Elliptic Curve Cryptography

ECMM705: Mathematical Sciences Project III

Jack Ronald Percival Hanslope

The University of Exeter

Supervisor: Gihan Marasingha

Sunday 14th April 2019

Abstract

This paper will give an introduction to Elliptic curves before moving on to explore some applications of elliptic curves to modern cryptography. This will include an application to cryptanalysis, a key-agreement protocol for use in symmetric key encryption and methods for asymmetric key encryption. The appendices contain some coded implementations of the discussed algorithms, written in SageMath.

Contents

1 Introduction to the project

The area of cryptography is one of great interest to mathematicians; modern cryptography underpins so much of our everyday lives. One of the more recent advances in cryptography has been in the area of elliptic curves. This paper aims to give an introduction to elliptic curves before exploring some of the different applications of they have to cryptography.

What follows is a brief overview of how this paper is laid out. In section 2 we introduce elliptic curves over finite fields, the group law for the points on

an elliptic curve and explicit values for elliptic curve point addition. Section 3 first discusses one of the most common integer factorisation methods, Pollard's $p-1$ method, before outlining one of the method's drawbacks. It then goes on to outline an algorithm for factorising integers using elliptic curves, following Neal Koblitz's 1994 work, *A course in number theory and cryptography* ([Kob94]). Said algorithm shares some similarities with Pollard's method, but avoids the main drawback. Section 4 introduces the reader to the Diffie-Hellman key exchange (first introduced in [DH76]), the Diffie-Hellman problem and the ElGamal public-key cryptosystem (of [ElG85]) before discussing the analogues of these using elliptic curves. Section 5 considers the current area of open research: bilinear pairings, following a lot of the work done by Alfred Menezes in his 2009 paper [Men09]. We first discus some motivation for bilinear pairings, before introducing the pairings themselves and some protocols using said pairings. We go on to introduce the Tate pairing, including Miller's algorithm for computing the pairing, and how we can adapt the pairing for use. We discus how one can choose the curve they are working over to provide a good level of security whilst still allowing computations to be feasible, using the work [FST10] of Freeman, Scott and Teske. Finally, we provide an explicit example of a three-party, one-round key-agreement protocol, introduced by Antoine Joux in his 2004 paper [Jou04]. In the appendices we present some scripts for implementing some of the protocols discussed in the paper. The scripts are written in SageMath, an open-source computer algebra system.

2 Some basic properties of Elliptic Curves

We define an **elliptic curve** E over a field K by a non-singular Weierstrass equation

$$y^2 + a_1xy + a_3yx^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

with $a_1, a_2, a_3, a_4, a_5, a_6 \in K$. When we refer to the elliptic curve, we are usually referring to the set of K-rational points, $E(K)$. $E(K)$ is the set of points $(x, y) \in K \times K$ satisfying (2.1) along with the **point at infinity**, ∞. Suppose the characteristic of K is p. If $p \notin \{2, 3\}$, we can transform an equation of the form (2.1) into an equation of the form

$$y^2 = x^3 + ax + b \tag{2.2}$$

where $a, b \in K$ and $4a^3 + 27b^2 \neq 0$, using a linear change of variables. If $p = 2$, then we can transform (2.2) into an equation of the form

$$y^2 + cx = X^3 + ax + b \qquad \text{or} \qquad y^2 + xy = x^3 + ax^2 + b.$$

If $p = 3$, then we can transform an equation of the form (2.1) into an equation of the form

$$y^2 = x^3 + ax^2 + bx + c$$

where the cubic on the right hand side has no repeated roots. For the majority of this project, we will be considering fields K for which $\text{char}(K) > 3$ and equations of the form (2.2).

2.1 Group structure of an elliptic curve

A useful property of the points on an elliptic curve is that they, with the following definition of addition and inverses and the point at infinity as the identity, form an abelian group.

Definition 2.1. Let E be an elliptic curve defined over some field K of characteristic not equal to 2 or 3 by an equation of the form (2.2). Let P and Q be two distinct points on the elliptic curve, neither equal to the point at infinity, ∞. We define inverses and addition as follows, with ∞ serving as the identity.

1. If $P = (x, y)$, let $-P = (x, -y)$. If $K = \mathbb{R}$, then $-P$ will be the point when P is reflected in the x-axis. From the equation (2.2), we can see that $-P$ is guaranteed to be on the curve. We define $P + -P = \infty$.
2. Suppose P and Q have different x coordinates, then the line through P and Q will intersect the curve at exactly on other point, R say. We define $P + Q = -R$
3. To find $P + P = 2P$, consider the tangent through P. This will intersect the elliptic curve at exactly one more point (which may be the point at infinity), R say. Then $2P = -R$. If P is a point of inflection, then R is taken to be P.

This definition of addition and inverses enables us to form the set of K-rational points of E into a group.

Proposition 2.1. *Let E and K be as in definition (2.1). Then the set of K-rational points of E, $E(K)$ forms an abelian group with ∞ as the identity and addition as in definition (2.1).*

Proof. Omitted. The references for §VI.1 of [Kob94] give some examples of where this proof can be found. □

2.2 Explicit values for addition

We will show that addition of elliptic curve points in the manner descirbed in the previous section is well-defined and in doing so, derive formulas for the point addition. We will follow closely Example 1 of §VI.1 of [Kob94]. Let $q > 3$, E be

an elliptic curve defined over a finite field $\mathbb{F}_q$ by equation $y^2 = x^3 + ax + b$ and points P and Q be on the elliptic curve with $R = P + Q$. Let (x_P, y_P),(x_Q, y_Q) and (x_R, y_R) be the coordinates of P, Q and R respectively. We are aiming to express x_R and y_R in terms of x_P, y_P, x_Q, y_Q . First suppose that $P \neq Q$ and that $x_P \neq x_Q$ so that the line ℓ through P and Q is not vertical. Let $y = \alpha x + \beta$ be the equation of this ℓ. We have that

$$\alpha = \frac{y_Q - y_P}{x_Q - x_P} \qquad \text{and} \qquad \beta = y_P - \alpha x_P$$

A point $T = (x, \alpha x + \beta)$ lies on the line ℓ if and only if

$$(\alpha x + \beta)^2 = x^3 + ax + b.$$

So each point of intersection of the line ℓ with the curve E will correspond to one root of the equation

$$x^3 - (\alpha x + \beta)^2 + ax + b = 0$$

We know there are roots at $(x_P, \alpha x_P + \beta)$ and $(x_Q, \alpha x_Q + \beta)$. Then, by Vieta's formulas, we deduce that $x_R = \alpha^2 - x_P - x_Q$ and then using $y_R = -(\alpha x_R + \beta)$ (the minus sign coming in because R is reflected in the x-axis from the point of intersection)

$$\begin{aligned} x_R &= \left(\frac{y_Q - y_P}{x_Q - x_P}\right)^2 - x_P - x_Q \\ y_R &= -y_P + \left(\frac{y_Q - y_P}{x_Q - x_P}\right)(x_P - x_R). \end{aligned} \tag{2.3}$$

Suppose $Q = P$, we must now take α as the derivative dy/dx and the point P. Differentiating the equation $y^2 = x^3 + ax + b$ gives

$$\begin{aligned} 2y\frac{dy}{dx} &= 3x^2 + a \\ \frac{dy}{dx} &= \frac{3x^2 + a}{2y} \\ \alpha &= \frac{3x_P^2 + a}{2y_P} \end{aligned}$$

Thus for $R = 2P$ we get

$$\begin{aligned} x_R &= \left(\frac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P \\ y_R &= -y_P + \left(\frac{3x_P^2 + a}{2y_P}\right)(x_P - x_R). \end{aligned} \tag{2.4}$$

3 Elliptic Curve Factorisation

For this section, we follow [Kob94]. One of the reasons mathematicians are most interested in elliptic curves is that they have provided a new mechanism for factoring large integers. This method, pioneered by Lenstra, does not provide enough of an improvement on existing methods to pose a threat to cryptographic systems whose security is built on the supposed intractability of factoring. However, this unexpected development does show that we cannot be too complacent with the supposed security of modern cryptography.

We first give a definition of smooth and powersmooth before outlining the Pollard $p-1$ method; finally we introduce Lenstra's method.

Definition 3.1. Let $B > 0$. An integer $n > 1$ is said to be B-**smooth** if every prime factor of n is less than or equal to B. n is said to be B-**powersmooth** if, for any prime p and integer α such that $p^\alpha \mid n$, $p^\alpha \leq B$.

