Elliptic Curve Cryptography

Shir Maimon

May 10, 2018

Abstract

In this paper I give an introduction into elliptic curves in order to introduce the Elliptic Curve Diffie-Hellman (ECDH) protocol and give motivation for its use as a cryptographic key exchange protocol. I then give an introduction to Shor's Algorithm for Elliptic Curves, a quantum algorithm which breaks ECDH, including describing the Quantum Fourier Transform, which is at the center of many quantum algorithms including those which solve discrete logarithms.

1 Introduction

The purpose of many cryptographic protocols is to send a message from one party to another or to reveal a limited amount of information or to compute with another party securely. Whatever it may be, most cryptographic protocols require some kind of a key, which is most likely an integer which is assumed to be hard to compute. However, this often means that some number of parties want access to a secret key without anybody else being able to discover the key. Often the parties can not meet in person so they must somehow communicate so that at the end of the communication they share a secret key, and even if someone were to see all of their communication, they would not be able to discover the key.

Elliptic Curve Cryptography is particularly useful in solving such problems. There are existing protocols, called key exchange protocols, which successfully do this, but not all key exchange protocols are made equal. Table 1 [NIS05] shows one of the most notable differences between elliptic curve protocols and protocols based on factoring or finite fields. The middle and right column give necessary key sizes for protocols with and without elliptic curves. The "symmetric key size" indicates about how many operations would be needed to break a protocol using those key sizes. If the symmetric key size is 80, this means that it would take about 2^{80} operations to break the protocol. We can see from this that the key sizes needed for elliptic curves are smaller and grow smaller than those needed without elliptic curves.

Additionally, elliptic curves can be useful because they can be put over fields of order 2^k instead of necessarily being over a prime order. This allows protocols to not use modular arithmetic but instead chop of the top bits of an integer. This is useful if elliptic curve cryptography is to be used on small devices, such as smart cards [MEES01].

In this paper I give the reasons that elliptic curves have these properties as well as present a drawback to traditional elliptic curve cryptography that may make it obsolete in the near(-ish) future.

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: Key size comparison for same level of security with and without elliptic curves

2 Elliptic Curves

An elliptic curve[1] is defined as points (x, y) satisfying $y^2 = x^3 + ax + b$ for some constants a and b. Note that the discriminant for this curve is $-16(4a^3 + 27b^2)$. Thus, in order to ensure that the curve is nonsingular we will add the restriction that this discriminant is not 0. Additionally, we will add a point at infinity, denoted by $\mathcal{O}$. Figure 1 shows examples of various elliptic curves. Figure 2 shows two examples of curves of the form $y^2 = x^3 + ax + b$, where $4a^3 + 27b^2 = 0$. We can see that in one case we are left with a cusp at $(0, 0)$ and in the other there is a self-intersection at $(1, 0)$. In either case, the slope of the curve at those respective points are undefined, which will not allow us to use the curves in our protocols.

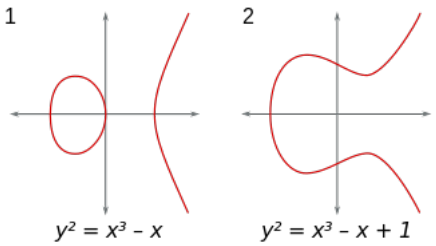

Figure 1: Examples of two elliptic curves, $y^2 = x^3 - x$ and $y^2 = x^3 - x + 1$
[Mra07]

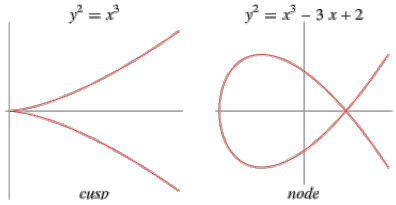

Figure 2: Examples of curves with singularities.
[Row]

Note that we will be considering elliptic curves defined over real numbers. If one considers curves defined over complex numbers, the curve is isomorphic to a torus along with a point at infinity. The curves that we see are slices of some torus.

Group Defined Over Elliptic Curves

The usefulness of elliptic curves arises from the fact that they naturally form a group. The elements of this group are exactly those previously described, namely $\{(x, y) \mid y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$, where a and b are constants such that $4a^3 + 27b^2 \neq 0$. The identity of this group is $\mathcal{O}$. In order to prove this is a group we first need to describe the group operation, which we will call addition.

Addition of Points on the Curve

Addition is defined as follows: for three linearly aligned non-zero points P, Q, and R on the curve, $P + Q + R = \mathcal{O}$. One can see in Figure 3 how this looks visually. In the first case, we have a curve and see that because the line connecting P and Q hits the curve at R, $P + Q + R = \mathcal{O}$ so $P + Q = \mathcal{O} - R = -R$ because $\mathcal{O}$ is the identity. However we now need to describe inverses so that we can say what $-R$ actually is.

[1]For the purpose of this paper we will consider only elliptic curves over fields of characteristic not equal to 2 or 3. This simplifies many of the equations and these curves are the ones commonly used in elliptic curve cryptography.

The inverse of $P = (x, y)$ is $-P = (x, -y)$. This point $-P$ is clearly on the curve if and only if P is on the curve. Thus in order to add two points P and Q we must take the line connecting P and Q and see where it intersects the curve. If it intersects at R then $P + Q = -R$.

Just by looking at Figure 3 we can see that there are many cases where there may not be a third point on the curve on the line connecting P and Q. One trivial case is when either P or Q is $\mathcal{O}$. It is trivial because $P + \mathcal{O} = P$ by definition of identity. Similarly, $P + -P = \mathcal{O}$ by the definition of an inverse. In any other case, the line connecting P and Q must be tangent to the curve at either P or Q. We treat this similarly to how we might treat a double root. That is, if the line is tangent at P, then we say the third point is P, and so $P + Q = -P$.

