Discrete Logarithm Based Cryptography with Abelian Varieties
Draft March 31, 2015

Contents

1 The Discrete Logarithm Problem

Given an arbitrary finite cyclic group G with group operation $\cdot$ and generator $g \in G$, discrete exponentiation by a in G is defined by

$$g^a = \overbrace{g \cdot g \cdots g}^{a \text{ times}}$$

If $y = g^a$ is known, computing a is called finding the discrete logarithm of y. With the method of fast exponentiation, y can be computed quickly, only $O(\log a)$ group operations. On the other hand, computing a can be much harder. In fact, in [2] it was shown that in an arbitrary group for which only the group operation and discrete exponentiation can be applied to group elements, computing discrete logarithms will take at least $O(\sqrt{|\ G\ |})$ operations. In most cases though, more structure is known about the group in use.

1.1 The Diffie-Hellman Key Exchange

This one-way property of discrete exponention has proven to be very useful for cryptographic purposes. The most notable of these is in the Diffie-Hellman Key Exchange Protocal in which two parties A and B wish to share a secret key k.

Algorithm 1 Diffie-Hellman Key Exchange Protocal

A and B share a publicly known group G and generator g.
A chooses a random private exponent a and computes g^a.
B chooses a random private exponent a and computes g^b.
A sends g^a to B and B sends g^b to A.
A raises g^b to their own private exponent a to obtain $k = (g^b)^a = g^{ab}$.
B raises g^a to their own private exponent b to obtain $k = (g^a)^b = g^{ba}$.

The two parties may now use k to communicate with a cryptographically secure communication protocol. The described protocol relies on the hardness of computing g^{ab} given g^a and g^b, which is conjectured in [3] to be equivalent to computing discrete logarithms.

1.2 The Digital Signature Algorithm

In many web based security protocols, it's important to have a scheme for demonstrating the authenticity of a digital message or document. In 1976, Whitfield Diffie and Martin Hellman described a solution to this problem with concept of *digital signatures*. Further information on digital signatures can be found in [8]. We describe the Digital Signature Algorithm (DSA) which is the US Federal Information Processing Standard for digital signatures.

Let G be a finite cyclic group of order N with generator g. Suppose party A wants to send a signed message to a party B. Finish describing algorithm

Algorithm 2 DSA

1.3 A Brief History of the Groups $\mathbb{F}_p^*$ and $\mathbb{F}_{2^n}^*$

Given a finite field $\mathbb{F}$, the multiplicative units $\mathbb{F}^* = \mathbb{F}\backslash\{0\}$ form a finte cyclic group and thus may be used for discrete logarithm based cryptography. It is a standard result from algebra that every finite field has order p^n, where p is a prime and $n \in \mathbb{Z}^+$. We divide the discussion of the cryptographic properties of the group $\mathbb{F}_{p^n}^*$ into two cases, when $n = 1$ and when $n > 1$.

In the later case, when working with finite fields of order p^n, the arithmetic is only really tractable when $p = 2$. In the 1980's researches at the University of Waterloo made attempts to construct discrete logarithm cryptosystems based on $\mathbb{F}^*_{2^{127}}$ which initially paralelled RSA in terms of bits of security. But in 1986, Don Coppersmith devised an astonishing algorithm in [7] which could compute discrete logarithms in the group $\mathbb{F}^*_{2^{127}}$ in about 5 minutes. Further attempts were made too increase the size to $n = 593$ but similar adaptations of Coppersmith's algorithm made researchers abandone public key cryptosystems based on the discrete logarithm problem in $\mathbb{F}^*_{2^n}$.

When $n = 1$, we have a group which is essentially just the non-zero integers mod a prime. As one might expect, when p is small, computing discrete logs in $\mathbb{F}^*_p$ can be done quickly with just trial exponentiation. When p is large though, say $p = 2^{1000}$, this method becomes completely intractable, even on todays fastest computers. That being said, there is an attack described in [6] which adapts the Number Field Sieve to solve discrete logs in $\mathbb{F}^*_p$. This attack has running time very similar to factoring

$$O(p) = e^{1.923(\log p)^{1/3}(\log\log p)^{2/3}}$$

This basically means that the bit length required for p in $\mathbb{F}^*_{p^n}$ based cryptosystems is the same as the bit length required for the modulus in RSA. In todays standards that mean $p = 2^{2048}$. Although this cryptographically viable, in practice using such large values of p has its limitations. Such as bandwidth in network communications or memory in a hand-held devices.