3.1 Pollard's $p-1$ method

We wish to find a (as yet unknown) prime factor p of some large composite number n. If $p-1$ has only small prime divisors then Pollard's method will usually find p; specifically, the method will usually be successful if n is B-powersmooth. We follow the algorithm as outlined in [Mar19].

Algorithm 3.1. *Given a composite number n and a bound B.*

1. *Let $p_1 < p_2 < \ldots < p_t$ be the primes less than or equal to B. Set $a \leftarrow 2$.*
2. *For all $i = 1, \ldots t$, find the largest integer α_i satisfying $P_i^{\alpha_i} \leq B$. Set*

$$M \leftarrow \prod_{i=1}^{t} p_i^{\alpha_i}$$

3. *Compute $g \leftarrow \gcd(a^M - 1, n)$.*
4. *If $1 < g < n$, then g is a non-trivial factor of n. If $g \in \{1, n\}$ then the algorithm has failed.*

We now give an example.

Example 3.1. Suppose we would like to factor the integer $n = 49153$ using Lenstra's method with the bound $B = 5$. The prime powers at or below B are 2^2, 3 and 5. We set $M = 2^2 \times 3 \times 5 = 60$. We compute $2^{60} \pmod{49153} = 25803$ and then compute $\gcd(2^M - 1, n) = \gcd(35802, 49153) = 13$ so $13 \mid 49153$.

In the previous example, we found prime divisor $p = 13$, giving $p - 1 = 12$. The only prime divisors of 12 are 2 and 3 which are both small. Now we consider an example where the bound B would have to be very high for the algorithm to be successful.

Example 3.2. Consider $n = 620497$. It turns out that $n = 719 \times 863$ and $719 - 1 = 2 \times 359$ and $863 - 1 = 2 \times 431$ where both 359 and 431 are prime. We would therefore have to use a value of $B \geq 359$ before the algorithm was successful.

3.2 Lenstra's Method

In this section we will follow [Kob94] and [Mar19]. When we run Pollard's method, we are hoping that the various groups $\mathbb{Z}_p^*$ given for primes $p \mid n$ will enable us to find a divisior. When we fix n, these groups are fixed too; if they all happen to have orders divisible by only large primes, then Pollard's method will take a long time. The method described below, is called the **Lenstra elliptic-curve factorisation** method, it being named after Hendrik Lenstra.

We execute the method by constucting an elliptic curve over the ring $\mathbb{Z}_n$. As $\mathbb{Z}_n$ is not a field for composite n, there will be some elements of $\mathbb{Z}_n$ that do not have multiplicative inverses. Multiplicative inverses are required when computing point addition, so the set of points $E(\mathbb{Z}_n)$ is not closed and hence, not a group. We will make use of this fact by attempting to compute point addition. An element $a \in \mathbb{Z}_n$ will have a multiplicative inverse if and only if $\gcd(a, n) = 1$ so when point addition fails, if $1 < \gcd(a, n) < n$, we will have found a non-trivial factor of n.

Given a composite number n with (as yet unknown) prime factor p, a rough outline of the method is

1. Select an elliptic curve E over $\mathbb{Z}/n\mathbb{Z}$ of the form (2.2) and a point on the curve P.
2. Set bound B and then set k as the product of all prime powers p_i less than or equal to B.
3. Attempt to calculate kP by first calculating $p_1 P$ and then $p_1^2 P$ and so on. If at any point, the intermediary calculation fails, we will have either found a non-trivial divisor of n or must start again with a new curve and point pair.

Algorithm 3.2 (Lenstra's algorithm)**.** *We would like to find a non-trivial divisor of some integer n given a bound B.*

1. *Check that n is not:*

- *divisible by* 2 *or* 3
- *prime*
- *a perfect power*

If it is any of these, we will have found a non-trivial factor. (All of these can be tested for in relatively little time when compared to the time for the whole Lenstra algorithm).

2. *Let* $p_1 < p_2 < \dots p_t$ *be the primes less than or equal to* B.
3. *For each* $i = 1, \dots, t$ *find the largest integer* α_i *such that* $P_i^{\alpha_i} \leq B$. *Set*

$$k \leftarrow \prod_{i=1}^{t} p_i^{\alpha_i}$$

4. *For each* i *in*

$$\left\{p_1, p_1^2, p_1^3, \dots, p_1^{\alpha_{p_1}}, p_2 \cdot p_1^{\alpha_{p_1}}, p_2^2 \cdot_1^{\alpha_{p_1}}, \dots, p_2^{\alpha_{p_2}} p_1^{\alpha_{p_1}}, \dots, k\right\}$$

attempt to find iP, *at each stage, multiplying the previous result by the next factor. At each attempt, if there is a failure and the result cannot be found, we will be able to find a non-trivial divisor of* n *by taking a gcd. If* kP *is successfully found, we must find a new curve-point pair* (E, P).

We will now consider a toy example.

Example 3.3. Suppose we would like to find a non-trivial factor of the integer $n = 77$ using the bound $B = 5$. We perform some quick checks to ensure n is not prime, a prime power, or a multiple of 2 or 3. We then compute $k = 60 = 2{\cdot}2{\cdot}3{\cdot}5$ and randomly generate an elliptic curve E over $\mathbb{Z}_{77}$ given by

$$y^2 = x^3 + 52x + 49$$

with point P with coordinates $(8, 47)$. We compute $2P$ as $(44, 71)$ using (2.4). We then attempt to compute $2(2P)$ however, when we try to find the inverse of $14 \pmod{77}$ we find that $\gcd(14, 77) = 7$ and so 7 is a non-trivial factor of 77.

4 Diffie-Hellman and ElGamal

In this section, we will consider some adaptations we can make to the Diffie-Hellman key-exchange and the ElGamal public key cryptosystem.

4.1 Diffie-Hellman

One issue with symmetric encryption, is that the parties that wish to communicate securely must first agree on a shared secret key. If an eavesdropper

intecepted this secret key, then the encryption would be broken; the Diffie-Hellman key exchange was one of the first solutions to this problem. We outline the algorithm below.

Algorithm 4.1 (Two-party Diffie-Hellman key agreement)**.** *Alice and Bob would like to exchange a shared secret.*

1. *Alice (or Trent) generates a finite cyclic, multiplicatively written group $G = \langle g \rangle$ of order n. She then chooses a secret $a \in \mathbb{Z}_n^*$. Alice publishes (G, g, g^a).*
2. *Bob selects a secret $b \in \mathbb{Z}_n^*$ and publishes g^b.*
3. *Alice computes $s_a \leftarrow (g^b)^a$ using her secret key a and Bob's public key g^b.*
4. *Bob computes $s_b \leftarrow (g^a)^b$ using his secret key b and Alice's public key g^a.*

Alice and Bob will now have the shared secret key $(g^a)^b = g^{ab} = (g^b)^a$. Suppose an eavesdropper, Eve, has access to all of Alice and Bob's communications and wants to recover the secret key g^{ab}. Eve has G, g, g^a, g^b, and wants to compute g^{ab}; this is known as the Diffie-Hellman problem (DHP). The assumed intractability of the DHP is the basis for the security of the Diffie-Hellman key-exchange. We will show that the DHP reduces in polynomial time, to the discrete logarithm problem (DLP). The DLP in a multiplicatively-written group G is the problem of, given $a, b \in G$, finding some $x \in \mathbb{Z}$ such that $a^x = b$. We often use the multiplicative group of a finite field $\mathbb{F}_q^*$ as the group G. Saying the DHP reduces in polynomial time to the DLP means that there is an algorithm which solves the DHP using an algorithm which solves the DLP and that this algorithm runs in polynomial time if the algorithm for the DLP does. To see that the DHP reduces in polynomial time to the DLP, suppose we are given g, g^a and g^b and we wish to find g^{ab}, we use an oracle to solve the DLP to find a then compute $g^{ab} = (g^b)^a$. We outline an example below.

Example 4.1. Suppose Trent sets $G = \mathbb{Z}_{23}^*$ with $g = 5$. Alice chooses her private key to be $a = 17$ and calculates and publishes

$$g^a = 5^{17} \equiv 15 \pmod{23}.$$

Bob chooses his public key to be $b = 20$ and publises

$$g^b = 5^{20} \equiv 12 \pmod{23}.$$

Alice then computes

$$(g^b)^a = 12^{17} \equiv 9 \pmod{23}$$

and Bob computes

$$(g^a)^b = 15^{20} \equiv 9 \pmod{23}$$

and so Alice and Bob have the shared secret 9. An adversary Eve would have to solve the DLP to recover either a or b and then g^{ab} .