On case that we have not covered is when there is more than one line connecting P and Q. This only occurs if $P = Q$. In this case we take the line tangent to the curve at P and find the other point where this intersects the curve. If this point is R then $P + P = -R$. If the tangent line is vertical then $P + P = \mathcal{O}$.

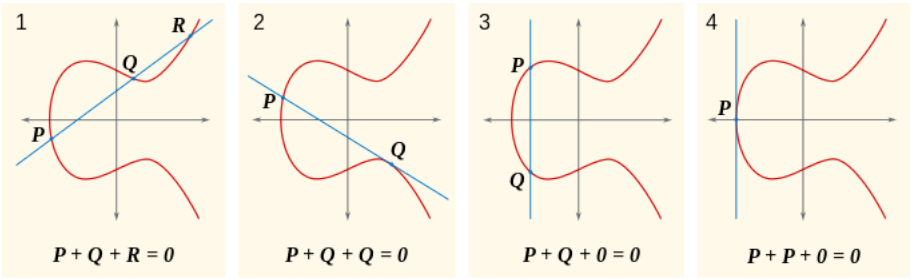

Figure 3: Addition of points on Elliptic Curves
[Sup07]

Computing the Sum of Two points on the Curve

Suppose that $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$. Then the slope of the line connecting P and Q is

$$\lambda = \begin{cases} \frac{y_P - y_Q}{x_P - x_Q} \text{ if } P \neq Q \text{ and } P \neq -Q \\ \frac{3x_P^2 + a}{2y_P} \text{ if } P = Q,\ y_P \neq 0 \end{cases}$$

where the second case is easily computed by taking the partial derivative of the curve. Note that we have missed the cases when $P = -Q$ or $P = Q$ and $y_p = 0$ but then the sum is just $\mathcal{O}$. The line intersects the y-axis at $\upsilon = y_1 - \lambda x_1$. Finding the third point is simply a matter of finding the points where both $y^2 = x^3 + ax + b$ and $y = \lambda x + \upsilon$, so $0 = x^3 + ax + b - (\lambda x + \upsilon)^2 = x^3 - \lambda^2 x^2 + (a - 2\lambda\upsilon)x + (b - \upsilon^2)$. As this is a cubic equation, we know that it is equal to $(x - z_1)(x - z_2)(x - z_3)$ for some z_1, z_2, z_3. The fact that (x_P, y_P) and (x_Q, y_Q) intersect both the line and the curve implies that the equation is actually equal to $(x - x_P)(x - x_Q)(x - x_R) = x^3 - (x_Q + x_P + x_R)x^2 + (x_Q x_P + x_P x_R + x_R x_Q)x - x_P x_Q x_R$, where (x_R, y_R) are the coordinates of the third point. Thus we have that $(x_Q + x_P + x_R) = \lambda^2$ and therefore $(x_R) = \lambda^2 - x_P - x_Q$. Using $y = \lambda x + \upsilon$ we have that $y_R = \lambda x_R + \upsilon$. Thus we can easily compute $P + Q = -R$.

Proof That the Curve is an Abelian Group

This mostly follows trivially from what we have shown. The elements include the identity and every element has an inverse. The group operation is well defined and the group is closed under the operation. It is also clearly commutative, so the group is abelian.

All that is left to show is associativity. Given P, Q, and R on the curve, we need to show that $(P + Q) + R = P + (Q + R)$. This can be shown by simply plugging in what we found in the last section as the formula for addition. As this proof is simply tedious algebraic manipulation it is omitted.

Fast Scalar Multiplication

The multiplication of a point on an elliptic curve is exactly as we would expect. To be rigorous, $1 * P = P$ and $n * P = P + (n - 1) * P$. In order to actually do any sort of protocol using an elliptic

curve we need a fast way to multiply a point by a scalar (we will see this in the next section). It is important to know that when we consider the run time of an algorithm, we look at how it changes as the **size of** the input grows. If we were to simply add P to itself n times, this would take time $O(n)$[2]. But the size of n is actually the length of n. The length of n, denoted $|n|$ is equal to $\lfloor log_2(n) \rfloor + 1$ (unless $n = 0$ in which case $|n| = 1$), or the number of digits used to represent n in base-2. Thus the naive algorithm takes time $O(2^{|n|})$, which is not fast enough[3].

The algorithm we describe has runtime $O(log(n)) = O(|n|)$, so we say it takes *linear time*. Consider the binary representation of n, say $b_k b_{k-1} ... b_0$. This means that $n = b_k 2^k + b_{k-1} 2^{k-1} ... b_1 2 + b_0$, where $k = |n| - 1$ and each $b_i \in \{0, 1\}$. Thus $nP = b_k 2^k P + b_{k-1} 2^{k-1} P ... b_1 2P + b_0 P$. The algorithm proceeds as follows:

procedure MULTIPLY(n, P)
  Let $n = b_k b_{k-1} ... b_0$ in binary
  Set $i = 0$
  Set $tempval = P$ ▷ tempval is $2^i P$
  Set $total = 0$
  while $i < |n|$ **do**
    if $b_i = 1$ **then**
      Set $total = total + tempval$
    end if
    Set $i = i + 1$
    Set $tempval = tempval + tempval$ ▷ $2^i P + 2^i P = 2^{i+1} P$
  end while
  return a
end procedure

The algorithm works by calculating $2^i P$ from $2^{i-1} P$ using one addition of points, and therefore only doing a total of $|n|$ additions. The algorithm therefore takes time $O(|n|)$, so we can use it in the protocol we use in the next section.

3 Elliptic Curve Diffie-Hellman

Elliptic Curves Over Finite Fields

Elliptic Curve Diffie-Hellman is a protocol done using an elliptic curve on a finite field. That is, rather than the elements being $\{(x, y) \mid y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$, they will be $\{(x, y) \in \mathbb{F}_p \mid y^2 \equiv x^3 + ax + b \bmod p\} \cup \{\mathcal{O}\}$ for some prime p. As before we will require that $4a^3 + 27b^2 \neq 0$. We can denote an elliptic curve over a finite field as $E(\mathbb{F}_p)$.