These two case made researches search for alternative groups with cryptographically strong properties.

2 Abelian Varieties

In this report we focus on groups which arise from the solution set of polynomial equations over finite fields. The most famous of these groups is the set of points on an elliptic curve characterized by the equation $y^2 = x^3+ax+b$ where $a, b \in \mathbb{F}_p$ where $4a^3 + 27b^2 \not\equiv 0 \bmod p$. In recent years, this group has found significant success in public key cryptography even though it's an open questions whether this group is actually cryptographically secure. One of the benefits of using elliptic curves is that the fastest known attack on them has $O(\sqrt{N})$ complexity, where N is the order of the group. This means significantly smaller keys can be used in comparison to the key length of RSA. A natural question is

What about other polynomial equations?

It turns out that there are infinitely many groups which arise from the solution sets of polynomial equations. Which of these groups are cryptographically secure is an active area of research. First we develope some theory required to describes these groups.

Let K be a field. Let $K[x_1, ..., x_n]$ represent the polynomial ring in n variables over K. Given a subset $T \subseteq K[x_1, ..., x_n]$, we may define

$$Z(T) = \{P \in K^n \mid f(P) = 0 \text{ for all } f \in T\}$$

to be the set of all common zeros of polynomials in T. We call $Y \subseteq K^n$ an *algebraic subset* if $Y = Z(T)$ for some subset $T \subseteq K[x_1, ..., x_n]$. We say a algebraic set Y is reducible if it be written as the union of two smaller algebraic sets. For example, let $Y = Z(x^2 - yz, xz - x)$ as a subset of K^3. That is, Y is the set of points in K^3 which satisfy each of the equations $x^2 - yz$ and $xz - x$. Notice that

$$\begin{aligned} Y &= Z(x^2 - yz, xz - x) \\ &= Z(x^2 - yz, x(z-1)) \\ &= Z(x^2 - yz, x) \cup Z(x^2 - yz, z - 1) \\ &= Z(yz, x) \cup Z(x^2 - y, z - 1) \\ &= Z(y, x) \cup Z(z, x) \cup Z(x^2 - y, z - 1) \end{aligned}$$

So Y is reducible. If an algebraic set is not reducible, it is called irreducible.

2.1 Dimension

2.2 Genus

3 Elliptic Curves

Let p be an odd prime and let $E = Z(y^2 - x^3 - ax - b)$. We call E and *elliptic curve* defined over $K = \mathbb{Z}_p$ if $4a^3 + 27b^2 \not\equiv 0 \bmod p$. It was first realized by Need reference Abel and Jacobi in the 1700's that remarkably, the $\mathbb{Z}_p$-rational points of E can be transformed into a group using a very specific group operation. In this section we describe this group operation along with other algorithms needed for cryptographic purposes.

3.1 The Group Operation

Given two $\mathbb{Z}_p$-rational points $P_1 = (x_1, y_1), P_2 = (x_2, x_2)$ which we want to add together, we first define the value

$$s = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} \bmod p & \text{if } P_1 \neq P_2 \\ \frac{3x_1^2 - a}{2y_1} \bmod p & \text{if } P_1 = P_2 \end{cases}$$

Then we define the coordinates of the point $P_3 = P_1 + P_2$ to be

$$\begin{aligned} x_3 &= s^2 - x_1 - x_2 \\ y_3 &= y_1 + s(x_3 - x_1) \end{aligned}$$

It's not immediately obvious that $P_3 = (x_3, y_3)$ is even a point on E and less obvious that this operation satisfies the axioms of a group.

Algorithm 3 The addition of two points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ on an elliptic curve $E : y^2 - x^3 - ax - b$

1: **function** ADD(E,P_1, P_2)
2:     **if** $P_1 = \mathcal{O}$ **then**
3:         **return** P_2
4:     **else if** $P_2 = \mathcal{O}$ **then**
5:         **return** P_1
6:     **else if** $P_1 = P_1$ **then**
7:         $s \leftarrow (3x_1^2 - a)(2y_1)^{-1} \bmod p$
8:     **else**
9:         **if** $x_1 \neq x_2$ **then**
10:             $s \leftarrow (y_1 - y_2)(x_1 - x_2)^{-1} \bmod p$
11:         **else**
12:             **return** $\mathcal{O}$
13:         **end if**
14:     **end if**
15:     $x_3 \leftarrow s^2 - x_1 - x_2 \bmod p$
16:     $y_3 \leftarrow y_1 + s(x_3 - x_1) \bmod p$
17:     **return** $P_3 = (x_3, y_3)$
18: **end function**

An implementation note is that in steps 7 and 10, modular inverses must be calculated. Also note that when $x_1 = x_2$ but $P_1 \neq P_2$, the second coordinates satisfy $y_1 = -y_2$. So the line from P_1 to P_2 is just a vertical line at x_1. This is taken to be the point at infinity $\mathcal{O}$.

