THE DISCRETE LOG PROBLEM AND ELLIPTIC CURVE CRYPTOGRAPHY

NOLAN WINKLER

ABSTRACT. In this paper, discrete log-based public-key cryptography is explored. Specifically, we first examine the Discrete Log Problem over a general cyclic group and algorithms that attempt to solve it. This leads us to an investigation of the security of cryptosystems based over certain specific cyclic groups: $\mathbb{F}_p$, $\mathbb{F}_p^\times$, and the cyclic subgroup generated by a point on an elliptic curve; we ultimately see the highest security comes from using $E(\mathbb{F}_p)$ as our group. This necessitates an introduction of elliptic curves, which is provided. Finally, we conclude with cryptographic implementation considerations.

CONTENTS

1. Introduction

In this paper, basic knowledge of number theory and abstract algebra is assumed. Additionally, rather than beginning from classical symmetric systems of cryptography, such as the famous Caesar or Vignière ciphers, we assume a familiarity with these systems and why they have largely become obsolete on their own. Rather, we focus on non-classical, modern public-key systems. The reader is most likely actually aware of at least one of these, the RSA cryptosystem developed by Ron Rivest, Adi Shamir and Leonard Adleman. We provide a reminder of how this system, used for transmitting a private message securely over a public channel, works.

RSA Suzy wants to be able to receive private messages securely, so she picks two secret large prime numbers, p, q and multiplies them together to get n. Then, with knowledge of p and q, she can calculate $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1)$, allowing her to pick a random secret number, $1 < e < \varphi(n)$ coprime to $\varphi(n)$. This also allows her to use the Extended Euclidean Algorithm to find $d := e^{-1} \pmod{\varphi(n)}$. Then, (n, e) is released to the public while $(p, q, \varphi(n), d)$ is kept secret. Anyone who wants to send her a message can then represent their message as $m < n$ and encrypt it as $c := m^e$, sending c to the Suzy, who recovers it by computing $c^d \equiv (m^e)^d \equiv m \pmod{n}$.

This is a secure system as finding d to decrypt the cipher text is computationally difficult without knowledge of $\varphi(n)$, which would require factoring n to find. This is public cryptography's key idea – it must be computationally much easier to, say, encrypt a message than it is to decrypt it. This leads us to the following definitions.

Definitions 1.1. A **trapdoor function** is a function, $f : X \longrightarrow X$ such that for all $x \in X$, it is easy to compute $y = f(x)$ but difficult to compute $x = f^{-1}(y)$ without some secret information. We call computing $f(x)$ a **forward computation** and computing an inverse, $f^{-1}(y)$, a **backward computation**.

In this paper, we define the security of a system based on the computational difficulty of inverting its trapdoor function. We formalize this by considering the running time with respect to the size of the function's input or 'key size.'

Definitions 1.2. We say a cryptosystem is **most secure** if the lowest-order known algorithm for doing backwards computations on its trapdoor function runs in exponential time, **moderately secure** if it runs in sub-exponential time, and **least secure** if it runs in polynomial time with respect to the size of the input.

With this definition of security based on run-time order, the more secure our system is, the more computation it takes to break the intended secrecy of the system. That is, for a given excessive amount of time one wants the backwards computation to take, the more secure a system is, the less time its forward computation would take.

Example 1.3. Consider, for example, wanting to make an attempted backwards computation take on the order of an infeasibly high 2^{2048} steps. Let's say our system is most secure, with the best-known algorithm for backwards computations taking $O(2^n)$ steps. We will also assume our forwards computations take $O(n)$ time, although they could run in any polynomial time amount. Then we can do this by having our input size, n, be 2048 bits and still perform our forward computations on the order of 2048 steps, a very feasible number.

However, we might want a more quantitative measure of the security of our systems, which we provide now, following [Blake, p. 8].

Definition 1.4. We define the function $L_p(v, c) := \exp(c(\ln p)^v(\ln \ln p)^{1-v})$

Thus, we can characterize our algorithms as taking time proportional to this function with specific values of v and c; those attack algorithms with smaller v values result in less secure systems.

Remark 1.5. Note that cryptosystems with backwards computations of v-value 1 are most secure as breaking them thus takes time exponential in p, while those with v-value 0 are least secure as breaking them thus takes time just polynomial in p. An intermediate value $0 < v < 1$ characterizes algorithms with $L_p(v, c)$ sub-exponential and super-polynomial in p, making them moderately secure cryptographically.

With this set-up, we now investigate public-key cryptosystems based on discrete group exponentiation as a trapdoor function; this consists of comparing the relative security of these systems in Section 3 and, in Section 4, investigating the hardness of backwards computations to find the discrete log in a general group. Then, we consider the specific cyclic groups one might implement these systems over, including elliptic curves, which are probably unfamiliar and we introduce in Section 5. Finally, we come to a conclusion of which is most secure and begin to discuss how to implement these systems in Section 6. We thus provide only a brief introduction to elliptic curve cryptography, leaving out such topics as pairings.

2. Public-Key Systems Over a Cyclic Group G

We look at two cryptosystems in this section, one which provides a secure messaging channel over a public medium, and one which creates a secret shared key for further messaging using a private key system. We keep both of these as general as possible, considering just an abstract cyclic group G. We will later focus on the multiplicative group of a prime-order finite field and the cyclic subgroup generated by a point on an elliptic curve, so we will keep them in mind these next few sections.

2.1. ElGamal Messaging.

The ElGamal system is a way in which two parties, Robert and Suzy, can exchange a secret message over a public channel without prior contact, similar to RSA. This is done by creating a public encryption key and private decryption key, so the message can be encrypted to be unreadable before being sent and can only be decrypted and read by the intended recipient, Robert.
Setup Robert chooses a cyclic group, G, and finds a generator g of G. Next, he choose a secret integer, s and computes $S = g^s$. Only s is kept a secret, while (G, g, S) is broadcast to the public.
Encryption Suzy first chooses one of a slew of methods to encode her message as an element of the group, $m \in G$. Then, she chooses a secret integer, c, and computes both $r := g^c$ and $e := mS^c$. The number c is kept secret, while (r, e) is sent to Robert.
Decryption Robert now retrieves the message m by calculating er^{-s}:

$$er^{-s} = mS^c r^{-s} = mS^c(g^c)^{-s} = mS^c g^{-sc} = mg^{sc}g^{-sc} = mg^0 = m \tag{2.1}$$

Note this message is kept secret as decryption required use of the secret key s.