Note that this clearly still describes a group when paired with the addition operation which is the same as the original one we defined earlier except that each coordinate is take modulo p. However, we now need to take inverses modulo p instead of dividing when finding the sum of two points. For example, the x-coordinate of the sum of P and Q if $P \neq Q$ and $P \neq -Q$ is $(\frac{y_P - y_Q}{x_P - x_Q})^2 - x_P - x_Q$, which may not be an integer. In order to add fast we can use the well known Extended Euclidean Algorithm to find the inverse of an integer modulo p.

Discrete Logarithm Problem on Elliptic Curves in Finite Fields

The discrete logarithm problem is the following: Given points P and Q on elliptic curve E such that $P = nQ$ for some n, find n.

Currently there is no known classical algorithm to quickly solve the discrete logarithm problem in general. One common algorithm is called Pollard's Rho algorithm, which we will discuss later on. However, note that in a finite field there are only a finite number of points and therefore only a finite number of points on the curve. Thus every point has a finite order. If P has a low order then it is possible to find n quickly, simply by iterating through all its possible values. Thus in

[2]Throughout this paper I will use this notation, called *Big-O notation*, to describe the runtime of an algorithm. We say that $f(x) = O(g(x))$ if there exists a positive constant c and a number x_0 such that $|f(x)| \leq c|g(x)|$ for $x > x_0$.

[3]In general in computer science an algorithm is considered fast if it takes polynomial time, that is it runs in $O(n^k)$ for some constant k

order for the protocol to be useful we must be able to quickly find the a point on an elliptic curve in a finite field with high order.

Finding a point with a High Order In $E(\mathbb{F}_p)$

First we need to quickly be able to find the order of a point P, or equivalently, the order of the subgroup generated by P. It is well know that the order of a subgroup divides the order of a group by Lagrange's Theorem. Thus if we have the order of $E(\mathbb{F}_p)$ then we can simply check if $nP = \mathcal{O}$ for all n dividing the order of the group and taking the smallest such n. I will not describe an algorithm to find the order of the curve however this algorithm, called Schoof's Algorithm, can be found in [Sch95].

For a curve to be useful it must not have prime order. In particular, it should have order N where N is divisible by some large prime q. Suppose that an elliptic curve over a finite field has order N and q is a large prime dividing N. For any point P on the curve $NP = \mathcal{O}$ because the order of P divides the order of the curve. If $h = N/q$ then $NP = qhP = q(hP) = \mathcal{O}$. Thus either the order of point hP divides q or $hP = \mathcal{O}$. If the order of hP divides q then in fact it is q because q is prime, and so hP is a point of high order in $E((F)_p)$. To find a point of high order, then, we can simply take random points P on the curve and multiply them by h until we find a P such that $hP \neq \mathcal{O}$.

Protocol

Elliptic Curve Diffie Hellman is a key exchange protocol which is very similar to the traditional Diffie Hellman protocol. There are two parties, say Alice and Bob, who want to share a secret. They can communicate over a channel which may have eavesdroppers, but they want to ensure that an eavesdropper can not find out the shared secret.

First there is some set of agreed upon domain parameters (p, a, b, G, n, h), where p is the size of the finite field (that is, the curve is $E(\mathbb{F}_p)$), a and b are the coefficients of the equation of the curve, G is a point in the curve which generates a subgroup of size n, and h is the cofactor of G. The exchange is described by the Table 2.

Alice	Insecure Channel	Bob
Generates random $d_A \in \{1, ..., n-1\}$		Generates random $d_B \in \{1, ..., n-1\}$
Set $H_A = d_A G$		Set $H_B = d_B G$
	$\Rightarrow$ H_A	
	$\Leftarrow$ H_B	
Compute $S = d_A H_B$		Compute $S = d_B H_A$

Table 2: Diffie-Hellman Key Exchange Protocol

In this exchange, both Alice and Bob generate a random integer between 1 and $n-1$, d_A and d_B respectively. They then each send G multiplied by their integer and compute the shared secret $S = d_A H_B = d_A(d_B G) = d_B(d_A G) = d_B H_A$. Alice and Bob may then use S somehow as a key, for example by taking the x-coordinate of the point.

For an adversary to find the secret S, they must be able to solve the problem of give $d_A G$ and $d_B G$, what is $d_A d_B G$. Of course a fast solution to the Discrete Logarithm Problem on Elliptic Curves in Finite Fields would solve this, which is why it is important that there is currently no fast classic algorithm which solves this problem.

In the next section I describe the Pollard's Rho Algorithm, which is a classic algorithm to solve the Discrete Logarithm Problem on Elliptic Curves in Finite Fields. The fact that this algorithm is truly exponential, whereas there are sub-exponential algorithms for factoring, is why Elliptic Curve Cryptography became popular in the first place. However in the following section I describe the quantum algorithm which *can* quickly break EEDH by solving the Discrete Logarithm Problem.

4 Pollard's Rho Algorithm

Pollard's Rho Algorithm is a probabilistic algorithm which originally was created for factoring integers but has a variation which solves the discrete logarithm problem for any abelian group[Pol78]. The algorithm has time complexity $O(\sqrt{n})$, where n is the number of points in the group. Note that this is $O(2^{|n|/2})$ and therefore an exponential algorithm.

All other algorithms to solve the discrete logarithm problem similarly take exponential time, but this algorithm differs in that the space complexity is constant. This improvement on other similar algorithms is why it is more likely to be used in practice and why I describe it here. The paper [WYMAR15] describes how Pollard's Rho can be used specifically for elliptic curves and much of this section is attributed to this explanation.