3.2 Scalar Multiplication of a Point

For many discrete logarithm protocols (such as Diffie-Hellman of DSA), we require to add point P to itself many times in order to perform discrete exponentiation. That is, given an integer m we need to calculate

$$mP = \overbrace{P + P + \cdots + P} m \text{ times}$$

fast in order to be cryptographically reasonable. The following algorithm does this.

3.3 Finding Points

Now that we have developed arithemtic on and elliptic curve E, then next step is figure out how to find points on E. If $y^2 = x^3 + ax + b$ for $a, b \in \mathbb{Z}_p$, then finding $\mathbb{Z}_p$-rational points on E is equivalent to determining if $x^3 + ax + b$ is a square mod p. This is a classical problem in number theory which can be reformulated to determing the value of the *Legendre Symbol*, which is defined as

Algorithm 4 Scalar multiplication of a point P by an integer m

```
function ScalarMult(m,P)
    if m = 0 then
        return 𝒪
    else if m = 1 then
        return P
    else if m ≡ 0 mod 2 then
        return ScalarMult(m/2,P + P)
    else
        return P + ScalarMult(m − 1,P)
    end if
end function
```

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a square mod } p \\ -1 & \text{if } a \text{ is not a square mod } p \\ 0 & \text{if } p \text{ divides } a \end{cases}$$

where (in our case) $a = x^3+ax+b$. The following five properties let us determine whether a is a square mod p in polynomial time. Let $a, b \in \mathbb{Z}$ and p, q be odds primes.

(i) If $a \equiv b \bmod p$, then $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$

(ii) $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{p}\right)$

(iii) $\left(\frac{-1}{p}\right) = 1$ if $p \equiv 1 \bmod 4$, and $\left(\frac{-1}{p}\right) = -1$ if $p \equiv 3 \bmod 4$

(iv) $\left(\frac{2}{p}\right) = 1$ if $p \equiv \pm 1 \bmod 8$, and $\left(\frac{2}{p}\right) = -1$ if $p \equiv \pm 3 \bmod 8$

(v) If p, q are distinct, then

$$\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right) \text{ if } p \text{ or } q \equiv 1 \bmod 4$$

$$\left(\frac{p}{q}\right) = -\left(\frac{q}{p}\right) \text{ if } p \equiv q \equiv 3 \bmod 4$$

For example, to determine if 105 is a square mod 227, we simply compute

$$
\begin{aligned}
\left(\frac{105}{227}\right) &\overset{\text{(ii)}}{=} \left(\frac{3}{227}\right)\left(\frac{5}{227}\right)\left(\frac{7}{227}\right)\\
&\overset{\text{(v)}}{=} (-1)\left(\frac{227}{3}\right)\left(\frac{227}{5}\right)(-1)\left(\frac{227}{7}\right)\\
&\overset{\text{(i)}}{=} \left(\frac{2}{3}\right)\left(\frac{2}{5}\right)\left(\frac{3}{7}\right)\\
&\overset{\text{(iv)}}{=} (-1)(-1)(-1)\left(\frac{7}{3}\right)\\
&\overset{\text{(i)}}{=} (-1)\left(\frac{1}{3}\right)\\
&= -1
\end{aligned}
$$

So 105 is a not a square mod 227.
This simple process requires $O(\log^3 p)$ reference bit operations but doesn't actually tell us the squareroot of a, if a is indeed a square mod p. If $\left(\frac{a}{p}\right) = 1$, the following method finds x such that $x^2 = a \bmod p$. We break the calcultions into two cases.

If $p \equiv 3 \bmod 4$, then $x = a^{(p+1)/4}$ satisfies $x^2 \equiv a \bmod p$. The second case where $p \equiv 1 \bmod 4$ is more involved.