We can also execute the Diffie-Hellman key exchange over an additively written group. The algorithm for which is below.

Algorithm 4.2 (Two-party Diffie-Hellman key agreement for a additive group)**.** *Alice and Bob would like to exchange a shared secret.*

1. *Alice (or Trent) generates a finite cyclic additively written group $G = \langle P \rangle$ of order n. She then chooses a secret $a \in \mathbb{Z}_n^*$. Alice publishes (G, P, aP).*
2. *Bob selects a secret $b \in \mathbb{Z}_n^*$ and publishes bP.*
3. *Alice computes $s_a \leftarrow a(bP)$ by using her secret key a and Bob's public bP.*
4. *Bob computes $s_b \leftarrow b(aP)$ by using his secret key b and Alice's public aP.*

Alice and Bob will now have the shared secret $a(bP) = b(aP)$. An eavesdropper has the information aP and bP and wishes to recover $abP = baP$; this is another example of the Diffie-Hellman problem (DHP). The DLP in an additivley-written group $G = \langle P \rangle$ whose order is n is the problem of, given $P, Q \in G$, finding $x \in \mathbb{Z}_n$ such that $Q = xP$. The DLP is intractable for certain groups.

We can use the group of points on an elliptic curve as our additive cyclic group G. We demonstrate a toy example below with an elliptic curve over a prime field; we could of course, use an elliptic curve over a prime power field.

Example 4.2. Suppose Trent picks the Elliptic Curve E over the field $\mathbb{F}_{73}$ given by equation

$$y^2 = x^3 + 4x + 69$$

and base point $B = (51, 8)$. The group $G = E(K)$ is of order 78. Suppose Alice chooses her private key $a = 71$ she then calculates her public key to be $71P = (34, 69)$ and publishes this. Suppose then that Bob chooses his private key to be $b = 45$ and publishes his public key $45P = (13, 37)$. Alice then computes the shared secret as

$$a \times 45P = 71 \times (13, 37) = (31, 45)$$

and Bob computes the shared secret as

$$b \times 71P = 45 \times (34, 69) = (31, 45).$$

4.2 ElGamal

The Diffie-Hellman key exchange is not used for sending secure communications, but sharing a secret key (which will often then be used for symmetric encryption). However, it can quite easily be adapted to the ElGamal public-key cryptosystem which can be used for sending secure communications, the algorithm for which is below and is reproduced from [Mar19].

Algorithm 4.3. *Bob wishes to send a secret message to Alice.*

1. *[Key generation]*
   (a) *Alice (or Trent) selects a finite, cyclic (multiplicatively written) group G and a generator g of the group. She publishes (G, g, n) where $n \leftarrow |G|$.*
   (b) *Alice chooses and publishes a secret key $a \in \mathbb{Z}_n^*$.*
2. *[Encryption]*
   (a) *Bob wishes to encrypt the plaintext message $m \in G$.*
   (b) *Bob chooses an ephemeral key $k \in \mathbb{Z}_n^*$. Bob computes $c_1 \leftarrow g^k$.*
   (c) *Bob computes $c_2 \leftarrow (g^a)^k m$. Bob publishes (c_1, c_2).*
3. *[Decryption]*
   (a) *Alice computes $m \leftarrow c_1^{-a} \cdot c_2$*

Again, we can use an additive group rather than a multiplicative group (with some alterations of course) and this additive group could be the group of points on an elliptic curve. The algorithm below gives a method for the ElGamal public key cryptosystem using the group of points on an elliptic curve over a finite field.

Algorithm 4.4. *Bob wishes to send a secret message to Alice.*

1. *[Key generation]*
   (a) *Alice (or Trent) selects a finite field $\mathbb{F}_q$ where q is a prime power. For simplicity, we assume the characteristic of the field is greater than 3. She then chooses an elliptic curve E over the field and a base point B on the curve.*
   (b) *Alice chooses a secret key $a \in \mathbb{Z}$.*
   (c) *Alice computes and publishes her public key aB.*
2. *[Encryption]*
   (a) *Suppose Bob wishes to send $P_m \in E(K)$. He chooses an ephemeral key $k \in \mathbb{Z}$ and computes kB.*
   (b) *He computes $P_m + k(aB)$ and sends $(kB, P_m + k(aB))$ to Alice.*
3. *[Decryption]*
   (a) *Alice recovers the plaintext message P_m as*

$$P_m + k(aB) - a(kB) = P_m$$

We will amend the example from before slightly to use the ElGamal PKC.

Example 4.3. Again, we have the elliptic curve E defined by

$$y^2 = x^3 + 4x + 69$$

defined over $\mathbb{F}_{73}$ with base point $B = (51, 8)$. Alice's private key is $a = 71$ and her public key is $aB = (34, 69)$. Suppose Bob wishes to send the secret message $P_m = (68, 56)$. Bob chooses an ephemeral key $k = 13$ and computes $kB = (24, 22)$. He then computes

$$P_m + k(aB) = (68, 56) + 13 \times (34, 69) = (22, 72)$$

Bob sends $((24, 22), (22, 72))$ to Alice. Alice computes P_m as

$$(22, 72) - 71 \times (24, 22) = (68, 56)$$

as expected.

5 Bilinear Pairings

One area of open research in crypotgraphy is that of bilinear pairings. We will discus some of their applications in the following section.

5.1 Motivation

The Diffie-Hellman key-exchange, as introduced in the previous chapter can be easily extended to three (or more) parties, however, two-rounds (or more) are then required. The algorithm is outlined below.

Algorithm 5.1 (Three-party Diffie-Hellman key agreement)**.** *Alice, Bob and Charlie would like to exchange a shared secret.*

1. *Trent (or Alice) generates a finite cyclic group $G = \langle P \rangle$ of order n and publishes (G, P).*
2. *Alice, Bob and Charlie each choose a secret key $a, b, c \in \mathbb{Z}_n^*$ respectively.*
3. *Alice, Bob and Charlie each calculate aP, bP, cP respectively.*
4. *Alice sends aP to Bob, Bob sends bP to Charlie and Charlie sends cP to Alice.*
5. *Alice calculates $a(cP)$, Bob calculates $b(aP)$ and Charlie calculates $c(bP)$.*
6. *Alice sends $a(cP)$ to Bob, Bob sends $b(aP)$ to Charlie and Charlie sends $c(bP)$ to Alice.*
7. *Alice calculates $a(cbP)$, Bob calculates $b(acP)$ and Charlie calculates $c(baP)$.*

All three parties will now have the shared secret $acbP = bacP = cbaP$; an eavesdropper would have to compute $abcP$ given $P, aP, bP, cP, abP, bcP, caP$. As previously mentioned, this protocol requires two-rounds of communication;

we would prefer to only use one-round. A one-round, three-party protocol for exchanging a shared secret is one of the applications of bilinear pairings that we will discus in this section.

5.2 Bilinear pairings

We follow [Men09]. For some prime number n, an additively written group $G_1 = \langle P \rangle$ of order n with identity ∞ and a multiplicitivly-written group G_T of order n with identitiy 1 we have the following definition.

Definition 5.1. A **bilinear pairing** on (G_1, G_T) is a map

$$\hat{e} : G_1 \times G_1 \to G_T$$

satisfying

1. (bilinearity) For all $R, S, T \in G_1$ we have

$$\hat{e}(R+S,T) = \hat{e}(R,T)\hat{e}(S,T), \text{ and}$$
$$\hat{e}(R,S+T) = \hat{e}(R,S)\hat{e}(R,T).$$

2. (non-degeneracy) $\hat{e}(P,P) \neq 1$
3. (computability) $\hat{e}$ can be efficiently computed

We have the following proposition on the properties of bilinear pairings.

