# Der Baby Step, Giant Step Algorithmus

Martin Albrecht (malb@informatik.uni-bremen.de)

7. Juni 2007

## 1 Motivation

Sei $P \in E(\mathbb{F}_q)$ Es geht darum die Ordnung von $P$ zu finden, d.h. die kleinste natürlichliche Zahl $n$, so dass $nP = \infty$. Sei $\#E(\mathbb{F}_q) = N$. Durch Lagrange's Theorem, wissen wir, dass $NP = \infty$. Ist $N$ unbekannt, wissen wir durch die Hasse Grenzen aber dennoch ein Interval in dem es liegt.

**Theorem 1.1** (Hasse Schranke)**.** *[Washington] Sei $E$ eine elliptische Kurve über einem endlichen Körper $\mathbb{F}_q$ Dann erfüllt die Ordnung von $E(\mathbb{F}_p)$ die folgende Ungleichung:*

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

Da die Ordnung des Punktes $N$ teilen muss, können wir einfach alle möglichen $N$ ausprobieren, sie faktorisieren und dann alle $N$-Teiler durchprobieren. Dafür müssen wir $4\sqrt{q}$ $N$s ausprobieren. Das ist schon mal schneller, als $O(q)$ Punkte auszuprobieren. Mit dem Baby Step, Giant Step Algorithmus kann $4\sqrt{q}$ auf $4q^{\frac{1}{4}}$ beschleunigt werden, indem wir mehr Speicher verwenden, d.h. Speicher gegen Zeit tauschen (Time-Memory-Tradeoff).

# 2 Der Algorithmus

**Algorithmus 1** (Baby Step Giant Step)**.**

```
def babystepgiantstep(P,q):
    # 1. Compute Q = (q+1)P
    Q = (q+1) * P

    # 2. Choos an integer m with m > q^{1/4}. Compute and store the
    # points jP for j = 0,1,2,...,m
    m = ZZ(floor(q**RR(0.25)) + 1)

    l = [j*P for j in range(m+1)]

    # 3. Compute the points Q + k(2mP) for k = -m, -(m+1), ..., m
    # until there is a match Q + k(2mP) = +- jP with a point or its
    # negative on the list
    for k in range(-m,m+1):
        W = Q + k*(2*m*P)
        if W in l:
            # 4a. Conclude that (q + 1 + 2mk - j)P = oo. Let M = q + 1 + 2mk - j
            M = q + 1 + 2*m*k - l.index(W)
            break
        elif -W in l:
            # 4b. Conclude that (q + 1 + 2mk + j)P = oo. Let M = q + 1 + 2mk + j
            M = q + 1 + 2*m*k + l.index(-W)
            break

    while True:
        N = M
        # 5. Factor M. let p1,...,pr be the distinct prime factors of M
        for p,n in factor(M):
            if (int(M/p) * P).is_zero():
                # 6. Compute (M/pi)P for i = 1...r. If (M/pi)P =
                # oo for some i, replace M with M/pi and go back
                # to step (5).
                M = M/p
                break
        if N == M:
            # If (M/pi)P != oo for all i then M
            # is the order of the point P.
            return M
```

# 3 Korrektheit

Die Korrektheit hängt an zwei Fragen:

## 3.1 Warum ist $\pm W \in l$?

Angenommen, wir finden $W$ oder $-W$ in $l$, dann ist klar, dass damit eine ganze Zahl $i$ gefunden wurde, so dass $iP = \infty$. Aber warum finden wir $W$ oder $-W$ in $l$?

**Lemma 3.1.** *[Washington] Sei $a$ eine ganze Zahl mit $|a| \leq 2m^2$. Es existieren ganze Zahlen $a_0$ und $a_1$, mit $-m < a_0 \leq m$ und $-m \leq a_1 \leq m$, so dass*

$$a = a_0 + 2ma_1$$

Ein Beispiel: Sei $m = 7$, sei $a = 22$. $|a| \leq 98$. Dann sind $(a_0, a_1)$ z.B. $(-6, 2)$, da gilt

$$22 = -6 + 2 \cdot 2 \cdot 7$$

sowie $-7 < -6 \leq 7$ und $-7 \leq 2 \leq 7$.

*Proof.* Setze $a_0 \equiv a \pmod{2m}$, mit $-m < a_0 \leq m$, und $a_1 = (a - a_0)/2m$. Dann gilt

$$|a_1| \leq (2m^2 + m)/2m < m + 1.$$

□

Sei nun $a = a_0 + 2ma_1$ wie im Lemma und setze $k = -a_1$. Dann

$$Q + k(2mP) = (q + 1 - 2ma_1)P = (q + 1 - a + a_0)P = NP + a_0P = a_0P = \pm jP,$$

wobei $j = |a_0|$. D.h. wir setzen $a$ so, dass $q + 1 + a = N$ und können sicher sein, in den erlaubten Grenzen ein passendendes $a_0$ zu finden. Also, finden wir $W$ oder $-W$ in $l$.