4.1 The Algorithm

The goal of the algorithm is, given an group $G, +$ and $P, Q \in G$ where $P = kQ$ for some integer k, to find k. The algorithm looks for a_1, b_1, a_2, b_2 such that $a_1P + b_1Q = a_2P + b_2Q$, so $\frac{a_1-a_2}{b_2-b_1}P = Q$. In other words, $k(b_2 - b_1) = (a_1 - a_2) \bmod n$, where n is the order of the group, unless $b_2 - b_1$ is not relatively prime to n.

If $b_2 - b_1$ is not relatively prime to n then we can essentially divide the equation $k(b_2 - b_1) = (a_1 - a_2) \bmod n$ out by the greatest common denominator of $b_2 - b_1$ and n and try every solution to that equation which is less than n. More precisely, if $d = \gcd(n, b_2 - b_1)$ then $k\frac{(b_2-b_1)}{d} = \frac{(a_1-a_2)}{d} \bmod \frac{n}{d}$ will have a unique solution less than n/d, so it will have d solutions less than n. One of these will yield $k(b_2 - b_1) = (a_1 - a_2) \bmod n$ and will therefore be the solution to the discrete logarithm. Note that this algorithm is probabilistic but not random, so unless one makes changes to things like the recurrence relation or partitions, the same coefficients will be found on every run.

First, we partition the group into three sets, S_0, S_1, and, S_2. We do this any way so that the sets are about all the same size, so for example taking the x-coordinate mod 3. Next we define a sequence of points $\{A_i\}$ such that each A_i is a linear combination of P and Q. We can do this by taking $A_0 = \alpha P$ for any α. The we define each A_i by the following recurrence relation:

$$A_{i+1} = \begin{cases} A_i + P & \text{if } A_i \in S_1 \\ 2A_i & \text{if } A_i \in S_2 \\ A_i + Q & \text{if } A_i \in S_3 \end{cases}$$

Each A_i is clearly a linear combination of P and Q.

The field has only a finite number of points, so the curve must also have a finite number of points. Thus it must be that at some point, $A_i = A_j$ for some i, j.

At this point we could use A_i and A_j to find k, but it would involve storing every A_i. Instead we notice that if $A_i = A_j$ then $A_{i+t} = A_{j+t}$ for any non-negative integer t. The reason that this is called the Rho algorithm is that the sequence of points can be thought of to look like a ρ, written from the bottom up. The intersection of the ρ is the first i, j where $A_i = A_j$, then after that we have a cycle. The part of the sequence before this cycle is called the "tail". Suppose the tail has length t and the cycle is of length c. I claim there is a point A_n such that $A_n = A_{2n}$.

Proof. For any A_i in the cycle, it is of the form $t + k + nc$, where $0 \leq k < c$. Here k represents the position of A_n after the initial intersection and n represents the number of times we have to go around the cycle. This point is equivalent to all points of the form $t + k + mc$. Thus we simply want $2(t + k + nc) = t + k + mc$ or $t + k + 2nc = mc$. Because we can choose m and n, any positive k such that $t + k$ is a multiple of c satisfies the requirement, and $A_{t+k} = A_{2(t+k)}$. □

Now for the algorithm all we need to store is (i, A_i, A_{2i}) at any step. If $A_i = A_{2i}$, we are done. Otherwise, it only takes one point addition to compute A_{i+1} from A_i a two point additions to compute $A_{2(i+1)}$ from A_{2i}, so we can find $(i + 1, A_{i+1}, A_{2(i+1)})$ using just three point additions.

4.2 Runtime of the Algorithm

In 1960, Harris [Har60] showed that if the sequence of point is actually random yet deterministic in the sense that if $A_i = A_j$ then $A_{i+t} = Aj + t$ for all non-negative t, then it should take about $\sqrt{n\pi/2}$ iterations to find two points which are the same. Our sequence of points is not random,

however as we add partitions and complicate the iteration function, our sequence looks more random and we are more likely to find a match in $\sqrt{n\pi/2}$ iterations. Thus the algorithm takes on the order of $\sqrt{n}$ time, so we say the algorithm is $O(\sqrt{n})$.

5 Introduction To Quantum Computation

In classical computation, any information is represented in bits. A bit is either 0 or 1 and therefore n bits have 2^n possibilities. In quantum computation, we deal with what is called a "quantum bit" or "qubit". A qubit is of the form $\alpha_0 |0\rangle + \alpha_1 |1\rangle$, where α_0 and α_1 are complex numbers and $|\alpha_0|^2 + |\alpha_1|^2 = 1$. When qubits are measured, they are either a 1 or a 0, and so what this notation actually means is that the probability that the qubit will be measured to be 1 is $|\alpha_1|^2$ and the probability that it will be measured to be 0 is $|\alpha_0|^2$.

The states $|0\rangle$ and $|1\rangle$ are called computational basis states. This is because a state can be considered to be a vector in a two dimensional complex vector space, where the vector must have a norm of 1. The two basis states form an orthonormal basis for this space.

Before a qubit is measured it is actually in state $\alpha_0 |0\rangle + \alpha_1 |1\rangle$, it is not 0 or 1 (unless either $\alpha_0 = 0$ or $\alpha_1 = 0$). After a qubit is measured, it collapses to whichever state it was measured as. That is, if we measure a qubit to be 0, then the state of the qubit after measurement will actually be $|0\rangle$. The information in this section is adapted from [Hir10].

5.1 Multiple Qubits and Quantum Entanglement

Quantum bits get much more interesting when we consider states of multiple quantum bits. When dealing with classical bits, two bits can be either 00, 01, 10, or 11. With quantum bits, the state of two qubits is of the form $\alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle$ where, similarly to before, $|\alpha_{ij}|^2$ is the probability that the qubits will be i and j respectively, so $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$.