Proposition 5.1. *For all $S, T \in G_1$:*

1. $\hat{e}(S,\infty) = 1$ *and* $\hat{e}(\infty,S) = 1$
2. $\hat{e}(S,-T) = \hat{e}(-S,T) = \hat{e}(S,T)^{-1}$
3. $\forall a, b \in \mathbb{Z},\ \hat{e}(aS,bT) = \hat{e}(S,T)^{ab}$
4. $\hat{e}(S,T) = \hat{e}(T,S)$
5. *If, for all* $R \in G_1$, $\hat{e}(S,R) = 1$ *then* $S = \infty$

Proof. Suppose that we have G_1 and G_T as in the definition, $S, T \in G_1$ and $\hat{e}$ a bilinear pairing on (G_1, G_T).

1. We will show that $\hat{e}(S,\infty) = 1$ and then $\hat{e}(\infty,S) = 1$ will follow once we have proved 4.

$$\hat{e}(S,\infty) = \hat{e}(s,\infty+\infty)$$
$$\hat{e}(S,\infty) = \hat{e}(S,\infty)\hat{e}(S,\infty)$$
$$\Rightarrow 1 = \hat{e}(S,\infty)$$

2. Observe that

$$\begin{aligned}\hat{e}(S,-T)\hat{e}(S,T) &= \hat{e}(S,-T+T)\\ &= \hat{e}(S,\infty)\\ &= 1\\ \Rightarrow \hat{e}(S,-T) &= \hat{e}(S,T)^{-1}\end{aligned}$$

   We can show similarly that $\hat{e}(-S,T) = \hat{e}((S,T)^{-1})$ by using $\hat{e}(\infty,T) = 1$

3. We have that

$$\begin{aligned}\hat{e}(aS,bT) &= \hat{e}(\underbrace{S+S+\ldots S}_{a \text{ times}},bT)\\ &= \hat{e}(S,bT)^a\\ &= \hat{e}(S,\underbrace{T+T+\ldots T}_{b \text{ times}})^a\\ &= \left(\underbrace{\hat{e}(S,T)\hat{e}(S,T)\ldots\hat{e}(S,T)}_{b \text{ times}}\right)^a\\ \hat{e}(aS,bT) &= \hat{e}(S,T)^{ab}\end{aligned}$$

4. Note that, since G_1 is cyclic, it is commutative (a known property of groups), and S,T can be expressed in terms of P. Let $S = sP$ and $T = tP$. Then we have

$$\begin{aligned}\hat{e}(S,T) &= \hat{e}(sP,tP)\\ &= \hat{e}(P,P)^{st}\\ &= \hat{e}(P,P)^{ts}\\ &= \hat{e}(tP,sP)\\ &= \hat{e}(T,S)\end{aligned}$$

5. We will instead show that the contrapositive holds, that is the following:

$$\infty \neq S \in G_1 \Rightarrow \exists R \in G_1 : \hat{e}(S,R) \neq 1$$

   Let $S \in G_1$ such that $S \neq \infty$. Then write $S = aP$. Consider $a \in \mathbb{Z}_n$; since n is prime, $\mathbb{Z}_n$ is a field and so a has a multiplicative inverse. Call this inverse b. So $ab = xn + 1$ for some $x \in \mathbb{Z}$. Let $R = bP$ and consider

$$\begin{aligned}\hat{e}(S,R) &= \hat{e}(aP,bP)\\ &= \hat{e}(P,P)^{ab}\\ &= \hat{e}(P,P)^{xn+1}\\ &= (\hat{e}(P,P)^n)^x\hat{e}(P,P)^1\\ &= 1^x\hat{e}(P,P)\\ &= \hat{e}(P,P)\\ \hat{e}(S,R) &\neq 1\end{aligned}$$

□

We now define the bilinear Diffie-Hellman problem and the decisional Diffie-Hellman problem.

Definition 5.2. The **bilinear Diffie-Hellman problem** (BDHP) is the problem of, given a bilinear pairing $\hat{e}$ on (G_1, G_T) and P, aP, bP, cP, computing $\hat{e}(P,P)^{abc}$.

Definition 5.3. The **decisional Diffie-Hellman problem** (DDHP) in G_1 is the problem of deciding whether a given quadruple (P, aP, bP, cP) of elements of G_1 is a valid Diffie-Hellman quadruple. That is whether $cP = abP$.

Note that the DDHP can be efficiently solved. Let $\gamma_1 = \hat{e}(P, cP) = \hat{e}(P,P)^c$ and $\gamma_2 = \hat{e}(aP, bP) = \hat{e}(P,P)ab$. We have that $cP = abP$ if and only if $\gamma_1 = \gamma_2$

5.3 Protocols

In this section, we will introduce some protocols which can be used with billinear pairings.

5.3.1 Joux's key agreement

In [Jou04], Joux proposed a one-round three-party key agreement which was then modified by Verheul in [Ver04]; here we breifly outline Verheul's adaptation.

Algorithm 5.2 (Joux's protocol)**.** *Three parties, Alice, Bob and Charlie would like to exchange a shared secret with only one round of communications.*

1. *Trent generates, for some prime number n, an additively written group $G_1 = \langle P \rangle$ of order n, a multiplicatively written group G_T also of order n and a bilinear pairing $\hat{e}$ on (G_1, G_T). He publishes $(G_1, G_T, \hat{e})$.*
2. *Alice, Bob and Charlie each choose a secret key $a, b, c \in \mathbb{Z}_n^*$, respectively.*
3. *Alice, Bob and Charlie each publish aP, bP, cP respectively.*
4. *Alice computes $K = \hat{e}(bP, cP)^a$, Bob computes $K = \hat{e}(cP, aP)^b$, and Charlie computes $K = \hat{e}(aP, bP)^c$.*

Then, all three parties will have the same shared secret $K = \hat{e}(P,P)^{abc}$. An eavesdropper will have to solve an instance of the BDHP. This protocol is not practically useful as it is not resistant to active attacks without another round of communications. We will consider an example of Joux's protocol at the end of the section.

5.3.2 Short Signatures

Cryptographic digital signature schemes are incredibly useful. All public-key cryptosystems (PKC) are vulnerable to a 'man-in-the-middle' attack. Suppose Alice wishes to send a secure message to Bob. Mallory is eavesdropping on their communications and is capable of not only interpreting their signals, but also altering the messages sent in both directions. If neither Alice nor Bob is aware of Mallory's presence, then he is able to read all (supposedly) secure communications between Alice and Bob and even alter these communications if he wishes. Digital signatures allow Alice and Bob to get around this problem.

The majority of digital signature schemes, such as the ElGamal digital signature scheme or the RSA digital signature scheme require sending a pair of integers modulo n where n is the order of some cyclic group. The Boneh-Lynn-Shacham (BLS) short signature scheme requires sending only one group element which can be represented in approximately the same number of bits as an integer modulo n.

This protocol will use a bilinear pairing $\hat{e}$ on (G_1, G_T); we require that the DHP in G_1 is intractable. We will need to use a hash function

$$H : \{0,1\}^* \to G_1 \backslash \{\infty\}$$

Alice selects a private key as some integer $a \in [1, n-1]$ and publishes her pubic key $A = aP$. Alice signs a message $m \in \{0,1\}^*$ with the group element $S = aM$ where $M = H(M)$. Bob verifies Alice's signature by computing $M = H(M)$ and verifying that (P, A, M, S) is a valid Diffie-Hellman quadruple, an instance of the DDHP as discussed above. Bob verifies $\hat{e}(P, S) = \hat{e}(A, M)$. Suppose Mallory wishes to forge Alice's signature on the message m, he needs to compute $S = aM$ given only P, A and $M = H(m)$; this is an instance of the DHP in G_1 which we assumed to be intractable.

5.4 Tate Pairing

We follow [Men09]. We have E, an elliptic curve defined over some finite field $K = \mathbb{F}_q$ by a Weierstrass equation $r(x, y) = 0$. Let $\overline{K}$ be the algebraic closure of K and use E to denote $E(\overline{K})$. The function field of E over K is the field of fractions $K(E)$ of $K[x, y]/(r(x, y))$. An automorphism σ of $\overline{K}$ over K is defined as $P^\sigma = (\sigma(x), \sigma(y))$ if $P = (x, y)$ and $\infty^\sigma = \infty$.

Definition 5.4 (Divisors)**.** For E, K as above.

1. A **divisor** on E is a formal sum of points $D = \sum_{P \in E} n_P(P)$, where the n_P are integers only finitely many of which are nonzero.