2.2. **Diffie-Hellman Key Exchanges.**

The Diffie-Hellman Key Exchange is a protocol by which two parties, Robert and Suzy, can establish a shared secret through a public medium by swapping secretly-generated information. This shared secret is then often transformed using some agreed-upon procedure into the key for a symmetric cipher so they can engage in long-term secure communication.

Setup Robert and Suzy agree upon a cyclic group, G, and an element, g, that generates it.
Party 1 Robert chooses a secret number, $s_1 < \#G$, and calculates $S_1 := g^{s_1}$. He then sends S_1 to Suzy, keeping s_1 secret.
Party 2 Suzy, in turn, chooses a secret number, $s_2 < \#G$, and calculates $S_2 := g^{s_2}$. She sends back S_2, keeping s_2 secret.
Shared Secret Finally, Robert uses s_1 to calculate $S_{12} := S_2^{s_1}$ while Suzy uses s_2 to calculate $S_{21} := S_1^{s_2}$. We note that these two values are equivalent.

$$S_{21} = S_1^{s_2} = (g^{s_1})^{s_2} = g^{s_1 s_2} = g^{s_2 s_1} = (g^{s_2})^{s_1} = S_2^{s_1} = S_{12} \tag{2.2}$$

Note that Robert and Suzy have thus come up with a shared secret, as s_1 or s_2 was required to compute the shared secret.

3. Security & Hardness of the Discrete Log Problem

These two cryptosystems, one might have noticed, were largely based off assuming the answer to the following question is "hard".

Question 3.1. *Given a cyclic group, G, a generator, g, of G, and an element $h = g^x$, how hard is it find x ?*

The problem of finding this x is known as the Discrete Logarithm Problem, and it is the basis of our trapdoor functions.

Definition 3.2. Let g be a generator of G. Let $h \in G$. The **discrete logarithm of h**, $L_g(h)$, is defined to be the element of $\mathbb{Z}/(\#G)\mathbb{Z}$ such that

$$g^{L_g(h)} = h$$

Thus, we can think of our trapdoor function as the following isomorphism:

$$E_g : \mathbb{Z}/(\#G)\mathbb{Z} \longrightarrow G$$
$$E_g(x) = g^x$$

So, the question becomes how hard is it to find the backwards isomorphism,

$$L_g : G \longrightarrow \mathbb{Z}/(\#G)\mathbb{Z}$$
$$L_g(g^x) = x$$

We investigate the security of these systems in relation to this trapdoor function.

First, we wonder whether our cryptosystems are in fact actually as hard to break as solving the Discrete Logarithm Problem. We show the security of the systems is properly defined to be in terms of our above discrete group exponentiation function.

Theorem 3.3. *If solving the Discrete Logarithm Problem is easy, the ElGamal and Diffie-Hellman systems can easily be broken.*

To show this claim, we first introduce a way to model such a solution.

Definition 3.4. An **oracle** is a theoretical constant-time "black box" function. We say a 'call' to an oracle is a use of the function on a specified input, giving us our desired output.

We often use the idea that we have an oracle to show rough computational equivalence between two different problems. For example, if we can solve the first problem in polynomial time using an oracle for the second problem, then finding a polynomial-time algorithm to solve the second problem instantly results in finding a polynomial-time algorithm to solve the first problem.

Examples 3.5. We will consider the following oracles to break our systems.
Discrete log oracle:

$$DL_g : G \longrightarrow \mathbb{Z}/(\#G)\mathbb{Z} \text{ where } DL_g(g^x) = x$$

ElGamal oracle:

$$EG_g : G \times G \times G \longrightarrow G \text{ where } EG_g(r, S, e) = m$$

where r, s, e, m are as in Subsection 2.1.
Diffie-Hellman oracle:

$$DH_g : G \times G \longrightarrow G \text{ where } DH_g(S_1, S_2) = S_{21}$$

where S_1, S_2, S_{21} are as in Subsection 2.2.

We can now show the proposition, proceeding in the exact way suggested above.

Proof of Theorem 3.3. Assume we have a solution, DL_g to the Discrete Log problem.

We calculate $s = DL_g(S)$ using a call to our oracle. Then we can calculate $m = er^{-s}$ and break ElGamal.

To break Diffie-Hellman, we calculate $s_1 = DL_g(S_1)$ using a call to our oracle and can then calculate $S_{21} = S_2^{s_1}$. □

The discrete log problem's hardness is thus an appropriate security measure.

A related question one might naturally have is to wonder what the relative security of these two systems is.

Proposition 3.6. *Solving the Diffie-Hellman system is equivalent to solving the ElGamal system.*

Proof. ($\Longrightarrow$) First, assume we have a Diffie-Hellman oracle, DH_g. We calculate $g^{sc} = DH_g(r, S)$ and invert this and multiply by e to find the ElGamal message $eg^{-sc} = er^{-s} = m$.

($\Longleftarrow$) Now, assume we have a ElGamal oracle, EG_g. We calculate $S_2^{-s_1} = EG_g(S_1, S_2, 1)$ and invert this to find the shared secret. □

Thus, these two cryptosystems seem equivalently secure. However, the following question, if the converse to Theorem 3.3 holds, remains open:

Question 3.7. *Does breaking the ElGamal system or solving the Diffie-Hellman problem solve the Discrete Log Problem?*

This is an important question because a positive answer, combined with Theorem 3.3, shows that solving the Discrete Log Problem is as hard as breaking the ElGamal and Diffie-Hellman systems. Essentially, this guarantees that there does not exist an unknown surprise algorithm that solves these systems in an easier way than solving the known-to-be hard problem of discrete logs [Washington, p. 171].

