius $= \left(\begin{smallmatrix}a & b\\c & d\end{smallmatrix}\right)$ with respect to this basis. What would the matrix of Frobenius look like if we had instead picked the basis to be $(-P, -Q)$? Take a moment to convince yourself that the matrix would be the same. This is because Frobenius acting on $E[3]$ is a linear function:

$$\mathrm{Frob}_{\mathfrak{p}}(P) = aP + bQ \Rightarrow \mathrm{Frob}_{\mathfrak{p}}(-P) = -\mathrm{Frob}_{\mathfrak{p}}(P) = -(aP + bQ) = a(-P) + b(-Q).$$

The result is that our choice of (P, Q) or $(-P, -Q)$ for a basis of $E[3]$ won't affect the output matrix. Similarly, a choice of $(-P, Q)$ would yield the same matrix as $(P, -Q)$.

However, it is *not* necessarily the case that (P, Q) and $(P, -Q)$ will produce the same Frobenius matrix, so we still need some way of differentiating our choice of basis up to the sign of one of the basis elements. This can be done using the Weil pairing mentioned in the introduction.

4.3 The Weil Pairing

Let E be an elliptic curve defined over field K, and let $\ell \neq 2$ be a prime $\neq$ the characteristic of K. Let μ_ℓ be the multiplicative group of the ℓth roots of unity in $K(E[\ell])$.
The *Weil Pairing* is a bilinear form that operates on pairs of elements in $E[\ell]$ and outputs an ℓth root of unity in $K(E[\ell])$.
That is, $\langle \cdot, \cdot \rangle : E[\ell] \times E[\ell] \to \mu_\ell$, and for $P, Q \in E[\ell]$ and $m \in \mathbb{F}_\ell$,

$$\langle mP, Q \rangle = \langle P, mQ \rangle = \langle P, Q \rangle^m.$$

Moreover, this homomorphism is surjective on μ_ℓ.

We won't go into the details of how the Weil pairing is computed in this project, as its details are not relevant here. See [4], pg. 92 for details on how the Weil pairing is defined. What matters it that $\langle P, Q \rangle$ is quick to compute, surjective on μ_ℓ, and it enables us to differentiate between choices of possible bases given the x-coordinate pair $\{xP, xQ\}$:

Suppose we have picked a basis (P, Q) for $E[\ell]$. Then by bilinearity

$$\langle P, Q \rangle = \langle -P, -Q \rangle = \langle -P, Q \rangle^{-1} = \langle P, -Q \rangle^{-1}.$$

Furthermore, one can show that if (P, Q) is a basis for $E[\ell]$, then $\langle P, Q \rangle \neq 1$.
Hence with the Weil pairing we can differentiate between choosing the bases (P, Q) or $(P, -Q)$ for $E[\ell]$.

Theorem 4.1. $K(E[\ell]) \supset K(\zeta_\ell)$. *That is, the field obtained by adjoining the ℓ-torsion coordinates of E to K contains all the ℓth roots of unity.*

In our case $\ell = 3$, so $Z_3 = (1, \zeta_3, (\zeta_3)^2)$ for primitive 3rd root of unity ζ_3; so for the basis (P, Q) we either have the Weil pairing $\langle P, Q \rangle = \zeta_3$ or $\langle P, Q \rangle = (\zeta_3)^2$.

Putting this all together, we see that, given the x-coordinates of a basis xP and xQ, specifying a primitive 3rd root of unity $\zeta_3 \in K(E[3])$ such that $\langle P, Q \rangle = \zeta_3$ completely determines the matrix of Frobenius with respect to that basis.

Note that ζ_3 is an element of $K(E[3])$, which is what we're avoiding trying to compute; so the above is not at the outset very useful to us. However, we are saved by the following conjecture:

Conjecture 4.2. $K(x(E[\ell]))$, *the field obtained by adjoining all the x-coordinates of $E[\ell]$ to K, contains $K(\zeta_\ell)$.*

This conjecture has been confirmed computationally for numerous E over $\mathbb{Q}$; it would appear to hold in general, though the proof remains for now out of reach.