1. Pick random r such that $\left(\frac{r^2-4a}{p}\right) = -1$ and write $d = r^2 - 4a$.
2. let $\alpha = \frac{r+\sqrt{d}}{2}$ and $\alpha^k = \frac{V_k+U_k\sqrt{d}}{2}$ where V_k, U_k are the coefficients of $1, \sqrt{d}$ in the k-th power of α.
3. $x = 2^{-1}V_{(p+1)/2}$ satisfies $x^2 \equiv a \bmod p$.

For this case, the proof that x does satisfy $x^2 \equiv a \bmod p$ is quite involved but can be found in GarrWALSH NOTES. Putting all this together we obtain the following algorthm which finds random points on an elliptic curve E.

3.4 Counting Points

When using an elliptic curve E for discrete logarithm based cryptosystems, it's of fundamental importance to know the number of $\mathbb{Z}_p$-rational points on E. This is because in 1978 Stephan Pohlig and Martin Hellman came up with an attack, described in [5], which uses the order of the group to solve discrete logs. Let G be a finite cyclic group with order N. We may factor $N = p_1^{e_1}p_2^{e_2}\cdots p_s^{e_s}$ where $p_1, p_2, ..., p_s$ are primes and $e_1, e_2, ..., e_s \in \mathbb{Z}^+$. All the subgroups $G_1, G_2, ..., G_s$ of G will have order $p_1^{e_1}, p_2^{e_2}, ..., p_s^{e_s}$ respectively. Given an element $h = g^a \in G$, the Pohlig-Hellman attack solves for a in the subgroups $G_1, G_2, ..., G_s$ and uses the chinese remainder theorem to piece these solutions back together to solve for a in the bigger group G. What they noticed is that using this method the

Algorithm 5 The Legendre symbol of an integer a modulo a prime p

```
function Legendre(a,p)
    s ← 1
    for q ∈ FACTORS(a) do
        s ← s∗LegendreForPrime(q,p)
    end for
    function LegendreForPrime(q,p)
        if q = 1 then
            return 1
        else if q ≡ 0 mod p then
            return 0
        else if q = −1 then
            if p ≡ 1 mod 4 then
                return 1
            else
                return −1
            end if
        else if q = 2 then
            if p ≡ ±1 mod 8 then
                return 1
            else
                return −1
            end if
        else if q > p then
            return LegendreForPrime(q mod p,p)
        else if q ≡ 1 mod 4 or p ≡ 1 mod 4 then
            return LegendreForPrime(p,q)
        else
            return −LegendreForPrime(p,q)
        end if
    end function
    return s
end function
```

Algorithm 6 Find random points on elliptic curve $E : y^2 - x^3 - ax - b$ modulo an odd prime p

```
function RandomPoints(E,p)
    pick random x ∈ {1, . . . , p − 1}
    while not Legendre(x^3 + ax + b,p) do
        pick another x ∈ {1, . . . , p − 1}
    end while
end function
```

complexity of solving for a in G (which can be done in $O(\sqrt{N})$ bit operations) gets reduced to $O(p_1^{e_1/2}) + O(p_2^{e_2/2}) + O(p_s^{e_s/2})$. This means a group's bits of security is only as high as the bits of security of its largest subgroup. Therefore we want the order of the groups we use to be prime, or at least a very small multiple of a prime.

With this is mind, we need an algorithm which calculates the number of points on an elliptic curve E and thus calcuales the order of E.

3.5 Generating Provably Random Elliptic Curves

4 Hyperelliptic Curves

Unlike elliptic curves, when the genus $\mathfrak{g}$ of a curve $\mathfrak{C}$ is greater than 1, the set of points on $\mathfrak{C}$ will not always form a group.

4.1 The Jacobian of a Hyperelliptic Curve

Luckily, there is another way to form an abelian group with hyperelliptic curves. Indeed, let $\mathfrak{D}$ be the set of all formal finite sums

$$\sum_i m_i P_i$$

where $m_i \in \mathbb{Z}$ and P_i are points on the curve $\mathfrak{C}$. We call elements of $\mathfrak{D}$ divisors of $\mathfrak{C}$. Given a rational function f in $\mathbb{Z}_p[\mathfrak{C}]$, we can define the corresponding divisor to f as

$$(f) = \sum_i m_i P_i$$

where P_i are the zeros and poles of f with multiplicities m_i.
Divisors of this form are called principal divisors and we let $\mathfrak{P}$ denote the subset of all of them in $\mathfrak{D}$. If we define the operation on $\mathfrak{D}$ by

$$\sum_i m_i P_i + \sum_i m'_i P_i = \sum_i (m_i + m')P_i$$

then $\mathfrak{D}$ becomes and abelian group. Unfortunetly, this group is far too large and unstructured for cryptographic purposes. So we consider the subgroup $\mathfrak{D}^0$ of all divisors of $\mathfrak{D}$ whose coefficients sum to 0. That is, divisors $\sum_i m_i P_i$ such that $\sum_i m_i = 0$.