We summarize the state of research on this topic: "It is believed for most groups in use in cryptography that the DHP and DLP are equivalent," [Blake, p. 3] The biggest step towards the answering of this problem is due to the Maurer Reduction first proposed in 1994. Essentially, Maurer, and later work of Wolf and Boneh showed that they are equivalent in 'most' groups assuming certain conditions [Blake, ch. IX.4]. For more on this theorem, which essentially uses a D-H oracle on points of an elliptic curve to get the discrete log, it is suggested the reader goes to see [13] and [14]. Further, for a more current summary of the state of this problem, see [6].

Now that we have considered the relative hardness of these problems and the security of their underlying systems, we now move to attacks on the discrete logarithm problem that could possibly make them insecure, by making it computationally feasible to crack them. We see that they are in fact super-polynomial and discuss conditions to counter-act these attacks to retain better security.

4. Algorithms for Solving the Discrete Log Problem

In this section, we investigate attempts to solve the Discrete Log Problem, leading towards finding the current computationally-best algorithms for performing the backwards computation,

$$L_g : G \longrightarrow \mathbb{Z}/(\#G)\mathbb{Z} \text{ where } L_g(h) = L_g(g^x) = x$$

4.1. **General Cyclic Groups.**

We first look at algorithms that work for any general cyclic group, G.

The first most obvious way to attack the discrete logarithm problem would be brute-force guessing,

Algorithm 4.1 (Brute-Force Guessing)**.** Compute g^i for all $i \in \mathbb{Z}/(\#G)\mathbb{Z}$, stopping when we get a match with h.

On average, one must compute $\#G/2$ powers of g; this algorithm is clearly exponential in the input as we can represent each member of G using $\log_2(\#G)$ binary digits. Thus, if this were the best-known algorithm for finding the DLP, then our systems would be most secure.

Those familiar with the classic "Birthday Problem" might instead suggest making two lists of size on the order of $N \approx \sqrt{\#G}$.

Algorithm 4.2 (Birthday Attack)**.** Compute list one of multiples

$$g^i \text{ for } i = 1, ... \sqrt{N}$$

and list two of terms

$$hg^{-j} \text{ for } j = 1, ..., \sqrt{N}$$

Then, with high probability, there will be a match, that is there will exist i_o and j_o such that we have

$$g^{i_o} = hg^{-j_o} \Rightarrow h = g^{i_o} g^{j_o} = g^{i_o + j_o}$$

and thus we have our discrete log, $x = i_o + j_o$. This stops in $O(\sqrt{N})$ steps; our v value has gone down. However, we note this process only works with high probability [Kolster, ch. 5.6]. This property of producing a correct output with some probability and no output otherwise makes it a so-called 'probabilistic algorithm.'

However, this idea can be adapted into a 'deterministic' process, meaning it always produces a match which leads to the correct output, and thus has worst-case complexity $O(\sqrt{\#G})$, about the same as Algorithm 4.2's average complexity.

Algorithm 4.3 (Baby Step, Giant Step)**.** First, one picks $m \geq \sqrt{\#G}$ and computes $M := g^m$. Then, one can compute and store

$$\text{List 1: } \{g^i | 0 \leq i \leq m\}$$

Similarly, one can compute and store

$$\text{List 2: } \{hg^{-jm} | 0 \leq j \leq m-1\}$$

Note $m \geq \sqrt{\#G}$ and for all x, $x \leq \#G - 1$, so,

$$0 + 0m = 0 \leq k \leq \#G - 1 \leq m^2 - 1 = (m-1) + (m-1)m$$

Therefore there exist $a, b < m$ such that $x = a + bm$. So, we can write $g^x = g^a g^{bm} \Rightarrow g^x g^{-bm} = g^a$. So, since g^a is in List 1 and $g^x g^{-bm} = hg^{-bm}$ is in List 2, we always have a match between our two lists and can find $x = a + bm$.

This is a definite improvement on guess-and-check, having $O(\sqrt{\#G})$ rather than $O(\#G/2)$ runtime yet also being guaranteed to eventually find our discrete logarithm. However, it is still exponential, so we try another algorithm.

We now present an algorithm with the same worst-case complexity, but which can do much better in certain cases. Afterwards, we will see what these cases are and consider how to avoid them.

Algorithm 4.4 (Pohlig-Hellman)**.** This algorithm makes use of the knowledge of the group order and its prime factorization, $\#G = \prod_i q_i^{e_i}$ to find the discrete logarithm, x. This is done by finding the discrete logarithm x modulo each prime power and using the Chinese Remainder Theorem to recombine them.
Prime power expansion Let q be one such prime factor and q^e be the maximal power of that prime that divides $\#G$, i.e. $q^e \mid (\#G)$ but $q^{e+1} \nmid (\#G)$. We can write x in terms of powers of q with $0 \leq x_j < q$ as follows:

$$x \equiv x_o + x_1 q + ... + x_{e-1} q^{e-1} \pmod{q^e}$$

Finding x_o We first compute the list of multiples

$$M := \left\{ g^{r\frac{\#G}{q}} | 0 \leq r < q \right\}$$

We then compute $h^{\#G/q}$, which we note is in M, because

$$h^{\#G/q} = g^{x(\#G)/q} = g^{(x_o + x_1 q + ... + x_{e-1} q^{e-1})\#G/q} = g^{x_o(\#G)/q}$$

as $g^{\#G} = 1_G$, the group identity. We have thus found x_o as the r that produces this match.

Finding x_j **given** x_{j-1} Now, assume that we have found x_{j-1} and still need to calculate x_j for $j \leq e-1$
Let $h_o := h$ and recursively define $h_j := h_{j-1} g^{-x_{j-1} q^{j-1}}$. We then compute $h_j^{\#G/q^{j+1}}$ and once again find this is an element of M as

$$h_j^{\#G/q^{j+1}} = g^{(x_j + x_{j+1} q + ... + x_{e-1} q^{(e-1)-j})\#G/q^{j+1}} = g^{x_j \#G/q^{j+1}}$$

Recombination So, we can find all x_j for each q and thus, by the Chinese Remainder Theorem, we can find x.