### 3.2 Warum erhalten wir die Ordnung von $P$?

**Lemma 3.2.** *[Washington] Sei $G$ eine additive Gruppe (mit 0) und $g \in G$. Sei $Mg = 0$ für eine natürliche Zahl $M$. Seien $p_1, \ldots, p_r$ paarweise verschiedene Primteiler von $M$. Wenn $(M/p_i)g \neq 0$ für alle $i$, dann ist $M$ die Ordnung von $g$.*

*Proof.* Sei $k$ die Ordnung von $g$. Dann $k \mid M$. Sei $k \neq M$ und $p_i$ eine Primzahl, welche $M/k$ teilt. Also gilt, dass $p_ik \mid M$ und damit $k \mid (M/p_i)$. Folglich gilt, dass $(M_i/pi)g = 0$, was im Widerspruch zur Annahme steht. Folglich ist $k = M$. □

## 4 Beispiel

Sei $E$ die elliptische Kurve $y^2 = x^3 - 10x + 21$ über $\mathbb{F}_{557}$. Sei $P = (2, 3)$

```
sage: k = GF(557)
sage: q = k.order()
sage: E = EllipticCurve(k,[-10,21]); E
Elliptic Curve defined by y^2  = x^3 + 547*x + 21 over Finite Field of size 557
sage: P = E(2,3); P
(2 : 3 : 1)
```

Wir folgen der Vorschrift:

1. $Q = 558P = (418, 33)$

   ```
   sage: Q = 558*P; Q
   (418 : 33 : 1)
   ```

2. Sei $m = 5$, was größer als $557^{1/4}$ ist. Die Liste $jP$ ist

   $$\infty, (2, 3), (58, 164), (44, 294), (56, 339), (132, 364).$$

   ```
   sage: m = 5;
   sage: m > 557^(0.25)
   True
   sage: l = [j*P for j in range(m+1)], l
   [(0 : 1 : 0), (2 : 3 : 1), (58 : 164 : 1),
   (44 : 294 : 1),(56 : 339 : 1), (132 : 364 : 1)]
   ```

3. Wenn $k = 1$, dann ergibt $Q + k(2mP) = (2, 3)$. Dieser Punkt ist auch in der Liste und das passende $j$ ist 1.

   ```
   sage: k = 1
   sage: W = Q + k*(2*m*P); W
   (2 : 3 : 1)
   sage: j = l.index(W); j
   1
   ```

4. Es gilt: $(q+1+2mk-j)P = 567P = \infty$. Strike!

```
sage: M = (q + 1 + 2*m*k -j); M
567
sage: M*P
(0 : 1 : 0)
```

5. Die Faktorisierung von $M = 567 = 3^4 \cdot 7$ Berechne $(567/3) * P = 189P = \infty$. Also ist 189 ein Kanditat für die Ordnung von $P$

```
sage: factor(M)
3^4 * 7
sage: 567/3
189
sage: int(567/3)*P
(0 : 1 : 0)
```

6. Faktorisiere $189 = 3^3 \cdot 7$. Es gilt $(189/3)P = (38, 535) \neq \infty$ und $(189/7)P = (136, 360) \neq \infty$. Also ist 189 die Ordnung von $P$.

```
sage: factor(189)
3^3 * 7
sage: int(189/3)*P
(38 : 535 : 1)
sage: int(189/7)*P
(136 : 360 : 1)
```

# 5 Wie schnell ist so etwas in aktuellen Computer Algebra Systemen?

## Modulo $p$

```
sage: p = next_prime(7^25); p
134106861966396490 0867
sage: k = GF(p)
sage: E = EllipticCurve(k,[2,4]); E
Elliptic Curve defined by y^2  = x^3 + 2*x + 4 over \
Finite Field of size 134106861966396490 0867
sage: P = E(475114122177702277610 , 792806591338212062383)
```

- SAGE 2.6.1[Sage]/Pari 2.3.1[Pari]

```
sage: time P.order() # PARI is called in the background
CPU times: user 0.01 s, sys: 0.01 s, total: 0.02 s
Wall time: 0.66
670534309818692356829
```

- MAGMA 2.13-5[Magma]

```
sage: PM = P._magma_()
sage: t = magma.cputime()
sage: PM.Order()
670534309818692356829
sage: print magma.cputime(t)
0.31
```

### Endliche Erweiterungskörper

```
sage: k.<a> = GF(7^10)
sage: E = EllipticCurve(k,[2,4]); E
Elliptic Curve defined by y^2 = x^3 + 2*x + 4 over Finite Field in a of size 7^10
sage: P = E(3*a^9+a^8+4*a^7+5*a^6+2*a^5+a^4+6*a^3+2*a+5 , \
            2*a^9+2*a^8+2*a^6+3*a^5+a^4+4*a^3+5*a+1)
sage: P
(3*a^9 + a^8 + 4*a^7 + 5*a^6 + 2*a^5 + a^4 + 6*a^3 + 2*a + 5 : \
 2*a^9 + 2*a^8 + 2*a^6 + 3*a^5 + a^4 + 4*a^3 + 5*a + 1 : \
 1)
```

- Die naive Implementation

```
sage: time babystepgiantstep(P)
4279550
CPU time: 5.58 s,  Wall time: 5.74 s
```

- Ein paar Optimierungen

```
sage: time P.order()
4279550
CPU time: 0.83 s,  Wall time: 0.86 s
```

- MAGMA

```
sage: PM = P._magma_()
sage: t = magma.cputime()
sage: print PM.Order()
sage: print magma.cputime(t)
4279550
0.01
```

# Literatur


[Magma] Bosma, Wieb and Cannon, John and Playoust, Catherine; *The Magma algebra system I: The user language*; Journal of Symbolic Computation, 24; 1997, 3-4, http://www.maths.usyd.edu.au:8000/u/magma/

[Pari] PARI/GP, version `2.3.1`, Bordeaux, 2007, `http://pari.math.u-bordeaux.fr/`.

[Sage] Stein, William and Joyner, David; *SAGE: System For Algebra And Geometry Experimentation*; 2005; available at `http://modular.ucsd.edu/sage`

[Washington] Washington, Lawcrence C.; *Elliptic Curves: Number Theory and Cryptography*; Chapman & Hall/CRC, Boca Raton [u.a.], 2003