Back to our algorithm. Explicitly, we can specify a primitive 3rd root of unity in $\mathbb{Q}(x(E[3]))$ by factoring the 3rd cyclotomic polynomial $x^2 + x + 1$ in $\mathbb{Q}(x(E[3]))$ and choosing one of the roots. Furthermore, by the above lemma we see that choosing such a root in $\mathbb{Q}(x(E[3]))$is equivalent to specifying a primitive 3rd root of unity in $\mathbb{Q}(E[3])$.

4.4 The Choice of Quotient Map

If we ditch computing $\mathbb{Q}(E[3])$ in favour of the smaller exension $K = \mathbb{Q}(x(E[3]))$, can we still specify a quotient map $K \to \mathbb{F}_{p^m}$ in a meaningful way? Yes; the reduction w.r.t $\mathfrak{p}$ $\mathbb{Q}(x(E[3])) \to \mathbb{Q}(\mathbb{F}_{p^m})$ is still valid. However, $\mathfrak{p}$ is now a prime lying above p in $K = \mathbb{Q}(x(E[3]))$, not $\mathbb{Q}(E[3])$.

Thus we could do as follows:

- Compute $K = \mathbb{Q}(x(E[3]))$;
- Factor (p) over K and choose $\mathfrak{p}$ lying above p;
- Reduce $\mathcal{O}_K$ modulo $\mathfrak{p}$ to obtain the finite field $\mathbb{F}_{p^m}$ for some m.

However, we can introduce a shortcut which will save on computation time. By the primitive element theorem, $K = \mathbb{Q}(\alpha)$ for some algebraic number α. Let f be the defining polynomial of α, i.e. $f(x) \in \mathbb{Z}[x]$ irreducible and $f(\alpha) = 0$. Recall that $\mathbb{Z}[\alpha]$ is a subring of $\mathcal{O}_K$.

Theorem 4.3. *If p is coprime to $[\mathcal{O}_K : \mathbb{Z}[\alpha]]$, then $\mathcal{O}_K/\mathfrak{p} \approx \mathbb{F}_p[x]/(\overline{g})$, where $\overline{g}$ is one of the irreducible factors of $\overline{f}$, the reduction of f modulo p.*

Proof. Given in [1], pp. 52-53. □

Since we are typically choosing a large p, this will almost always be the case. However, care must be taken for p small, in which case the above assertion might fail.

So for our purposes, specifying a prime $\mathfrak{p}$ above p is equivalent to specifying an irreducible factor $\overline{g}$ of the reduced polynomial $\overline{f} \in \mathbb{F}_p[x]$. Hence in our algorithm, instead of specifying a prime above p, we can instead specify a factor of $\overline{f}$.

4.5 The Algorithm

Using this neat trick involving Weil pairings, we can reduce the problem of computing the matrix of Frobenius to one involving an extension of degree no more than 24 over $\mathbb{Q}$.
Computing in a degree 24 extensions over $\mathbb{Q}$ in Sage turns out to be completely feasible on a personal computer, allowing us to give the workable algorithm below.

Given elliptic curve E and a prime p of good reduction over E:

1. Precomputation:
   - Compute $x(E[3])$, the set of x-coordinates of all elements in $E[3]$.
   - Construct $K = \mathbb{Q}(x(E[3]))$. K is then a Galois extension of degree dividing 24 over $\mathbb{Q}$.
   - Compute f, the defining polynomial of K.
   - Factor $x^2 + x + 1$, the 3rd cyclotomic polynomial over K.
   - Compute $\overline{f}$, the reduction of f modulo p, and factor $\overline{f}$ over $\mathbb{F}_p$.
2. Specifying choices:
   - Pick $xP, xQ \in x(E[3])$, representing the x-coordinates of our basis.
   - Pick ζ, one of the roots of $x^2 + x + 1$ in K.
   - Pick $\overline{g}$, one of the factors of $\overline{f}$. This corresponds to picking a prime $\mathfrak{p}$ lying over p in K.
3. Computing the Frobenius matrix with respect to these choices:
   - If m is the degree of $\overline{g}$, define $k = \mathbb{F}_{p^m}$, where $\mathbb{F}_{l^m}$ is constructed to have modulus $\overline{g}$. Then k is exactly the field $\mathcal{O}_K/(\mathfrak{p})$, where $\mathfrak{p}$ corresponds to $\overline{g}$.
   - Compute $\overline{xP}, \overline{xQ}$ and $\overline{\zeta}$, the reductions of xP, xQ and ζ in k respectively.
   - The reduced elements of $E[3]$ all lie within $E(k')$, where k' is at most a quadratic extension of k; so compute k' if necessary. There are two (3-torsion) elements in $E(k')$ with x-coordinate equal to $\overline{xP}$; choose $\overline{P}$ to be one of them. Similarly, obtain one such $\overline{Q}$.
   - Compute the Weil pairing $< \overline{P}, \overline{Q} >$. This is either equal to $\overline{\zeta}$ or $(\overline{\zeta})^{-1}$.
   - If $< \overline{P}, \overline{Q} >= \overline{\zeta}$, move on to the next step. If $< \overline{P}, \overline{Q} >= \overline{\zeta}^{-1}$, set $\overline{Q} = -\overline{Q}$. We will now have $< \overline{P}, \overline{Q} >= \overline{\zeta}$.
   - Compute $\text{Frob}_{\mathfrak{p}}(\overline{P})$ and $\text{Frob}_{\mathfrak{p}}(\overline{Q})$ via $(x, y) \to (x^p, y^p)$.
   - Express the above as linear combinations of $\overline{P}$ and $\overline{Q}$, i.e. $\text{Frob}_\ell(\overline{P}) = a \cdot \overline{P} + b \cdot \overline{Q}$ and $\text{Frob}_\ell(\overline{Q}) = c \cdot \overline{P} + d \cdot \overline{Q}$ for some $a, b, c, d \in \mathbb{F}_3$.
   - The matrix representing the action of Frobenius is then $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

5 Implementation

Below is the Sage code for some helper functions that we will need:

```
# Computes the action of Frobenius for elements of an elliptic curve over a finite field
def frobenius(P):
    """
    If P = (x,y) is defined over field of characteristic P,
    returns the point (x^p, y^p).

    EXAMPLES::

        sage: F = EllipticCurve(GF(3001^3,'a'), [-5,9]); F
        Elliptic Curve defined by y^2 = x^3 + 2996*x + 9 over Finite Field in a
        of size 3001^3
        sage: P = F.an_element(); P
        (1637*a^2 + 1297*a + 1392 : 1394*a^2 + 2454*a + 1142 : 1)
        sage: frobenius(P)
        (1187*a^2 + 1221*a + 792 : 2582*a^2 + 611*a + 2726 : 1)

    """

    if P.is_zero():
        return P
    E = P.curve()
    x, y = P.xy()
    return E(x.frobenius(),y.frobenius())


# Double discrete log
def ddl(R, r, basis):
    """
    For R in E[r] with basis (P, Q), returns (a,b), where R = a*P + b*Q.

    EXAMPLES::

        sage: F = EllipticCurve(GF(10009^2,'a'), [3,7]); F
        Elliptic Curve defined by y^2 = x^3 + 3*x + 7 over Finite Field in a of
        size 10009^2
        sage: F(0).division_points(3)
        [(0 : 1 : 0), (1760 : 4880*a + 249 : 1), (1760 : 5129*a + 9760 : 1),
        (448 : 2911*a + 4187 : 1), (448 : 7098*a + 5822 : 1), (8614 : 342*a +
        9325 : 1), (8614 : 9667*a + 684 : 1), (9196 : 4314*a + 1381 : 1), (9196
        : 5695*a + 8628 : 1)]
        sage: P = F(0).division_points(3)[1]
        sage: Q = F(0).division_points(3)[3]
        sage: ddl(P + 2*Q, 3, (P,Q))
        (1,2)
        sage: ddl(2*P, 3, (P,Q))
        (2,0)

    """

    P, Q = basis[0], basis[1]
    a, b = 0, 0
    while R != 0:
        R = R - P
        a += 1
        if a == r:
            R = R - Q
            a = 0
            b += 1
            if b == r: raise ValueError("Basis not basis.")
    return (a, b)
```

```
# Compute Frobenius matrix with given two-element basis
def frob_matrix(r, basis):
    """
    If basis = (P,Q) for E[r], returns the matrix [[a,c],[b,d]],
    where frobenius(P) = a*P + b*Q, and frobenius(Q) = c*P + d*Q

    EXAMPLES::

        sage: F = EllipticCurve(GF(10009^2,'a'), [3,7]); F
        Elliptic Curve defined by y^2 = x^3 + 3*x + 7 over Finite Field in a of
        size 10009^2
        sage: P = F(0).division_points(3)[1]
        sage: Q = F(0).division_points(3)[3]
        sage: frob_matrix(3, (P,Q))
        [2 0]
        [0 2]

    """

    row1 = ddl(frobenius(basis[0]), r, basis)
    row2 = ddl(frobenius(basis[1]), r, basis)
    return matrix(Integers(r),[row1, row2]).transpose()
```