As this algorithm computes a list of q elements, M, for each prime, it will run faster if each such q is small and slower if there is a large q. That is, this algorithm runs in $O(\sqrt{l})$ time, where $l := \max\{q|q$ is a prime dividing $\#G\}$, meaning we want the maximum prime factor to have a similar order of magnitude as $\#G$. If we can choose a prime-order group, then this has the same run-time as Algorithm 4.3.

As such, finding the order of the group becomes very important, as knowing the order of the group is necessary to ensure that Algorithm 4.4 runs slowly. For example, if we are working over $\mathbb{F}_p^\times$, we can choose a p such that $p = 2q + 1$, where q is also a prime to guarantee that there exists a subgroup of large prime order q which we can choose as our G.

Below, we present a probabilistic algorithm for calculating the discrete log with expected run-time $O(\sqrt{\#G})$, but which requires minimal storage. Additionally, it works well partially due to attempting to keep the Pollig-Hellman algorithm as slow as possible by choosing the group to be of large prime order.

First, we define a function f that acts randomly, but gives us useful information when two inputs get mapped to the same output. Below, we use what is essentially a random walk among points in the group. For other suitable functions, including analyses of their running times, see [17].

Algorithm 4.5 (Pollard's Rho Method)**.** To define our f, we partition G into several approximately equal cells.

$$G = \coprod_i C_i$$

We then choose a step for each cell by randomly picking two integers less than $\#G$, c_i, d_i, for each cell such that

$$S_i := g^{c_i} h^{d_i}$$

Thus, our random walk function, $f : G \to G$ is given by

$$f(E) = ES_i \text{ if } E \in C_i$$

Finally, we start our random walk at random element $E_0 := g^{g_0} h^{h_0}$ where g_o and h_o are random integers less than $\#G$. We define g_i and h_i as the sum of the exponents on g and h, respectively, 'picked up' along the way such that after each step $E_i = g^{g_i} h^{h_i}$. We continue this random walk until there exists $j \neq k$ such that $E_j = E_k$. When this occurs, we have

$$g^{g_j} h^{h_j} = g^{g_k} h^{h_k} \Rightarrow g_j + xh_j = g_k + xh_k$$

Thus, assuming that $\gcd(h_k - h_j, \#G) = 1$, we have

$$(g_j - g_k)(h_k - h_j)^{-1} = x$$

Since $\#G$ can be assumed to be prime to prevent against the Pohlig-Hellman attack, the probability that $\gcd(h_{k_o} - h_{j_o}, \#G) = 1$ is high [Washington, p. 149].

However, we have avoided the question of how to detect such a match. As we claimed this algorithm normally takes $\sqrt{\#G}$ steps to terminate, storing each point would lead to $\sqrt{\#G}$ storage being taken up. We can dramatically reduce the amount of storage needed by making the observation that if $E_j = E_k$, then for all l, $E_{j+l} = E_{k+l}$. Thus, we can, say, compute pairs (E_i, E_{2i}) until we get a matching pair. The extra computation required for this is only a constant times it would be if we stored all points, and thus runtime remains $O(\sqrt{\#G})$ while storage has been reduced to a constant 2, a truly negligible amount [Washington, p. 148].

So, despite our choice of prime group order to defend against the Pollig-Hellman algorithm being exploited by the Pollard Rho algorithm, we can still say that over an arbitrary cyclic group, our cryptosystems are most secure because we have not found a sub-exponential algorithm that solves the discrete logarithm problem.

However, this does not mean that sub-exponential algorithms for any cyclic groups do not exist, just that there is not a sub-exponential algorithm that applies to all cyclic groups. In fact, while all cyclic groups of order $\#G$ are isomorphic to $\mathbb{Z}/(\#G)\mathbb{Z}$, the ease of finding and computing the backwards function L_g varies considerably for different groups. Thus group choice in cryptography is crucial.

Example 4.6. Let $G = \mathbb{Z}/p\mathbb{Z}$ for some prime p. Then any non-zero element g is a generator. Given any $h = xg$, we can quickly find $L_g(h) = x$ by first using the Extended Euclidean Algorithm to find g^{-1} and then calculate $hg^{-1} = x$.

We now consider the case of the multiplicative group of a finite field of prime order, as historically this has been the group used in cryptographic applications.

4.2. **The cyclic group $\mathbb{F}_p^\times$.** As one might have guessed, much research has been done into attempting to crack this system by finding sub-exponential or even polynomial-time discrete log algorithms, and progress has indeed been made. Below, we reproduce one of the best-known and fastest algorithms for solving the discrete log problem in sub-exponential time. While the running time of the algorithm actually depends on the exact set-up, a testament to its power is that in 2001, it was successfully used to find discrete logs modulo a 120-digit prime [Washington, p. 145] and similar algorithms have since been used to find a discrete log modulo a 180-digit prime [10].

Algorithm 4.7 (Index Calculus)**.** Let p be a prime and g a generator of $\mathbb{F}_p^\times$. As before, the following method finds x where $g^x = h \in \mathbb{F}_p^\times$.
Relations First, choose a limiting size, N and create the "factor base" $F = \{f_i | f \leq N \text{ is a prime}\}$. Similarly, choose a limiting size, R, and for $x = 1, 2, ..., R$ find relations of the form

$$g^x \equiv \pm \prod_{f \in F} f^{e_x(f)} \text{ (mod p)}$$

where $e_x(f)$ are the proper exponents to express g^x in terms of elements of F.

Calculations Now, because of logarithm properties, we can change these relations into the form

$$x \equiv \sum_{f \in F} e_x(f) L_g(f) \text{ (mod p-1)}$$

Then, we do heavy linear algebra computation to solve this system of equations. This results in having access to the complete set $\{L_g(f) | f \in F\}$, our "index table."
Finding the Discrete Log Now, to solve $g^{L_g(h)} = h$ for $L_g(h)$ we compute $g^{j_i} h$ until there exists j_o such that we get a relation of the form

$$g^{j_o} h \equiv \pm \prod_{f \in F} f^{e_{j_o+x}(f)} \text{ (mod p)}$$

where $e_{j_o+x}(f)$ are as above. This gives us

$$L_g(h) \equiv \sum_{f \in F} e_{j_o}(f) L_g(f) - j_o \text{ (mod p-1)}$$

and thus we have found our discrete logarithm.