2. The **support** of D is the set of points $P \in E$ for which $n_p \neq 0$.
3. A **zero divisor** is a divisor satifying

$$\sum_{P \in E} n_P = 0.$$

4. D is said to be **defined over** K if, for all automorphisms σ of $\overline{K}$ over K, $D^\sigma = \sum_P n_P(P^\sigma) = D$.
5. The set of all divisors defined over K is denoted by $\mathrm{Div}_K(E)$
6. A **principal divisor** is divisor of a function $f \in K(E)$, defined as $\mathrm{div}(f) = \sum_{P \in E} m_P(P)$ where m_P is the multiplicity of P as a root of f.
7. For $D_1, D_2 \in \mathrm{Div}_K(E)$, we say that D_1 is euqivalent to D_2, denoted $D \sim D_2$, if there is some $f \in K(E)$ such that $D_1 = D_2 + \mathrm{div}(f)$.
8. For $f \in K(E)$ and $D = \sum_P(P) \in \mathrm{Div}_K(E)$, if $\mathrm{div}(f)$ and D have disjoint support, then

$$f(D) := \prod_{P \in E} f(P)^{n_P} \in K.$$

Note that $f(D) \neq 0$.

Theorem 5.1. *A divisor $D = \sum_{P \in E} n_P(P)$ is principal if and only if*

$$\sum_{P \in E} = 0 \qquad \textit{and} \qquad \sum_{P \in E} n_P(P) = \infty$$

Proof. Omitted. □

We now work towards defining the Tate pairing. Suppose the number of points on the curve E defined over $K = \mathbb{F}_q$, is hn where n is some prime number such that q is not a multiple of n (so $\gcd(n, q) = 1$). Now set k to be the smallest natural number satisfying $n \mid q^k - 1$. The set of ***n*-torsion points** of the elliptic curve is defined as

$$E[n] := \left\{P \in E(\overline{K}) : nP = \infty\right\}$$

We can show that $E[n] \cong \mathbb{Z}_n \oplus \mathbb{Z}_n$. We will denote the order-n subgroup of $\mathbb{F}_{q^k}^*$ by μ_n. We'd like $k > 1$ so let us assume that $n \nmid q - 1$. We will have $E[n] \subseteq E(\mathbb{F}_{q^k})$ and so $n^2 \mid \#E(\mathbb{F}_{q^k})$. Also suppose that $\gcd(n, h) = 1$ and that $n \nmid \#E(\mathbb{F}_{q^k})/n^2$.

The (modified) Tate pairing is a map

$$e : E[n] \times E[n] \to \mu_n$$

defined as follows. We take $P, Q \in E[n]$ and f_P a function such that

$$\mathrm{div}(f_P) = n(P) - n(\infty)$$

This will mean that f_P has a zero of order n at P, a pole of order n at ∞ and no other zeros or poles. The existance of this f_P is ensured by Theorem 5.1. Now let $R \in E[n] \backslash \{\infty, P, -Q, P-Q\}$ and let $D_Q = (Q+R) - (R)$. Our selection of R means that D_Q and $\mathrm{div}(f_P)$ have disjoint support. Then we define the Tate pairing by

$$e(P,Q) = f_P(D_Q)^{(q^k-1)/n} = \left(\frac{f_P(Q+R)}{f_P(R)}\right)^{(q^k-1)/n} \tag{5.1}$$

Note that the Tate pairing does not depend on the choice of f_P and R (so is well defined) and that it is bilinear and non-degenerate. The difficult part in computing the Tate pairing, is finding a function f_P satisfying the correct requirement for the divisor. Miller's algorithm describes a method for doing so.

5.4.1 Miller's Algorithm

We will describe Miller's algorithm for computing $e(P,Q)$ where $P, Q \in E[n]$ but first we discus some preliminaries. For each $i \in \mathbb{N}$ let f_i be a function whose divisor is

$$\mathrm{div}(f_i) = i(P) - (iP) - (i-1)(\infty).$$

Note that we have $f_1 = 1$ and $f_n = f_P$. The following Lemma helps us to efficiently compute f_n.

Lemma 5.1. *For $P \in E[n]$ and $i, j \in \mathbb{N}$, let ℓ be the line through iP and jP and let v be the vertical line through $iP + jP$. Then we have that*

$$f_{i+j} = f_i f_j \frac{\ell}{v} \tag{5.2}$$

Proof. The divisors of the lines ℓ and v encode the definition of the group law for E. We have

$$\begin{aligned}
\mathrm{div}\left(f_i f_j \frac{\ell}{v}\right) &= \mathrm{div}(f_i) + \mathrm{div}(f_j) + \mathrm{div}(\ell) - \mathrm{div}(v) \\
&= \{i(P) - (iP) - (i-1)(\infty)\} \\
&\quad + \{j(P) - (jP) - (j-1)(\infty)\} \\
&\quad + \{(iP) + (jP) + (-(i+j)P - 3(\infty)\} \\
&\quad - \{((i+j)P) + (-(i+j)P) - 2(\infty\} \\
&= (i+j)(P) - ((i+j)P) - (i+j-1)(\infty) \\
&= \mathrm{div}(f_{i+j})
\end{aligned}$$

□

Now we convert n to binary as $n = (n_t, \ldots, n_1, n_0)_2$. We will compute f_P with a left-to-right double-and-add method. Suppose that, after we have examinded the leftmost $t-u$ bits, we have f_m where $m = (n_t, n_{t-1}, \ldots, n_{u+1})_2$. We compute f_{2m} by setting $i = j = m$ in equation (5.2). Also, if $n_u = 1$, then we can compute f_{2m+1} by setting $i = 2m$ and $j = 1$ in equation 5.2. f_P will have been computed after $t + 1$ iterations. Note that to compute the Tate pairing, we only need the values of f_P at $Q + R$ and R so at intermediate steps, we only calculate the values of the functions f_i at these points. Miller's algorithm is the following.

Algorithm 5.3. *Given $P, Q \in E[n]$, we compute $e(P, Q)$:*

1. *Let the binary representation of n be $n = (n_t, \ldots, n_0)_2$.*
2. *Choose a point $R \in E[n] \backslash \{\infty, P, -Q, P - Q\}$.*
3. *Set $f \leftarrow 1$, $T \leftarrow P$.*
4. *for i from t down to 0:*
   (a) *Let ℓ be the tangent line through T, and let v be the vertical line through $2T$.*
   (b) *Set $T \leftarrow 2T$.*
   (c) *Set $f \leftarrow f^2 \cdot \frac{\ell(Q+R)}{v(Q+R} \cdot \frac{v(R)}{\ell(R)}$.*
   (d) *If $n_i = 1$ then*
      (i) *Let ℓ be then line through T and P, and let v be the vertical line through $T + P$.*
      (ii) *Let $T \leftarrow T + P$.*
      (iii) *Let $f \leftarrow f \cdot \frac{\ell(Q+R)}{v(Q+R)} \cdot \frac{v(R)}{\ell(R)}$*
5. *Return $(f^{q^k-1)/n}$.*

The algorithm is not always successful in theory but is usually in practice (when used in pairing based protocols). If any of the intermediate lines ℓ or v has a zero at $Q + R$ or R, then the algortithm may fail. However, in practice, we usually have $P \in E(\mathbb{F}_q)$ and $Q \notin E(\mathbb{F}_q)$, then the zeros of ℓ and v are all in $\langle P \rangle \subseteq E(\mathbb{F}_q)$ and so $R \in E[n] \backslash E(\mathbb{F}_q)$ means that ℓ and v do not have zeros at $Q + R$ or R.

5.4.2 Converting the Tate pairing into a usable pairing

Again, following [Men09]. To use the Tate pairing for pairing-based cryptography, it is required to satisfy the conditions of Definition 5.1. Although the Tate pairing, is bilinear, non-degenerate and efficiently computable, $E[n]$ is not a cyclic group of order n. We can convert the Tate pairing into a usable pairing in one of two ways, depending on the type of elliptic curve we have. We first give the definition of a supersingular curve.

Definition 5.5. An elliptic curve E over a field K is **supersingular** if the endomorphism ring of E over $\overline{K}$ has rank 4 as a $\mathbb{Z}$-module. An elliptic curve that is not supersingular is **ordinary**.