And the code for the `matrix_of_frobenius` function:

```
# Compute the matrix of Frobenius
def matrix_of_Frobenius(E, p, x_root_choices, zeta_choice, gbar_choice):
    """
    Returns the matrix representing the action of Frobenius w.r.t p on E[3],
    given the choices specified by x_root_choices, zeta_choice and gbar_choice.

    INPUT:

    -  ``E`` - The elliptic curve for which the Frobenius matrix is being computed
    -  ``p`` - A prime of good reduction for E whose Frobenius element we are considering
    -  ``x_root_choices`` - A tuple (a1, a2) of a pair of integers between 0 and 3
       representing our choice of the two x-values of the r-division polynomial
       which constitute the x-values of our basis
    -  ``zeta_choice`` - An integer either 0 or 1, representing which of the two
       primitive third roots of unity we are choosing
    -  ``gbar_choice`` - An integer between 0 and some divisor of 24 representing our
       choice of irreducible divisor of the defining polynomial of QQ(xE[r]) reduced
       modulo p

    OUTPUT:

    -  An invertible 2x2 matrix over the finite field GF(3).

    The returned matrix will always have the same trace in GF(3), regardless of the
    values specified in x_root_choices, zeta_choice and gbar_choice.

    EXAMPLES::

        sage: E = EllipticCurve([-5,9]); E
        Elliptic Curve defined by y^2 = x^3 - 5*x + 9 over Rational Field
        sage: p = 3001
        sage: x_root_choices = (0,1)
        sage: zeta_choice = 0
        sage: gbar_choice = 0
        sage: matrix_of_Frobenius(E, p, x_root_choices, zeta_choice, gbar_choice)
        [1 2]
```

```
        [1 0]
        sage: zeta_choice = 1; gbar_choice = 4
        sage: matrix_of_Frobenius(E, p, x_root_choices, zeta_choice, gbar_choice)
        [2 0]
        [1 2]
        sage: x_root_choices = (3,1)
        sage: matrix_of_Frobenius(E, p, x_root_choices, zeta_choice, gbar_choice)
        [2 0]
        [2 2]

        sage: zeta_choice = 2
        sage: matrix_of_Frobenius(E, r, x_root_choices, zeta_choice, gbar_choice)
        Traceback (most recent call last)
        ...
        IndexError: list index out of range
        sage: zeta_choice = 1; x_root_choices = (2,4)
        sage: matrix_of_Frobenius(E, r, x_root_choices, zeta_choice, gbar_choice)
        Traceback (most recent call last)
        ...
        IndexError: list index out of range
        sage: x_root_choices = (3,1); gbar_choice = 13
        sage: matrix_of_Frobenius(E, r, x_root_choices, zeta_choice, gbar_choice)
        Traceback (most recent call last)
        ...
        IndexError: list index out of range
    """

    # Compute the 3-division polynomial for E and make it monic, so that we can construct
    # a Number field with it
    h = E.division_polynomial(3)
    h = sage.schemes.elliptic_curves.heegner.make_monic(h)[0]

    # Construct K, the number field containing all the roots of h
    factor_list = []