The key idea here is that since we are working in $\mathbb{F}_p^\times$, we can write elements uniquely as a product of primes, and attackers have found fast algorithms for factoring integers in terms of powers of primes to find the types of relations above to create our index table. For example, the number field sieve is an extremely fast algorithm to produce new relations of that form which is used in most of the lowest-order algorithms for finding discrete logarithms in $\mathbb{F}_p^\times$ [Washington, p. 145].

We note that in the notation of Definition 1.4, $v = 1/2$ for this algorithm and, in fact, Antoine Joux has recently proposed algorithms with v values as low as $1/4$ [12] and a "quasi-polynomial" algorithm in May 2014, found in [4]. Thus, cryptosystems over $\mathbb{F}_p^\times$ are quite less secure than our abstract group analysis might have suggested.

So, these perhaps natural choices for cyclic groups turn out to not be the most secure, leaving us to search for other such groups. What we would most like is to find a cyclic group whose elements we can easily represent in a computer and on which we can easily perform forwards operations, but does not have a sub-exponential algorithm for solving its discrete logarithm problem. This way, our cryptosystems can be as secure as possible and prevent against these attacks without the key size (the input size in Definitions 1.2) being too large. We care about key sizes because, returning to Example 1.3, we note that because we are only considering the big O order of our algorithms, we might run into unwanted multiplicative constants or high exponents on the polynomials that characterize our running times. If such a thing happens, then increasing the key size to large values might actually make it less feasible to compute our forward operations. This increasingly becomes a problem as the average computational power available to technology that has to implement these systems decreases; for example, smart phones with far less processing power than normal computers might want to implement a Diffie-Hellman exchange in order to browse the Internet securely.

Neal Koblitz and Victor Miller independently suggested using the group of points on elliptic curves, frequently-studied objects in number theory [Blake, p. 7].

4.3. **Subgroup Generated by a Point on an Elliptic Curve.** It seems that little to no progress has been made into finding a sub-exponential algorithm for solving the Elliptic Curve DLP. Most of these attempts have focused on trying to find an analogue to performing index calculus methods on the curve, which has turned out to fail and can be read about in [5] and [9]. Another method that has been attempted is Semaev's "summation polynomials", which has produced a sub-exponential algorithm (but only for the case of $\mathbb{F}_{2^n}$, not for $\mathbb{F}_p$), has a v value of $2/3$ [15], which is higher than the v values for the finite field discrete log problem, and only works well for key sizes much larger than are currently implemented [16]. Yet, this is an active research area and such algorithms theoretically could be found.

As an example of how elliptic curves thus satisfy what we want from a group to base our cryptosystems on, consider the following approximate formula for the relative security of using $\mathbb{F}_p^\times$ vs. $E(\mathbb{F}_q)$ given on [Blake, p. 9]:

$$\log_2 q \approx 5(\log_2 p)^{1/3}(\ln(\log_2 p)\ln 2))^{2/3}$$

This means, for example, that a traditional key size of 1024 bits is approximately as secure as a key size of 173 bits for elliptic curves and a traditional key size of 2048 bits is approximately as secure as a key size of 313 bits for ECC [Blake, p. 9].

As it seems this group is most safe, we now provide an introduction to it.

5. Elliptic Curves: A Better Group for Cryptography

5.1. **Basic Theory and Group Law.** There are multiple ways to define an elliptic curve; for our purposes in studying cryptography, we follow Kolster and Washington's expositions using the Weierstrass Equation definition.

Definition 5.1. An **elliptic curve** over a given field, F, is a curve $y^2 = x^3+Ax+B$ where A, B are in F and $-16(4A^3 + 27B^2) \neq 0$. For L/F, a field extension, we define $E(L)$ to be the set of equivalence classes given by ratios of solutions $(x : y : z)$ to the equation $y^2z = x^3 + Axz^2 + Bz^3$. This is the set $E(L) := \{(x, y) \in L^2 | y^2 = x^3 + Ax + B\} \cup \{\infty\}$.

We note that $y^2z = x^3+Axz^2+Bz^3$ is the projective version of $y^2 = x^3+Ax+B$, so their sets of solutions (x, y, z) to the first and (x, y) to the second are naturally related. Due to the homogeneity of the polynomials in the first equation, for all t, (tx, ty, tz) is also a solution, so instead we consider ratios of solutions $(x : y : z)$. There is a clear bijection between our set $E(L)$ and solutions to this equation. The points (x, y) map to $(x : y : 1)$ and the point ∞ maps to $(0 : y : 0)$. Now, surprisingly, this set constitutes a group with the operation given below.

First, we present a nice, slightly informal, geometric interpretation of this operation: To add two points in the elliptic curve group, take their secant line and find its intersection with the elliptic curve. The result of their addition is the reflection over the y-axis of this intersection. This definition runs into a few problems when considering adding a point to itself or the point ∞, which we can think of as the top and bottom of the y-axis, so we formalize it below.

Group Law Let P, Q be points in our set. We define their sum, R, as follows:
1) If $P = \infty$, $R = Q$ and if $Q = \infty$, then $R = P$.
Now, assume neither P nor Q is ∞, so we represent $P = (x_1, y_1)$ & $Q = (x_2, y_2)$.
2) if $x_1 = x_2, y_1 \neq y_2$, then $m := \infty$, which gives $R = \infty$.
3) if $P = Q, y_1 = 0$, then $m := \infty$, which gives $R = \infty$.
Else, we can represent $R := (x_3, y_3)$ as follows.
4) if $x_1 \neq x_2$, then $m := (y_2 - y_1)/(x_2 - x_1)$ which gives
$x_3 = m^2 - (x_1 + x_2), \quad y_3 = m(x_1 - x_3) - y_1$.
5) if $P = Q, y_1 \neq 0$, then $m := (3x_1^2 + A)/2y_1$, which gives
$x_3 = m^2 - 2x_1, \quad y_3 = m(x_1 - x_3) - y_1$.