This subgroup is still infinite, but that can be remedied by defining two divisors D_1, D_2 of $\mathfrak{D}^0$ to be equal if $D_1 - D_2$ is equal to the divisor of a rational function on $\mathfrak{C}$. That is, $D_1 - D_2 = (f)$ for $f \in \mathbb{Z}_p[\mathfrak{C}]$. This new quotient group, denoted

$$\mathfrak{J} = \mathfrak{D}^0/\mathfrak{P}$$

is called the jacobian of the curve $\mathfrak{C}$ and is a finite cyclic group. This will be the group used to build hyperelliptic cryptosystems.

4.2 Representation of Divisors

Athough the Jacobian $\mathfrak{J}$ of an hyperelliptic curve $\mathfrak{C}$ is a finite abelian group, elements of $\mathfrak{J}$ are very hard to represent.
To make the group operation in $\mathfrak{J}$ tractable, we ustilize the mumford representation of a divisor which is described as follows. Let D be a semi-reduced with points $P_i = (x_i, y_i)$. We associate to D polynomials $a, b \in \mathbb{Z}_p[x]$ such that

$$a(x) = \prod_{i}^{r}(x - x_i)$$

$$b(x_i) = y_i \; 1 \leq i \leq r$$

where $\deg b < \deg a$ and $(x - x_i)^{k_i} \mid b - y_i$, if k_i is the multiplicity of P_i. Denote this representation $D \stackrel{\text{def}}{=} \operatorname{div}(a, b)$.

4.3 The Group Operation

The group operation can be divided into two parts - *Composition* and *reduction* as described in [4].

Given two divisors represented as $D_1 = \operatorname{div}(a_1, b_1), D_2 = \operatorname{div}(a_2, b_2)$

1. compute $d_0 = \gcd(a_1, a_2)$ and find the unique $c_1, e_1 \in \mathbb{Z}_p[x]$ such that $d_0 = c_1 a_1 + e_1 a_2$
2. compute $d = \gcd(d_1, b_1 + b_2)$ and find the unique $c_2, e_2 \in \mathbb{Z}_p[x]$ such that $d = c_2 d_1 + e_2(b_1 + b_2)$
3. compute $a_3 = \frac{a_1, a_1}{d^2}$
4. compute $b_3 = \frac{c_2 c_1 a_1 + c_2 e_1 a_2 + e_2(b_1 b_2 + f)}{d} \bmod \frac{a_1, a_1}{d^2}$
5. compute $a_3' = \frac{f - b_3^2}{a_3}$ and $b_3' = -b_3 \bmod a_3'$
6. while $\deg(a_3') > g$, reassign $a_3 = a_3', b_3 = b_3'$ and repeat step 5
7. divide a_3' by its leading coefficient so that a_3' becomes monic
8. the output $div(a_3', b_3') = D_1 + D_2$

4.4 Finding Points

4.5 Counting Points

5 Implementation

```
/*
Here will be the python code.
use xelatex -shell-escape report.tex to compile
*/
```

6 References

1 Robin Hartshorne, *Algebraic Geometry, Graduate Texts in Mathematics, vol. 52*, Springer-Verlag, New York, 1977, ISBN 0-387-90244-9.

2 Victor Shoup, *Lower bounds for discrete logarithms and related problems*, Theory and Application of Cryptographic Techniques, 1997, pp. 256 - 266.

3 Whitfield Diffie and Martin E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **IT-22** (1976), no. 644-654.

4 Tanja Lange, *Formulae for Arithmetic on Genus 2 Hyperelliptic Curves*, ... not complete.

5 Stephan Pohlig and Martin E. Hellman, *An Improved Algorithm for Computing Logarithms over $GF(p)$ and its Cryptographic Significance*, IEEE Transactions on Information Theory, vol. m24, NO. 1, January 1978, pg 106.

6 An Commeine and Igor Semaev, *An Algorithm to Solve the Discrete Logarithm Problem with the Number Field Sieve*, Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, vol, 3958, pg 174-190, 2006.

7 Dan Coppersmith,

8 Doug Stinsons book