For elliptic curve cryptography to be safe, you should pick your elliptic curve carefully. Having not learned from my RSA debacle, I forgot the above principle when I asked my friend Kondwani in Malawi to encrypt a message for me using the elliptic curve version of the ElGamal message exchange system. Let me explain how that works.

Let \( F_p \) be the finite field with \( p \) elements for a large prime \( p \). (Note that many cryptographers would use the notation \( GF(p) \) for \( F_p \).) Let \( E \) be an elliptic curve over \( F_p \). Assume \( G \in E(F_p) \) is a point on \( E \) with coordinates in \( F_p \) that generates a large subgroup of \( E(F_p) \). Note that \( p, E \) and \( G \) are public parameters.

Let \( a_E \) be my private key number (a positive integer) and \( a_E G \) be my public key point. Kondwani has a plaintext message that he encodes on the x-coordinate of a point \( Q \in E(F_p) \). Kondwani chooses a random positive integer \( k \). He sends me the two points \( kG \) and \( Q + ka_E G \). I’ll let you convince yourself that Kondwani can find those two points and that I can find \( Q \) from those two points without needing to determine \( k \).

I was the one who originally set up the parameters of the system. We use \( p = 1937795458736239813839407261656518979123304596217 \) and the curve \( y^2 = x^3 + 963218343336110113016174981681596148974425738450x^2 + 64659056709322492479302778449481955110774251117x + 1373469572801116216304262434997226868149786911218 \) over \( F_p \). We use \( G = (1, 209937098944447720931521991946579213161073163246) \) as the generating point. My public key point is \( a_E G = (1143911082840867135024522665665689548850979758, 45431354246013168078641751647759414217206517820) \).

Kondwani sends me the point \( kG = (629912080964738498794283591624344703079802252096, 7299682339195084672923969115180432574012865805) \) and the point \( Q + ka_E G = (191793935928421628302957784538215468729955703651, 127116997489968700918382606250304914275543255608) \).

Find \( Q \) and then turn its x-coordinate into a binary string. Append a 0 at the left. That is the ASCII representation of the plaintext message, which is the codeword.

For example, if the x-coordinate were 22895 = (1011001 01101111)\(_2\), then we append 0 on the left to get 01011001 01101111 which is the ASCII encoding of Yo.

PS: You might be wondering what Kondwani would have done when his message had not been the x-coordinate of a point in \( E(F_p) \). We could pad the message with 8 bits and try each of the 256 padded messages until we find one that is the x-coordinate of a point in \( E(F_p) \). Since about half of all elements of \( F_p \) are the x-coordinates of points in \( E(F_p) \), our probability of failure is about \( 1/2^{256} \).