We note this operation is a binary relation on $E(F)$ as $\infty, x_i, y_i, A, B \in F$ and we only perform rational functions on these.

Theorem 5.2. *If F is finite, $E(F)$ is a finite Abelian group.*

Proof. We give a sketch of a proof of this result.
Finiteness There are only a finite number of possible pairs (x, y).
Identity There exists a unique element which satisfies the identity property of our group - for all $P \in E(F)$, $P + \infty := P$ as we defined our operation this way for ∞.
Inverses For all $P \in E(F)$, there exists $-P \in E(F)$, which obviously holds for ∞. For all elements other than the identity, $(x, y) \in E(F) \Rightarrow (x, -y) \in E(F)$ and by 2) above, $(x, y) + (x, -y) = \infty$.
Associativity One could check that the formulas defined above work out, or if one desires a cleaner, more theoretical proof, see [Washington, ch 2.4].
Commutativity This easily follows from the formulas given above. □

5.2. **Finding the Order.** In this section, we briefly state classic results from elliptic curve studies on the group size. We then discuss their application to finding the size of the group, and how knowing these results and being able to find the size of the group allows us to better prevent against attacks by picking the size of the group correctly, as motivated by the Pohlig-Hellman attack.

First, the group structure of $\mathbb{F}_q$-valued points on an elliptic curve is known to have at most 2 generators.

Theorem 5.3. *Either* $E(\mathbb{F}_q) \cong \mathbb{Z}/n\mathbb{Z}$ *or* $E(\mathbb{F}_q) \cong \mathbb{Z}/n_1\mathbb{Z} \oplus \mathbb{Z}/n_2\mathbb{Z}$

The proof of this statement involves another useful concept, $E[n]$.

Definition 5.4. The set of n**-torsion points**, $E[n] := \{R \in E(\overline{\mathbb{F}_q}) | nR = \infty\}$.

Proof. We remember that the Fundamental Theorem of Finite Abelian Groups tells us $E(\mathbb{F}_q) \cong \mathbb{Z}/n_1\mathbb{Z} \oplus ... \oplus \mathbb{Z}/n_r\mathbb{Z}$, with $n_i \mid n_{i+1}$ for $i = 1, ..., r-1$. Suppose that $E(\mathbb{F}_q)$ does have a subgroup of the form $\mathbb{Z}/n_1\mathbb{Z} \oplus ... \oplus \mathbb{Z}/n_r\mathbb{Z}$ with $r \geq 3$. Then, the set of n_1-torsion points in $E(\mathbb{F}_q)$ has at least n_1^r elements. The above set in $E(\mathbb{F}_q)$ is by definition a subset of $E[n_1]$ but this contradicts a lemma that says $\#E[n_1] \leq n_1^2$. So, $r \leq 2$. For the lemma's proof, see [Washington, ch. 3.2] $\square$

The obvious way to find the group order is to list all of the elements, going through all possible x values and seeing if there is a y that satisfies the equation $y^2 = x^3 + Ax + B$. Intuitively, the order of the group is heavily related to the number of squares, which we might guess is $q - 1$. In fact, we can write out the order of the group explicitly using this notion, helped by the Legendre symbol.

Definition 5.5. The **Legendre symbol**, is defined as such.

$$\left(\frac{x}{\mathbb{F}_q}\right) := \begin{cases} +1 & \text{if } \exists t \in \mathbb{F}_q^\times \text{ s.t. } t^2 = x \\ 0 & \text{if } x = 0 \\ -1 & \text{if } \nexists t \in \mathbb{F}_q \text{ s.t. } t^2 = x \end{cases}$$

This gives us

Proposition 5.6.

$$\#E(\mathbb{F}_q) = q + 1 + \sum_{x \in \mathbb{F}_q} \left(\frac{x^3 + Ax + B}{\mathbb{F}_q}\right)$$

Proof. For all $x_o \in \mathbb{F}_q$, if $x_o^3 + Ax_o + B$ is a non-zero square, there exist 2 points on the curve, $(x_o, \pm\sqrt{x_o^3 + Ax + B})$, if $x_o^3 + Ax_o + B = 0$, there is the unique point $(x_o, 0)$, and if $x_o^3 + Ax_o + B$ is not a square, there do not exist any points with that x-coordinate. So, for each x_o, there are $1 + \left(\frac{x_o^3 + Ax_o + B}{\mathbb{F}_q}\right)$ points. There are q such x_o, so adding our one ∞ point, we get the claimed number. $\square$

Doing an analysis of this "error" term, $t := \#E(\mathbb{F}_q) - (q + 1)$, from what one might expect the number of squares to be in $E(\mathbb{F}_q)$ yields the following theorem.

Theorem 5.7 (Hasse's Theorem)**.**

$$(q + 1) + 2\sqrt{q} \geq \#E(\mathbb{F}_q) \geq (q + 1) - 2\sqrt{q}$$

Proof. This is presented without proof, which can be found in [Washington, ch. 4.2]. $\square$

These results are key for our algorithms, as we can now tackle the slightly easier problem of finding the order of just a few points to find the group order.

Theorem 5.3 tells us that there exist at most 2 points whose orders completely determine the order of the group, meaning if we could just find them, we would not have to compute the order of very many points.

Theorem 5.7, Hasse's Theorem, tells us that we have an interval of size $4\sqrt{q}$ that the group order lies within, meaning that, due to LaGrange's Theorem, we can dramatically restrict our candidates for the order of the group. As the order of every element of a group divides the group order, high-order points will have few multiples in that interval. Thus, we only need to find a small number of high-order points or a point, H, with order $ord(H) \geq 4\sqrt{q}$ in order to find the group order.