Note that though each of the bases has a value for both of the qubits, we can still measure each qubit individually. This gives interesting properties to qubits because of the collapsing described earlier. If we measure the first qubit to be a 0, then the sate of the qubits collapses to $\frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}$, where we divide by $\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}$ to keep the norm at 1. Before measuring the first qubit, the probability that the second qubit was 0 was $|\alpha_{00}|^2 + |\alpha_{01}|^2$, but after measuring the first qubit, the probability that the second qubit is 0 is $|\alpha_{00}|^2$. Though we have not acted on the second qubit, the probability that it is 0 may change, just be measuring the first qubit. When the state of a particle can not be described independently of other qubits, we say that these particles are *entangled*.

One simple example of entanglement is the Bell State, or $\frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle$. In this case, if we measure a qubit to be 0, the second qubit will be measured to be a 0 with probability 1, and if we measure a qubit to be 1, the other will be 1 with probability 1.

Any arbitrary number of qubits works similarly. That is, if we have n qubits, our states look like $\Sigma_{i=1}^{2^n} \alpha_i |i\rangle$, where i is actually the i'th bit string, and $\Sigma_{i=0}^{2^n} |\alpha_i|^2 = 1$.

When states are not entangled they are called *decomposable*.

5.2 Quantum Computation

Just as in classical computing, in quantum computing we need a way to act on bits. We have just described how we read bits, by measuring, but now we describe how computations work in general. First, as we said, the state of a qubit can be considered a vector in two dimensional complex vector space. For example, if we have a one state quantum system, the state $|0\rangle$ can be represented as $\binom{1}{0}$, $|1\rangle$ can be represented as $\binom{0}{1}$, and any linear combination of $|0\rangle$ and $|1\rangle$ should then be represented as a linear combination of these two matrices. Similarly, the state of n qubits could be considered a vector in 2^n dimensional vector space. For example, in a quantum system of 2 qubits, the state $\alpha_1 |00\rangle + \alpha_2 |01\rangle + \alpha_3 |10\rangle + \alpha_4 |11\rangle$ can be written as the vector

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix}.$$

In classical computation, we often see algorithms in the form of pseudocode, but in fact any algorithm can be written as something called a boolean circuit. Quantum algorithms are often written as *quantum circuits*. A quantum circuit is similar to a boolean circuit except that the gates can not be the same as boolean gates. Instead every gate is a *unitary matrix*.

Definition 5.1. A matrix M is *unitary* if MM^*=I, where M^* denotes taking the complex conjugate of every entry in the matrix and then transposing the matrix, and I is the identity matrix.

The crucial property of unitary matrices in this case is that they preserve the L^2-norm of a vector. That is, if U is unitary, then for any vector x, $\|Ux\| = \|x\|$, where in this case $\|x\|$ is the sum of the norms of entries in x. The proof of this is fairly straight forward. If U is unitary and x is any vector, then $\|Ux\| = (Ux)^*(Ux) = x^*U^*Ux = x^*x = \|x\|$. Similarly, if $\|Ux\| = \|x\|$ for all x, then $x^*U^*Ux = x^*x$ for all x so we must have that $U^*U = I$. Thus, unitary matrices are actually exactly those which preserve L^2-norm.

A quantum gate works simply by applying the matrix to the vector. It is clear why it is important that a quantum gate preserves L^2-norms. The norm of a quantum state must always be 1, so both before and after the operation, the norm must be 1, and thus the matrix must be unitary.

Example 5.1. The matrix

$$M_\neg = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

is known as the quantum not-gate. It is easy to verify that $M_\neg$ is unitary and that for any state $\alpha_0 |0\rangle + \alpha_1 |1\rangle$, $M_\neg$ applied to the state is $\alpha_1 |0\rangle + \alpha_0 |1\rangle$. Thus, the matrix maps $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$, a natural analog of the classical not-gate.

Example 5.2. The matrix

$$W_2 = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$$

is called the Hadamard or Hadamard-Walsh gate and is an important gate for many quantum algorithms. Again it is easy to verify that this is a unitary matrix. It is easy to verify that $W_2 |0\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$ and that $W_2 |1\rangle = \frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle$. Because the probability of observing some basis is the norm of the coefficient of that basis, in both the previous cases we end up with a $\frac{1}{2}$ probability of observing either basis.

Note that W_2 is its own complex conjugate transpose so, as it is unitary, $W_2W_2 = I$. In particular, this means that $W_2(\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle) = |0\rangle$ and $W_2(\frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle) = |1\rangle$, even though the initial probabilities for either $|0\rangle$ and $|1\rangle$ before the operation was the same. This showcases what is called *constructive and destructive interference*. In the first case we can see that

$$\begin{aligned} W_2(\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle) = \\ \frac{1}{\sqrt{2}} W_2 |0\rangle + \frac{1}{\sqrt{2}} W_2 |1\rangle = \\ \frac{1}{2} |0\rangle + \frac{1}{2} |1\rangle + \frac{1}{2} |0\rangle - \frac{1}{2} |1\rangle = \\ |0\rangle \end{aligned}$$

The effect on the amplitudes of $|0\rangle$, where overall its probability increased, is constructive interference whereas the effect on the amplitudes of $|1\rangle$ is destructive interference. In quantum algorithms, the hope is that the values that we want to see (the values that will help us find the solution to the problem) interfere constructively and the values we do not want to see interfere destructively, so overall there is a high probability that we observe some "good" value.

In the previous section we mentioned that if a quantum state is not entangled then it is decomposable. Another way to say that a state z is decomposable is that it can be written as the tensor product of states x and y, or $z = x \otimes y$. For example, $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

Example 5.3. Let $W_4 = W_2 \otimes W_2$, where W_2 is as in the previous example. Then we have

$$W_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Because W_4 is the tensor product of two unary gates, if we apply W_4 to any non-entangled state, the resulting state will similarly be decomposable.