If E is supersingular and $k > 1$, then we take a point $P \in E(\mathbb{F}_q)$ of order n and an endomorphism $\Psi : E \to E$ such that $\Psi(P) \notin \langle P \rangle$. Then consider the function

$$\begin{aligned} \hat{e} : \langle P \rangle \times \langle P \rangle &\to \mu_n \\ (Q, R) &\mapsto (Q, \Psi(R)). \end{aligned}$$

This function satisfies $\hat{e}(P, P) \neq 1$ and so $\hat{e}$ is a bilinear pairing on $(\langle P \rangle, \mu_n)$ and satisfies Definition 5.1. Ψ is called a distortion map.

Again, if $k > 1$ but now E is ordinary, then we can show that there is no such distortion map. To find $\hat{e}$ in this case, we first select order-n points $P \in E(\mathbb{F}_q)$ and $Q \notin E(\mathbb{F}_q)$ and define

$$\hat{e} : \langle P \rangle \times \langle Q \rangle \to \mu_n$$

This restriction is a non-degenerate, asymmetric, billinear pairing $\hat{e} : G_1 \times G_2 \to G_T$ where $G_1 = \langle P \rangle$, $G_2 = \langle Q \rangle$ and $G_T = \mu_n$ are cyclic groups of order n. We may need to adjust the protocols given in section 5.3 to use this restriction.

5.5 Curve Selection

Thus far, we have made no discussion of how we can choose an elliptic curve for use in a cryptographic system. We can choose these randomly, but as we will see, this can lead to curves over which computations take too long and/or do not provide sufficient security. In this section, we will discus how we may choose curves that provide good security whilst still enabling us to perform computations.

When we choose our elliptic curve, we need the parameters to satisfy certain conditions in order for any cryptographic uses of the elliptic curve to be secure. These conditions are the following:

1. n needs to be large enough such that the discrete logarithm problem in an order-n subgroup of $E(\mathbb{F}_q)$ is intractable.
2. k needs to be large enough so that the discrete logarithm problem in $\mathbb{F}_{q^k}$ is intractable.
3. k needs to be small enough that arithmetic in $\mathbb{F}_{q^k}$ can be efficiently performed.

An important parameter when considering the security offered by pairing-friendly curves is

$$\rho = \frac{\log q}{\log n}.$$

This measures the size of the base field relative to the prime-order subgroup. We recreate table 1.1 of [FST10].

Security level (in bits)	Subgroup size n (in bits)	Extension field size q^k (in bits)	Embedding degree k	
			$\rho \approx 1$	$\rho \approx 2$
80	160	960 –1280	6 –8	2*, 3 –4
112	224	2200 –3600	10 –16	5 –8
128	256	3000 –5000	12 –20	6 –10
192	384	8000 –10000	20 –26	10 –13
256	512	14000 –18000	28 –36	14 –18

Table 1: Comparison of various common security levels and the subgroup sizes, extension field sizes and embedding degrees required to achieve them.

So if we wanted a security level of 256-bits, we would need to choose a prime n somewhere around 2^{512} and q^k approximately 2^{16000}.

For example, if we choose $q = 692342753$, a randomly generated curve might be

$$y^2 = x^3 + 64093880x + 47115169$$

which has a subgroup of prime order $n = 5273$. It turns out that the embedding degree is $k = 2639$ which gives q^k an approximate size (in bits) of 77500. Our subgroup order n only has a bit length of 13; we would like this bit length to be at least 160 before we have any meaningful security. Also, the embedding degree k is very large. So not only is the subgroup size too small to provide useful security, but the embedding degree is too large for the elliptic curve to be usable. On page 8 of [FST10], the authors claim that in if we choose a random elliptic curve over a random field $\mathbb{F}_q$, with subgroup of prime order $n \approx q$, we can expect the embedding degree to be around n. So if we were to increase our subgroup size here to provide useful security, we would expect the embedding degree to increase with it. What we require is a method of finding an elliptic curve with large prime subgroup sizes but small embedding degrees. We will call such curves **pairing-friendly**. Below is a formal definition of pairing-friendly as suggested in [FST10].

Definition 5.6. An elliptic curve E defined over a finite field $\mathbb{F}_q$ is said to be **pairing-friendly** if:

1. There exists some prime $n \geq \sqrt{q}$ that divides $\#(\mathbb{F}_q)$, and
2. The embedding degree k of E with respect to n satisfies $k < \log_2(n)/8$

We will consider supersingular pairing-friendly curves and ordinary pairing-friendly curves.

5.5.1 Supersingular Curves

We will not make too much discussion of supersingular elliptic curves as it has been shown that the embedding degree k for supersingular curves is $k \in \{1, 2, 3, 4, 6\}$. So if we would like to have a embedding degree greater than 6, we must use ordinary elliptic curves.

5.5.2 Ordinary Curves

Before we delve into this section, we need to define what it means for an elliptic curve to have **complex multiplication**. Suppose E is an elliptic curve defined over a field K. If the ring of $\overline{K}$-endomorphisms of E is larger than the set of integers, then we say that E has complex multiplication and is CM curve. Note that if K is a finite field, then E is guarenteed to have complex multiplication. We define, for an ordinary elliptic curve E over a finite field $\mathbb{F}_q$, the **CM discriminant** D to be minus the square-free part of $4q - t^2$ where $t = q + 1$. There are many methods for generating pairing-friendly ordinary elliptic curves which are discussed at some length in [FST10]; here we will explore the method originally suggested by Cocks and Pinch. The Cocks-Pinch method takes as input an embedding degree k, a prime n and a complex multiplication discriminant D and outputs a prime q and a trace t such that there is an elliptic curve over $\mathbb{F}_q$ that has $q + 1 - t$ points and

$$n \mid q + 1 - t \qquad \text{and} \qquad n \mid q^k - 1$$

One we have aplied the Cocks-Pinch method, we must use the **complex multiplcation mathod** of curve consturction to actually construct the curve. Said method is originally credited to Atkin and Morain in [AM93] and is not reproduced here. We outline Algorithm IX.4 of [BSS05].

Algorithm 5.4. *Given $k, n, D \in \mathbb{Z}$ such that D is square modulo n, n is prime and $k \mid n - 1$. We will output q, t such that q is prime and there is an elliptic curve over $\mathbb{F}_q$ with $q + 1 - t$ points and $n \mid q + 1 - t$ and $n \mid q^k - 1$.*

1. *Choose a primitive kth root of unity g in $\mathbb{F}_n$.*
2. *Choose an integer $a \equiv 2^{-1}(g + 1) \pmod{r}$.*
3. *If $\gcd(a, D) \neq 1$, then choose another g.*
4. *Choose an integer $b_0 \equiv \pm(a - 1)/\sqrt{D} \pmod{r}$.*
5. *Set $j \leftarrow 0$. Then until q is prime:*
   (a) *Set $q \leftarrow a^2 - D(b_0 + jr)^2$*
   (b) *Set $j \leftarrow +1$.*
6. *Set $t \leftarrow 2a$.*
7. *Return q and t.*

We will now give an example. We let $D = -7$, $k = 7$ and $n = 2437$. A quick check shows that the conditions of the algorithm are satisfied; the Legendre symbol $\left(\frac{D}{n}\right) = 1$. We find a primitive 7th root of unity by picking a random element of $\mathbb{Z}_n$ and raising it to the power of $(n-1)/k$ modulo n. Here the random element we pick is 1979 giving $g = 801$, $a = (g+1)/2 = 401$ and then $b_0 = (a-1)/\sqrt{(D)} \equiv 196 \pmod{n}$. Following the algorithm, we get that, for $j = 4$, $q = 692342753$ and $t = 802$.

We now are left with the challenge of finding an elliptic curve using the complex multiplication method. We calculate the curve to be given by

$$y^2 = x^3 + 210819370x + 271090911.$$

It turns out that this curve does not satisfy definition 5.6, however, it is significantly easier to work with than the randomly chosen curve from earlier.

5.6 An example of Joux's protocol

We will give an example of Joux's protocol using the Tate pairing. As mentioned earlier, we will have to modify the protocol slightly; the protocol is similar to one of [Jou04].