Due to Hasse's theorem, using just brute-force guessing and checking, finding the group order would run in $O(\sqrt{q})$ time. However, we can once again reduce this using a baby-step, giant-step method, which runs in $O(\sqrt[4]{q})$ time.

Algorithm 5.8 (Shanks-Mestre Baby Step, Giant Step)**.** This algorithm finds ord(P) for $P \in E(\mathbb{F}_q)$ below.

First, we compute $Q = (q+1)P$. Then we pick $m \geq \sqrt[4]{q}$ and compute and store

$$\text{List 1: } \{iP | 1 \leq i \leq m\}$$

Similarly, one can compute and store

$$\text{List 2: } \{Q + k(2mP)| - m \leq k \leq m\}$$

until we find a match, that is, there exist $\imath_o$ and k_o such that we have

$$Q + k_o(2mP) = \pm i_o P$$

which gives us

$$(q + 1 + 2mk_o \mp i_o)P = \infty$$

Then, taking this 'killing multiple' $M := q+1+2mk_o \mp i_o$, we can look at its factors $p_1 p_2 ... p_r = M$ to find the true order by computing $\frac{M}{p_i}P$ for all factors, redefining M by dividing out by the p_i such that $\frac{M}{p_i}P = \infty$ until we are left with $ord(P)$.

While there are improvements to the above algorithm, we ignore them and instead present a much faster algorithm for finding an order of a point. We briefly sketch this algorithm below, providing the necessary background on the theory of the Frobenius endomorphism and specially-defined division polynomials that help in detecting roots.

Definitions 5.9. The **Frobenius Endomorphism**, $\varphi : \mathbf{E}(\overline{\mathbb{F}_\mathbf{q}}) \to \mathbf{E}(\overline{\mathbb{F}_\mathbf{q}}), (\mathbf{x}, \mathbf{y}) \mapsto (\mathbf{x}^\mathbf{q}, \mathbf{y}^\mathbf{q})$ is a map which, with $t := \#E(\mathbb{F}_q) - (q + 1)$, has characteristic equation

$$\varphi^2 - t\varphi + q = 0 \tag{5.10}$$

Polynomials denoted $\psi_\mathbf{m}, \mathbf{\Phi_m}, \omega_\mathbf{m} \in \mathbb{F}[x, y]$ have been found which represent an algebraic expression for the coordinates of point multiplication. That is, if we let $P = (x, y)$ be an element of $E(\overline{\mathbb{F}_q})$, then we can express the results of adding P to itself n times according to the group law given above as

$$nP = (\frac{\Phi_n(x)}{\psi_n^2(x)}, \frac{\omega_n(x, y)}{\psi_n(x, y)})$$

In particular, the $\mathbf{m^{th}}$ **division polynomial**, $\psi_{\mathbf{m}}$, satisfies the property

$$\psi_m(x) = 0 \Leftrightarrow P \in E[m]$$

For why this is true and explicit formulas for these polynomials, see [Washington, ch. 3.2].

Keeping these definitions in mind, we can now understand Blake's presentation of the point-counting algorithm proposed by René Schoof [Blake, ch. VII.I]

Algorithm 5.11 (Schoof's Algorithm)**.** This algorithm runs in $O(\log^8 q)$ steps and finds $\#E(\mathbb{F}_q)$ by finding the 'error term' t such that $\#E(\mathbb{F}_q) = q+1-t$. By Hasse's Theorem, $|t| < 4\sqrt{q}$. We find t mod l_i for $i = 1, ..., r$ the smallest primes such that

$$\prod_{l=1}^{r} l_i > 4\sqrt{q}$$

$l = 2$ **case** In the $l = 2$ case, it is very easy to determine whether t is odd or even.

Since q is an odd prime, $q + 1$ is even and thus $t \equiv \#E(\mathbb{F}_q) \pmod 2$. If there exists a root $r \in \mathbb{F}_q$ such that $0 = r^3 + Ar + B$, then there is a point of order 2 on the curve, which implies that the group has even order and thus $t \equiv 0 \pmod 2$. If there does not exist such a root, then $t \equiv 1 \pmod 2$.

$l > 2$ **case** Let R be a member of the set of l-torsion points, $E[l]$.
By (5.10), we know that for such R,

$$\varphi^2 R - t\varphi R + qR = 0$$

Thus, if we let $q_l :\equiv q$ (mod l) and $t_l :\equiv t$ (mod l), then if we find a $0 \leq \tau < (l-1)/2$ such that

$$\varphi^2 R + q_l R = \pm\tau\varphi R \tag{5.12}$$

then $\pm\tau = t_l$.

We can see $\varphi^2 R + q_l R = t_l \varphi R$, thus we have $(t_l \mp \tau)\varphi R = 0$. Since $ord(\varphi R) = l$, we know $l \mid (t_l \mp \tau)$, and so we conclude $t_l = \pm\tau$.

So, to find t_l, we can try each candidate τ as follows, only paying attention to the x-coordinate and working modulo ψ_l:

One can compute $\varphi^2 R$ directly, while $q_l R$ and $\tau\varphi R$ are rational functions of x, y, ψ_m. Thus, one can use the point-addition formula to compute $\varphi^2 R + q_l R$ symbolically, ending up with an equation $h_\tau(x) = 0$. We can thus use this $h_\tau(x)$ polynomial to determine whether a given value of τ being tried satisfies (5.12).

One can check if $h_\tau(x) = 0$ has a solution with $(x, y) \in E[l]$ by computing $\gcd(h_\tau, \psi_l)$. If their greatest common divisor is 0, then there is no solution $Q \in E[l]$ to (5.12) and we move onto the next τ. If gcd $\neq 0$, then we can say there does exist such a Q. This will allow us to determine what t_l is, as only one such τ will give $\gcd(h_\tau, \psi_l) \neq 0$.

Finally, we use the knowledge of each t_l to find t via the Chinese Remainder Theorem. For greater detail, including, why only one such τ satisfies this property and how to determine, by looking at the y-coordinates, whether we should use $+\tau$ or $-\tau$ as our t_l, see [Washington, ch. 4.5].