Example 5.4. Now let

$$M_{\text{cnot}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

This is called the controlled not gate as, for any basis state, it flips the second qubit if and only if the first qubit is 1. Suppose we have an initial (clearly decomposable) state $|00\rangle = |0\rangle\,|0\rangle$. We have

$$\begin{aligned} M_{\text{cnot}}((W_2\,|0\rangle)\,|0\rangle) = \\ M_{\text{cnot}}(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,|0\rangle) = \\ M_{\text{cnot}}(\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \\ \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \end{aligned}$$

That is, if we take $|00\rangle$ and apply the Hadamard-Walsh gate to the second qubit, then apply the controlled not gate to the system, we end up with entangled state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. We mentioned in the previous section that this state is entangled, however it is easy to verify that it can not be written as the product of decomposable states. This is significant as it shows how to actually entangle two particles.

6 Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves

Shor's Algorithm is a very well known quantum algorithm used to factor integers. In the same paper, [Sho97], Shor also introduced a similar quantum algorithm to find the discrete logarithm over cyclic groups. This algorithm was extended by Boneh and Lipton [BL95] to work for noncyclic abelian groups as well. Thus, this algorithm works for the elliptic curve discrete logarithm problem as well. The basis of these algorithms is the quantum Fourier Transform (QFT). The information in this section is adapted from Shor's paper, [Bac06], and [Hir10].

6.1 Quantum Fourier Transform

The Fourier transform is a concept in mathematics unrelated to quantum computation. For some function f, the Fourier transform of f is

$$F(k) = \textstyle\int_{-\infty}^{\infty} f(x)e^{-2\pi ikx}dx.$$

This is also called the *continuous* Fourier transform because there is a discrete Fourier transform which is taken over a sum rather than an integral. The discrete Fourier transform is given by

$$F_j = \Sigma_{k=0}^{N-1} e^{-\frac{2\pi ijk}{N}} f_k$$

where here f is a vector of length N. Discrete fourier transforms are particularly useful in evaluating the periodicity of a function. The Quantum Fourier Transform is an analog of the Discrete Fourier Transform where the vector is a quantum state. That is, the Quantum Fourier Transform performs the following map:

$$|x\rangle \to \frac{1}{\sqrt{n}}\Sigma_{k=0}^{n-1} e^{-\frac{2\pi i x k}{n}} |k\rangle$$

where n is the number of basis states. Note that here we are implicitly mapping quantum states to integers in a natural way (the state that is all 0s maps to 0, for example). Note that we multiply by $\frac{1}{\sqrt{n}}$ in order to normalize the summation and ensure that the norm of the vector is 1.

For a state in general, we must then have:

$$\Sigma_{j=0}^{n-1}\alpha_j |j\rangle \to \Sigma_{j=0}^{n-1}\alpha_j \frac{1}{\sqrt{n}}\Sigma_{k=0}^{n-1} e^{-\frac{2\pi i j k}{n}} |k\rangle = \Sigma_{k=0}^{n-1}\frac{1}{\sqrt{n}}\Sigma_{j=0}^{n-1}\alpha_j e^{-\frac{2\pi i j k}{n}} |k\rangle.$$

Thus the coefficient for basis state $|k\rangle$ in the Fourier transform of $\Sigma_{j=0}^{n-1}\alpha_j |j\rangle$ is

$$\alpha'_k = \frac{1}{\sqrt{n}}\Sigma_{j=0}^{n-1}\alpha_j e^{-\frac{2\pi i j k}{n}}.$$

Thus the matrix for this transformation, say A, is simply the one which has $\frac{1}{\sqrt{n}} e^{-\frac{2\pi i j k}{n}}$ as the entry in spot (k, j) of the matrix.

Claim 6.1. The matrix A is unitary.

Proof. We need to show that $A^*A = I$. The entry at (m, m') of A^*A is the dot product of the nth row of A^* and the mth column of j which is:

$$\Sigma_{j=0}^{n-1} A^*_{(m,j)} A_{(j,m')} =$$
$$\frac{1}{n}\Sigma_{j=0}^{n-1} e^{\frac{2\pi i j m}{n}} e^{-\frac{2\pi i j m'}{n}} =$$
$$\frac{1}{n}\Sigma_{j=0}^{n-1} e^{\frac{2\pi i j (m-m')}{n}}$$

If $m = m'$ then every term in the sum equals 1 and so the sum is $\frac{1}{n}n = 1$. If $m \neq m'$ then the sum is 0 because $e^{\frac{2\pi i j (m-m')}{n}}$ are exactly the nth roots of unity and so must sum to 0. Thus $(A^*A)_{m,m'}$ is 1 if $m = m'$ and 0 otherwise, so it is the identity matrix. □

As the matrix needed is unitary, the Quantum Fourier Transform can actually be computed with some quantum algorithm. In fact, if $n = 2$, the QFT matrix is exactly the Hadamard-Walsh gate, which will also be useful in computing the QFT for other orders.

6.2 Quantum Fourier Transfer Algorithm

In this section I describe the algorithm, or circuit, for the Quantum Fourier Transform. Although we have shown that there must be a circuit to compute the QFT, we still need to show that this circuit is of polynomial size, as in it consists of a polynomial number of gates. For our purposes, we are considering a system of b bits, so the basis states are binary strings of length b, so $n = 2^b$.

First, we rewrite the Quantum Fourier Transform to be in a product form. From this point on, we denote $e^{\frac{2\pi i}{n}}$ as w_n for ease of notation. If $n = 2^b$ we have that the QFT is the following transformation:

$$|x\rangle \to \frac{1}{\sqrt{2^b}}\Sigma_{y=0}^{2^n-1} w_{2^b}^{-xy} |y\rangle.$$

The state $|y\rangle$ is simply $|y_1, y_2, \ldots, y_b\rangle$ where each $y_i \in \{0, 1\}$. Additionally, the y in the exponent is the number such that its binary representation is $y_1 y_2 \ldots y_b$ so it is $2^{b-1}y_1 + 2^{b-2}y_2 + \ldots + 2y_{b-1} + y_b$. Thus the QFT can be rewritten as

$$|x\rangle \to \frac{1}{\sqrt{2^b}}\Sigma_{y=0}^{2^n-1} w_{2^b}^{-x\Sigma_{k=1}^{b} 2^{b-k} y_k} |y_1, y_2, \ldots, y_b\rangle.$$