Algorithm 5.5 (Joux's key agreement with the Tate pairing)**.** *Suppose Alice, Bob and Charlie wish to exchange a shared secret with only one round of communications.*

1. *Trent chooses an Elliptic curve E over a finite field $\mathbb{F}_q$ for some prime q such that the set of points of the elliptic curve has a subgroup of prime order n. He chooses points $P \in E(\mathbb{F}_q)$ and $Q \notin E(\mathbb{F}_q)$ both of order n. Trent publishes $(\langle P\rangle, \langle Q\rangle, \hat{e})$ where $\hat{e}$ is the restricted Tate pairing $\hat{e} : \langle P\rangle \times \langle Q\rangle \to \mu_n$.*
2. *Alice, Bob and Charlie each choose a secret integer in $\mathbb{Z}_n^*$ and publish their public key pairs $(P_A, Q_A) = (aP, aQ)$, $(P_B, Q_B) = (bP, bQ)$ and $(P_C, Q_C) = (cP, cQ)$ respectively.*
3. *Each of Alice, Bob and Charlie computes the shared secret S as*

$$\begin{aligned} S &= \hat{e}(bP, cQ)^a \qquad \textit{(Alice)} \\ S &= \hat{e}(cP, aQ)^b \qquad \textit{(Bob)} \\ S &= \hat{e}(aP, bQ)^c \qquad \textit{(Charlie)} \end{aligned}$$

We will now give an example using the elliptic curve we constructed earlier with the Cocks-Pinch method.

Example 5.1. Suppose Alice, Bob and Charlie would like to exchange a shared secret. Trent chooses the elliptic curve generated earlier by the cocks-pinch method. We have E given by

$$y^2 = x^3 + 210819370x + 271090911$$

over the field $\mathbb{F}_q$ for $q = 692342753$. He selects the point P with coordinates $(565432016, 168138289)$ and point Q with coordinates

$$(77275493x^6 + 43128140x^5 + 303325271x^4 + 80803561x^3 + 505621793x^2 + 102464972x + 488677159, 556744255x^6 + 256498800x^5 + 677864764x^4 + 216963775x^3 + 304849498x^2 + 628184040x + 522278325)$$

He also specifies the Tate pairing $\hat{e} : \langle P \rangle \times \langle Q \rangle \to \mu_n$ where $n = 2437$. Suppose Alice, Bob and Charlie choose their private keys to be $a = 689$, $b = 1045$ and $c = 1978$ respectively. Alice, Bob and Charlie each compute their public key pairs (P_A, Q_A), (P_A, Q_B) and (P_A, Q_B) which are below in A.4. Alice, Bob and Charlie can each calculate the shared secret, again, we do not explicitly specify the values here, but they are in A.4. The shared secret is

$$S = 592222373x^6 + 202365451x^5 + 657132309x^4 + 463067684x^3 + 103777291x^2 + 156353336x + 590907738$$

A Code

In this appendix, we have some scripts written in SageMath.

A.1 Elliptic Curve Class

This is a script that serves only to define an ECurve class for use in pairing based cryptography. The class initiates an elliptic curve and then has functions for finding the embedding degree and prime subgroup order and giving n-torsion points $P \in E(\mathbb{F}_q)$ and $Q \notin E(\mathbb{F}_q)$.

```
1 class ECurve:
2
3     def __init__(self, q, a, b, n = None, t = None):
4         F.<tt> = GF(q)
5         self.field = F
6         self.curve = EllipticCurve(self.field, [a, b])
7         self.a = a
8         self.b = b
9         self.q = q
```

```
10          if n is not None:
11              self.n = n
12          if t is not None:
13              self.t = t
14
15      def get_prime_subgroup(self):
16          card = self.curve.cardinality()
17          prime_facs = [i[0] for i in factor(card)]
18          potential_ns = [i for i in prime_facs if gcd(
                self.q, i) == 1]
19          potential_ns = [n for n in potential_ns if (
                self.q - 1) % n != 0]
20          if potential_ns == []:
21              self.n = None
22          else:
23              self.n = potential_ns[-1]
24
25      def get_embedding_degree(self):
26          k = 2
27          while (self.q ^ k - 1) % self.n != 0:
28              k += 1
29              if k > 10000:  # an escape for when the
                    embedding degree is large
30                  print 'k > 10000'
31                  return
32          self.k = k
33
34      def get_big_curve(self):
35          if not hasattr(self, 'k'):
36              self.get_embedding_degree()
37          self.big_curve = (ECurve(self.q**self.k, self.
                a, self.b, self.n))
38
39      def give_n_torsion_point(self, restrict=False):
40          if not hasattr(self, 'n'):
41              self.get_prime_subgroup()
42          if not hasattr(self, 'k'):
43              self.get_embedding_degree()
44          if restrict:  # usually want a point from the
                bigger field
45              while True:
46                  Q = self.curve.random_element()
47                  if hasattr(self, 't'):
48                      C = self.q + 1 - self.t
49                  else:
50                      C = self.curve.cardinality()
51                  h = int(C / self.n)
52                  while  h % self.n == 0:
53                      h = int(h / self.n)
54                  Q *= h
55                  if Q != self.curve(0) and self.n * Q
                        == self.curve(0):
56                      break
57          else:
58              if not hasattr(self, 'big_curve'):
```

```
                self.get_big_curve()
            Q = self.big_curve.give_n_torsion_point(
                True)
        return(Q)

    def def_point(self, x, y):
        var('tt')
        F.<tt> = GF(self.q)
        x_, y_ = 0, 0
        for i in x:
            x_ = x_ + i[0] * tt**i[1]
        for i in y:
            y_ = y_ + i[0] * tt**i[1]
        return(self.curve(x_, y_))

    def speak(self):
        print 'We have the', self.curve
        if hasattr(self, 'n'):
            print 'n = ', self.n
        if hasattr(self, 'k'):
            print 'k = ', self.k
```

A.2 Lenstra

This is an implimentation of Lenstra's method. The first function is some basic checks before we begin the metod propper. Then we have a function for generating the bound k given a bound B and a function for generating a curve and point combination satisfying the required condtions. The final function is the iteration of choosing multiple curves until we find one that factors the required integer. We run the script to attempt to factorise 162248396707; the output of running the script is below.

```
def check_n(n):
    # check if n is prime
    if n.is_prime():
        print "{} is prime".format(n)
        quit()
    # checking if n is divisble by 2 or 3
    if n % 2 == 0:
        print "{} is even".format(n)
        quit()
    if n % 3 == 0:
        print "{} is a multiple of 3".format(n)
        quit()
    # check if n is a perfect power
    if n.is_perfect_power():
        print "{} is a perfect power".format()
        quit()


# generating the bound k given B
```

```
20  def gen_k(B):
21      arr = []
22      for p in Primes():
23          if p <= B:
24              b = 1
25              while True:
26                  if p ^ (b + 1) <= B:
27                      b += 1
28                  else:
29                      break
30              arr.append(p ^ b)
31          else:
32              break
33      k = 1
34      for i in arr:
35          k *= i
36      return k
37
38
39  # generating the elliptic curve and point
40  def gen_curve(n):
41      while true:
42          a = randint(2, n - 1)
43          x_P = randint(2, n - 1)
44          y_P = randint(2, n - 1)
45          b = y_P ^ 2 - x_P ^ 3 - a * x_P
46
47          # checking the curve is nonsignular on Z/nZ
48          g = gcd(4 * a ^ 3 + 27 * b ^ 2, n)
49          if g == 1:
50              break  # good to continue in this case
51          elif g == n:
52              continue  # need to regenerate the variables
53          else:  # 1 < g < n
54              print "{} divides {}".format(g, n)
55              quit()
56
57      E = EllipticCurve(Integers(n), [a, b])
58      P = E(x_P, y_P)
59
60      return [P, E]
61
62
63  def lenstra(n, B):
64      check_n(n)
65      k = gen_k(B)
66      cond = True
67      while cond:
68          [P, E] = gen_curve(n)
69          P_1 = P
70          F = factor(k)
71          factors = flatten([([p] * m) for (p, m) in F])
72          for i in factors:
73              try:
74                  P_1 *= i
```

```
            except ZeroDivisionError as e:
                s = int(str(e).split(' ')[2])
                ans = gcd(s, n)
                print "{} divides {}".format(ans, n)
                cond = False
                break


lenstra(162248396707, 10)
```