    for j in h.factor():
        factor_list.append(j[0])
    variables = ['a1','a2','a3','a4']
    variables = variables[:len(factor_list)]
    K = NumberFieldTower(factor_list,variables)
    K.<a> = K.galois_closure()

    # Compute the x-values of our basis corresponding to the values in x_root_choices
    root_list = K[x](E.division_polynomial(3)).roots()
    xP = root_list[x_root_choices[0]][0]
    xQ = root_list[x_root_choices[1]][0]

    # Compute the primitive 3rd root of unity correspoding to the value specified
    # in zeta_choice
    z = K[x](x^2 + x + 1)
    zeta = z.roots()[zeta_choice][0]

    # Compute the irreducible factor of f reduced modulo p, corresponding to the
    # value specified in gbar_choice
    f = K.defining_polynomial()
    fbar = GF(p)[x](f)
    gbar = fbar.factor()[gbar_choice][0]

    # Construct the finite field k = GF(p)[x]/(gbar), and reduce xP, xQ and zeta into
    # this field
    k.<abar> = GF(p^(gbar.degree()),modulus=gbar)
    F = E.change_ring(k)
    xPbar = k(xP.polynomial())
    xQbar = k(xQ.polynomial())
    zetabar = k(zeta.polynomial())
```

```
    # Construct E over k
    F = E.change_ring(k)

    # Look for points in F corresponding to xPbar and xQbar. If none exist in E(k),
    # construct a quadratic extension k' of k for which corresponding points in
    # E(k') do exist. Pick two respective points arbitrarily.
    try:
        Pbar = F.lift_x(xPbar)
        Qbar = F.lift_x(xQbar)
    except ValueError:
        # No points exist. The following is a hack to construct a quadratic extension
        # of k such that the finite field is of the right type
        R = GF(p)[x]
        hbar = R(x^2)
        lbar = gbar(hbar)
        while not lbar.is_irreducible():
            hbar = hbar + R(x)
            lbar = gbar(hbar)
        k.<abar> = GF(p^(lbar.degree()),modulus=lbar)

        # Coerce xPbar, xQbar and zetabar into the new k and try lift again
        xPbar = k(xPbar.polynomial()(hbar))
        xQbar = k(xQbar.polynomial()(hbar))
        zetabar = k(zetabar.polynomial()(hbar))
        F = E.change_ring(k)
        Pbar = F.lift_x(xPbar)
        Qbar = F.lift_x(xQbar)

    # The Weil Pairing: if <Pbar, Qbar> != zetabar, we have chosen the wrong Qbar;
    # So instead set Qbar to -Qbar
    if -(Pbar._miller_(Qbar,r)/Qbar._miller_(Pbar,3)) != zetabar: Qbar = -Qbar

    # Finally, return the matrix of Frobenius acting on the basis (Pbar, Qbar)
    return frob_matrix(3, (Pbar, Qbar))
```

References

[1] W. Stein, *Algebraic Number Theory: a Computational Approach,*
Published free online at
`http://wstein.org/books/ant/`

[2] D. Husemöller, *Elliptic Curves,*
Graduate Texts in Mathematics
Springer, New York, 2004.

[3] H Cohen & G Frey, *Handbook of Elliptic Curve Cryptography,*
Discrete Mathematics and its Applications
Chapman & Hall/CRC, Boca Raton, 2006.

[4] J. H. Silverman, *The Arithmetic of Elliptic Curves,*
Graduate Texts in Mathematics
Springer, New York, 2009.