But $w_{2^b}^{-x\Sigma_{k=1}^{b} 2^{b-k} y_k} = \Pi_{k=1}^{b} w_{2^b}^{-x 2^{b-k} y_k}$. Thus we have that the QFT can also be written as

$$|x\rangle \to \frac{1}{2^b}\Sigma_{y_1,\ldots,y_b\in\{0,1\}} \bigotimes_{k=1}^{b} w_{2^b}^{-x2^{b-k}y_k} |y_k\rangle$$
$$= \frac{1}{2^b} \bigotimes_{k=1}^{b} [\Sigma_{y_k\in\{0,1\}} w_{2^b}^{-x2^{b-k}y_k} |y_k\rangle].$$

Notice that the sum is now over only two values, $y_k = 0$ and $y_k = 1$. Thus the transformation can easily be rewritten as

$$|x\rangle \to \frac{1}{2^b} \bigotimes_{k=1}^{b} [\Sigma_{y_k\in\{0,1\}} (|0\rangle + w_{2^b}^{-x2^{b-k}} |1\rangle)].$$

Now the QFT is written completely as a product of states. We will complicate this product slightly more so that it is easier to see the sequence of gates that can create this transformation. First, note that $w_{2^b}^{-x2^{b-k}} = e^{\frac{-2\pi i x 2^{b-k}}{2^b}} = e^{\frac{-2\pi i x}{2^k}}$. This function is periodic such that if we let $f(x) = e^{\frac{-2\pi i x}{2^k}}$, $f(x+2^k) = f(x)$. Thus, any bits past the bottom k bits of x do not actually matter when evaluating this function.

Additionally, note that if a has $a_1 a_2 \ldots a_n$ as its binary representation then $a/2^n = a_1/2 + a_2/4 + \ldots + a_n/2^n$. Thus if we say that $0.a_l a_{l+1} \ldots a_n = a_l/2 + a_{l+1}/4 + \ldots + a_n/2^n$, similar to decimal notation, we have that the transformation is

$$|x\rangle \to [|0\rangle + e^{-2\pi i 0.x_n} |1\rangle] \otimes [|0\rangle + e^{-2\pi i 0.x_{n-1}x_n} |1\rangle] \otimes \ldots \otimes [|0\rangle + e^{-2\pi i 0.x_1 x_2 \ldots x_{n-1}x_n} |1\rangle].$$

Note that the coefficients for the ith qubit are 1 for $|0\rangle$ and $e^{-2\pi i 0.x_{n-i+1}\ldots x_{n-1}x_n}$ for $|1\rangle$. Thus, if we consider the qubits to be in opposite order, the coefficient of the qubit i only depends on qubit j if $j < i$ and both qubits are 1. The way that it affects a coefficient is by multiplying the coefficient by $e^{\frac{\pi i}{2^{j-i}}}$.

Consider again the Hadamard gate:

$$W_2 = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$$

Consider the state $|x\rangle = |x_1, x_2 \ldots x_b\rangle$. Applying the Hadamard gate to just the first qubit creates the state

$$\frac{1}{\sqrt{2}}[|0\rangle + (-1)^{x_1} |1\rangle] \otimes |x_2, \ldots, x_b\rangle =$$
$$\frac{1}{\sqrt{2}}[|0\rangle + e^{-2\pi i 0.x_1} |1\rangle] \otimes |x_2, \ldots, x_b\rangle.$$

Now let us consider the following matrix, which is a rotation matrix as it does not change any amplitudes but simply what is called the *phase*.

$$M_{j,l} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\frac{\pi i}{2^{l-j}}} \end{pmatrix}$$

Then $M_{j,l}$ does exactly what we need. It multiplies the coefficient by $e^{\frac{\pi i}{2^{l-j}}}$ if and only if both the qubits are 1. Thus if we apply the gate $M_{1,l}$ to qubits 1 and l respectively for all $l > 1$, we will be in the state

$$\frac{1}{\sqrt{2}}[|0\rangle + e^{-2\pi i 0.x_1 x_2 \ldots x_n} |1\rangle] \otimes |x_2, \ldots, x_b\rangle.$$

We can continue this process of applying the Hadamard onto the next bit and the rotation between this bit and all bits higher than it for all of the qubits. At the end we will obtain

$$[|0\rangle + e^{-2\pi i 0.x_1 x_2 \ldots x_{n-1}x_n} |1\rangle] \otimes \ldots \otimes [|0\rangle + e^{-2\pi i 0.x_{n-1}x_n} |1\rangle] \otimes [|0\rangle + e^{-2\pi i 0.x_n} |1\rangle]$$

which is the reverse of what we wanted. To fix this we can either just read the bits in reverse or apply a simple swap transformation which will reverse the order of the bits.

Figure 4 shows the circuit for the Quantum Fourier Transform where in the diagram H represents the Hadamard gate and R_i represents, if it is on line j, $M_{j,i}$. We can see that first we apply the Hadamard to the first bit, then the rotation matrix (the matrix which affects the phase only) between a bit and all bits after it. We do this for every qubit until we have the full QFT.