The output is:

```
918583 divides 162248396707
```

A.3 Cocks-Pinch Method

Below we have a script for generating a pairing-friendly curve using the Cocks-Pinch method. Whilst writing the script, it was ran multiple times and then updated with intermediary values. Again, the output is below. This is the curve we use in examples in section 5.

```
load(
        '/home/jack/documents/work/4th_year'
        '/dissertation/code/elliptic_class.sage'
        )
D, k, n = -7, 7, 2437
K = GF(n)
g = (K(1979)) ^ ((n - 1) / k)  # g = 801
a = K((g + 1) / 2)  # a = 401
b_0 = K(a - 1) / K(D).sqrt()  # b_0 = 196
j, q = -1, 4
while not Integer(q).is_prime():
    j += 1
    q = int(a) ^ 2 - D * (int(b_0) + j * n) ^ 2
poly = hilbert_class_polynomial(D)
L = GF(q)
new_poly = poly.change_ring(L)
list_of_roots = new_poly.roots()
j = list_of_roots[0][0]
k = L(j / (1728 - j))
c = L(159103842)
a = 3 * k * c ^ 2
b = 2 * k * c ^ 3
print 'a = ', a
print 'b = ', b
print 'q = ', q
E = ECurve(q, a, b)
E.speak()
```

The output is:

```
a =  210819370
```

```
b =  271090911
q =  692342753
We have the Elliptic Curve defined by y^2 = x^3 +
    210819370*x + 271090911 over Finite Field of size
    692342753
```

A.4 Tate Pairing

We have a script for Joux's protocol as in 5.6. We define the function `tt_par` as we do as Sage requires both points to come from the same curve.

```
load(
        '/home/jack/documents/work/4th_year'
        '/dissertation/code/elliptic_class.sage'
        )

a, b, q = 210819370, 271090911, 692342753
E = ECurve(q, a, b)
P = E.curve(565432016, 168138289)
Q1 = [
        [77275493, 6],
        [43128140, 5],
        [303325271, 4],
        [80803561, 3],
        [505621793, 2],
        [102464972, 1],
        [488677159, 0]
        ]
Q2 = [
        [556744255, 6],
        [256498800, 5],
        [677864764, 4],
        [216963775, 3],
        [304849498, 2],
        [628184040, 1],
        [522278325, 0],
        ]



def tt_par(P, Q, E):
    return E.big_curve.curve(P).tate_pairing(Q, E.n, E
        .k)


E.get_prime_subgroup()
E.get_embedding_degree()
E.get_big_curve()
Q = E.big_curve.def_point(Q1, Q2)
# print 'P', P
# print 'Q', Q
# print E.n
```

```
a, b, c = 689, 1045, 1981
Q_A, Q_B, Q_C = a * Q, b * Q, c * Q
P_A, P_B, P_C = a * P, b * P, c * P
print 'Q_A =', Q_A
print 'Q_B =', Q_B
print 'Q_C =', Q_C
print 'P_A =', P_A
print 'P_B =', P_B
print 'P_C =', P_C
S_A = tt_par(P_B, Q_C, E)**a
S_B = tt_par(P_C, Q_A, E)**b
S_C = tt_par(P_A, Q_B, E)**c
print 'Shared secrets the same?', S_A == S_B and S_A
    == S_C
print 'Shared secret =', S_A
```

The output for which is

```
Q_A = (664452163*tt^6 + 605251211*tt^5 + 523248074*tt
    ^4 + 255090228*tt^3 + 149292898*tt^2 + 663506829*tt
     + 7940698 : 486250160*tt^6 + 76211383*tt^5 +
    16956031*tt^4 + 510641267*tt^3 + 439276018*tt^2 +
    343359925*tt + 22203957 : 1)
Q_B = (429642517*tt^6 + 142581427*tt^5 + 418685613*tt
    ^4 + 437566590*tt^3 + 483466948*tt^2 + 448980515*tt
     + 254773409 : 294842493*tt^6 + 296218475*tt^5 +
    498364755*tt^4 + 160399192*tt^3 + 100602147*tt^2 +
    553837880*tt + 542995682 : 1)
Q_C = (441765606*tt^6 + 179179203*tt^5 + 687850474*tt
    ^4 + 264680871*tt^3 + 425868865*tt^2 + 45420919*tt
    + 23712548 : 23241641*tt^6 + 460341032*tt^5 +
    206417946*tt^4 + 678475761*tt^3 + 529432213*tt^2 +
    13833738*tt + 236354811 : 1)
P_A = (436557109 : 413067132 : 1)
P_B = (210938951 : 347490330 : 1)
P_C = (625029634 : 201311858 : 1)
Shared secrets the same? True
Shared secret = 592222373*tt^6 + 202365451*tt^5 +
    657132309*tt^4 + 463067684*tt^3 + 103777291*tt^2 +
    156353336*tt + 590907738
```

A.5 Miller's Algorithm

Finaly, we include a broken implimentation of Miller's Algorithm; in this, we follow [Men09]. As mentioned in said paper, the algorithm should be independent of the choice of R, however this implementation is not. Also, when $P \in E(\mathbb{F}_q)$ and $Q \notin E(\mathbb{F}_q)$, there should be no zero division errors, however this implementation does give zero-division errors in this case. These issues mean that we instead use SageMath's built-in function for Miller's algorithm, which we can see by studying the source code, uses a different algorithm without the

point R.

```
load('pair_classes.sage')

def millers_algorithm(E, P, Q):
    R = E.give_point()
    set1   = set([E.curve(0), P, -1 * Q, P - Q])
    set2   = set(E.curve.points())
    set1u2 = set1.union(set2)
    R_in_union = False
    try:
        R_small = R.change_ring(E.curve.base_field())
        R_in_union = R_small in set1u2
    except ValueError:
        print 'changing R'
    while R_in_union or R in set1u2:
        R_in_union = False
        R = E.give_point()
    print 'R', R
    one = E.curve.base_field().one()
    f, T = one, P
    for i in range(len(E.n_bin)):
        l = TanLine(T)
        v = VertLine(2 * T)
        T = 2 * T
        num = l.eval(Q + R) * v.eval(R)
        denom = v.eval(Q + R) * l.eval(R)
        f = f**2 * (num / denom)
        if E.n_bin[i] == 1:
            l = ThruLine(T, P)
            v = VertLine(T + P)
            T = T + P
            num   = l.eval(Q + R) * v.eval(R)
            denom = v.eval(Q + R) * l.eval(R)
            f = f * (num / denom)
    expon = Integer((E.q**E.k - 1) / E.n)
    return(f^expon)
```

B Alice and Bob

This paper, like many others in cryptography use the names Alice and Bob to represent the two parties that would like to exchange a shared secret, or communicate securely. These names were first used by Ron Rivest, Adi Shamir and Leonard Adleman in their paper [RSA78]. Their purpose is to aid comprehension, for example, "Bob would like to send Alice a secret message" may be easier to understand than "B would like to send a secret message to A". After the introduction of Alice and Bob, many other characters have been introduced; we list the characters used in this paper below.

- *Alice and Bob.* The two original characters. Usually Alice and Bob would like to agree upon a shared secret or send one another secure communications.
- *Charlie.* A generic third party.
- *Eve.* An *eavesdropper*. A passive attacker who is able to listen in on communications between Alice and Bob without altering or disrupting them.
- *Mallory.* A *malicious attacker*. Like Eve, Mallory is able to listen to all communications between Alice and Bob, however unlike Eve, he is able to modifiy the messages. Mallory could leave messages intact, modify them and send them on without Alice and Bob's knowledge or even stop messages all together. Securing communications against Mallory is significantly more difficult than against Eve, indeed, any system secure against Mallory is secure against Eve.
- *Trent.* A *trusted third party.* Trent will often generate for Alice and Bob the initial structures needed for them to generate their public keys.

References

[AM93] A Oliver L Atkin and François Morain. Elliptic curves and primality proving. *Mathematics of computation*, 61(203):29–68, 1993.

[BSS05] Ian F Blake, Gadiel Seroussi, and Nigel P Smart. *Advances in elliptic curve cryptography*, volume 317. Cambridge University Press, 2005.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[FST10] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of cryptology*, 23(2):224–280, 2010.

[Jou04] Antoine Joux. A one round protocol for tripartite diffie–hellman. *Journal of cryptology*, 17(4):263–276, 2004.

[Kob94] Neal Koblitz. *A course in number theory and cryptography*. Springer Science & Business Media, 1994.

[Mar19] G. Marasingha. ECM3726 - Cryptography Lecture Notes, 2019.

[Men09] Alfred Menezes. An introduction to pairing-based cryptography. *Recent trends in cryptography*, 477:47–65, 2009.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Ver04] Eric R. Verheul. Evidence that xtr is more secure than supersingularelliptic curve cryptosystems. *Journal of Cryptology*, 17(4):277–296, Sep 2004.