We note this huge improvement in running time is due to Algorithm 5.11 only taking $\gcd(h_\tau, \psi_l)$, meaning we can work entirely modulo ψ_l. Thus, our computations are done only on small-order polynomials, for which computations such as Euclid's Algorithm are fast [Washington, p. 124].

Thus we have seen that the order can be quickly computed and therefore we can find curves of safe orders to get around Pohlig-Hellman, once again making elliptic curves a good choice for the underlying cryptographic group.

6. Ensuring Security

Finally, we discuss the practical applications of how to choose an elliptic curve group in order to implement one of our cryptosystems.

In order to make the asymptotic running times of super-polynomial attack algorithms excessive, we are required to choose curves with a subgroup of large prime order which we can work in. Specifically, the National Security Agency's Suite B recommendations state that a 384-bit prime is sufficient for 'top secret information'. Similarly, Certicom, in its Elliptic Curve Cryptography Challenge, still offers $40,000 for cracking a discrete logarithm on a curve defined over $\mathbb{F}_p$ where p is a 191-bit prime and believes it to be out reach of current computational power, estimating it to take 4.8 x 10^{19} days for an Intel Pentium 100 to do so. The National Institute of Standards and Technology recommends that to achieve security on the order of 2^{112} steps, one must choose a prime of size 2048 bits for $\mathbb{F}_p^\times$ while only a prime of size 224 bits for $E(F)$. The data here is taken from [11], which gives a nice summary of current recommendations.

6.1. Special Cases and Notes.

Additionally, there are certain curve forms such as anomalous curves where $\#E(\mathbb{F}_q) = q$, curves with $\#E(\mathbb{F}_q) = q \pm 1$, or curves with $q^s \equiv 1 \pmod{q}$ for some small s which are highly susceptible to attacks specifically designed against them and which should not be used. A good reference for listing such curve types is [7]. The reasons for these specific conditions needing to be met are due to attacks based on the Weil and Tate pairings, which one can read about in [Washington, ch 3.3-3.4, 11.2-11.3]

7. Further Reading

In this paper, we have only examined the basics of elliptic curve cryptography. There is still a wealth of material in elliptic curve cryptography to delve into. I would particularly suggest the curious reader to read about the aforementioned Weil and Tate pairings as new, improved elliptic curve cryptographic methods based on these pairings have recently been developed, such as modifications to the Diffie-Hellman Key Exchange which provides verification of who is sending the messages or are efficient three-party versions. For information on these as well as hyperelliptic curve cryptography, I recommend checking out [8].

8. Acknowledgments

It is a pleasure to thank my mentor Preston Wake for helping me choose a topic and focus on it, even when I wanted to stray, as well as encouraging me to pursue the sub-topics most interesting to me, even when I needed significant help understanding them, which his explanation provided. Similarly, Yun Cheng provided me with helpful comments at our meetings. I would also like to thank Prof. Laszlo Babai for his wonderful Apprentice Program class that truly challenged me to dive into problems whole-heartedly, even if only to make very slight progress towards solutions. Finally, thanks to Peter May for organizing the entire REU program.

References

[1] [Blake] Ian Blake, Gadiel Seroussi, & Nigel Smart. Elliptic Curves in Cryptography. Londom Mathematical Society Lecture Note Series 265, 1999.

[2] [Kolster] Manfred Kolster. Introduction to Cryptography, 2009.

[3] [Washington] Lawrence C. Washington. Elliptic Curves: Number Theory and Cryptography, ed. 2. Discrete Mathematics and Its Applications 50, 2008.

[4] Razvan Barbulescu, Pierrick Gaudy, Antoine Joux, Emmanuel Thom. A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic. Advances in Cryptology, EUROCRYPT 2014, 1-16.

[5] R. Balasubramanian, Neal Koblitz. The Improbability That an Elliptic Curve Has Subexponential Discrete Log Problem under the Menezes-Okamoto-Vanstone Algorithm. Journal of Cryptology, March 1998, Volume 11, Issue 2, 141.145.

[6] K. Bentahar. The equivalence between the DHP and DLP for elliptic curves used in practical applications, revisited. Proceedings of the 10th international conference on Cryptography and Coding, 2005, 376-391.

[7] Daniel J. Bernstein and Tanja Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography. Accessed August 17, 2014 from http://safecurves.cr.yp.to.

[8] Ian Blake, Gadiel Seroussi, & Nigel P. Smart. Advances in Elliptic Curve Cryptography. London Mathematical Society Lecture Note Series 317, 2005.

[9] Claus Diem. On the discrete logarithm problem in elliptic curves II. Algebra & Number Theory 7.6, 2013, 1281-1323.

[10] Discrete logarithm records. Wikipedia, retrieved August 18, 2014 from http://en.wikipedia.org/wiki/Discrete_logarithm_records.

[11] Damien Giry. Keylength. Accessed August 17, 2014 from http://www.keylength.com/en/.

[12] Antoine Joux. A new index calculus algorithm with complexity L(1/4 + o(1)) in small characteristic. Cryptology ePrint Archive, Report 2013/095.

[13] U.M. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. Advances in Cryptology, CRYPTO 94, 271-281.

[14] U.M. Maurer and S. Wolf. Diffie-Hellman oracles. Advances in Cryptology, CRYPTO 96, 333-344.

[15] Cristophe Petit, Jean-Jacques Quisquater. On Polynomial Systems Arising from a Weil Descent. Advances in Cryptology ASIACRYPT 2012 Lecture Notes in Computer Science Volume 7658, 451-466.

[16] Michael Shantz, Edyln Teske. Solving the Elliptic Curve Discrete Logarithm Problem using Semaev Polynomials, Weil Descent and Gröbner Basis Methods An Experimental Study. Number Theory and Cryptography Lecture Notes in Computer Science Volume 8260, 2013, 94-107.

[17] Edyln Teske. Speeding up Pollard's rho method for computing discrete logarithms. Algorithmic Number Theory Lecture Notes in Computer Science Volume 1423, 1998, 541-554