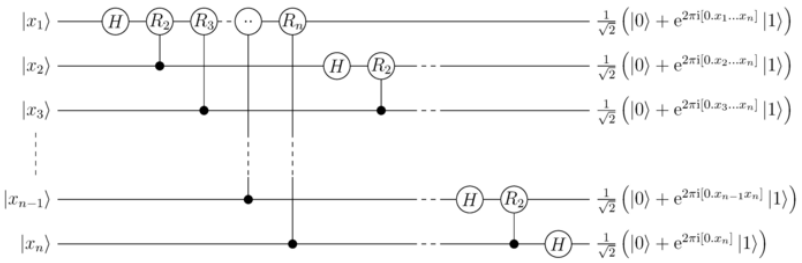

Figure 4: Circuit for the Quantum Fourier Transfer with n qubits [Tre18]

6.3 Application of Quantum Fourier Transform to the Discrete Logarithm Problem

The reason that the Quantum Fourier Transform is useful is that it, and its inverse, allows us to find periods of functions. Consider the function $f(x, y) = xP + yQ$. If P and Q are points on an elliptic curve over a finite field such that P is a multiple of Q then there is some a_1, a_2 such that $f(x+ia_1, y+ia_2) = f(x, y)$. Thus we consider a_1 and a_2 to jointly be the period of the function. If we can find the period of the function then we can find two linear combinations of P and Q which equal each other. At that point we can use the same method we used in Pollard's Rho Algorithm to find the discrete logarithm.

7 Conclusion

I have described Elliptic Curve Cryptography, why it is considered useful, and introduced a quantum attack on the Discrete Logarithm Problem for Elliptic Curves. One important thing to note is that the quantum attack works for elliptic curves and is also the attack on finite fields and factoring, which are used in RSA and some forms of Diffie-Hellman. The quantum attack on elliptic curves does not use less qubits or gates than the attack on finite fields, which in a way makes the attack stronger on elliptic curves. This is because, as we have mentioned a few times, the keys for elliptic curves can be shorter than those for finite fields or factoring and offer the same amount of classical security. Thus, if we take two protocols, one using elliptic curves and the other using factoring, which offer the same amount of classical security, the quantum attack on the elliptic curve protocol will be stronger in that it will require less time and qubits. This may seem convoluted or not to be a good argument that elliptic curves are more vulnerable to quantum attacks, however, when protocols are implemented, shorter keys really are used on algorithms that utilize elliptic curves. Thus, currently, elliptic curve protocols are in fact more vulnerable to quantum attacks.

There is still, however, some hope for the use of elliptic curves in a post-quantum world. An *isogeny* is a rational homomorphism between curves. There is a class of elliptic curves called supersingular elliptic curves in which the group of isogenies is not commutative, and there is a key exchange protocol similar to Diffie-Hellman which utilizes this fact and thus far there have been no polynomial time quantum attacks on this protocol. In this protocol, created by Jao and De Feo [JDF11], Alice and Bob, rather than picking a multiple of a point, pick an isogeny on the curve. The question then becomes not "What is the discrete logarithm given two points?" but rather "What is the isogeny between two curves?" Note that in this protocol, Alice and Bob both end up with two curves, but the curves may not be the same. Instead, the curves have the same j-invariant, which is $\frac{4a^3}{4a^3+27b^2}$,. They can use this j-invariant as their secret key.

Overall, much of this content is still being researched. It is not known if supersingular isogeny key exchange is resistant to quantum attacks as it is still a fairly fresh idea. It is also important to note that quantum computers are not nearly extensive enough to actually perform an attack on a practical protocol yet. Still, based on this vulnerability there has been some backlash against elliptic curves. Most notably, the NSA began recommending to their security groups not to switch their systems to utilizing elliptic curves. Thus at this point, research efforts ideally should focus on creating protocols that will likely be secure in a post-quantum world.

References

[Bac06] Dave Bacon. Cse 599d - quantum computing the quantum fourier transform and jordan's algorithm. https://courses.cs.washington.edu/courses/cse599d/06wi/lecturenotes9.pdf, 2006. [Online; accessed May 8, 2018].

[BL95] Dan Boneh and Richard J. Lipton. Quantum cryptanalysis of hidden linear functions (extended abstract). In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '95, pages 424–437, London, UK, UK, 1995. Springer-Verlag.

[BSS99] Ian F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, New York, NY, USA, 1999.

[Har60] Bernard Harris. Probability distributions related to random mappings. *Ann. Math. Statist.*, 31(4):1045–1062, 12 1960.

[Hir10] Mika Hirvensalo. *Quantum Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2010.

[JDF11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Proceedings of the 4th International Conference on Post-Quantum Cryptography*, PQCrypto'11, pages 19–34, Berlin, Heidelberg, 2011. Springer-Verlag.

[MEES01] E. Mohammed, A. E. Emarah, and K. El-Shennawy. Elliptic curve cryptosystems on smart cards. In *Proceedings IEEE 35th Annual 2001 International Carnahan Conference on Security Technology (Cat. No.01CH37186)*, pages 213–222, Oct 2001.

[Mra07] Yassine Mrabet. Eccline-3svg. https://en.wikipedia.org/wiki/Elliptic_curve#/media/File:ECClines-3.svg, 2007. [Online, accessed May 5, 2018].

[NIS05] NIST. Recommendation for key management—part 1: general, special publication 800-57. 2005.

[Pol78] J.M. Pollard. Monte carlo methods for index computation mod p). 32, 07 1978.

[Row] Todd Rowland. Elliptic discriminant. http://mathworld.wolfram.com/images/eps-gif/DiscriminantEllipticCurve_1000.gif. [Online, accessed May 5, 2018].

[Sch95] René Schoof. Counting points on elliptic curves over finite fields. *Journal de théorie des nombres de Bordeaux*, 7(1):219–254, 1995.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[Sil09] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Springer-Verlag New York, 2nd edition, 2009.

[Sup07] SuperManu. example elliptic curves. https://en.wikipedia.org/wiki/Elliptic_curve#/media/File:ECClines.svg, 2007. [Online, accessed May 5, 2018].

[Tre18] Trenar3. Q fourier nqubits. https://en.wikipedia.org/wiki/Quantum_Fourier_transform#/media/File:Q_fourier_nqubits.png, 2018. [Online, accessed May 10, 2018].

[WYMAR15] Wienardo, Fajar Yuliawan, Intan Muchtadi-Alamsyah, and Budi Rahardjo. Implementation of pollard rho attack on elliptic curve cryptography over binary fields. *AIP Conference Proceedings*, 1677(1):030019, 2015.