Der Algorithmus von Montgomery

In der letzten Vorlesung haben wir explizite Formeln für die Addition und die Inversenbildung auf einer elliptischen Kurve in WEIERSTRASS-Normalform hergeleitet. Für die Anwendung elliptischer Kurven in der Kryptographie, zur Faktorisierung ganzer Zahlen oder für Primzahltests brauchen wir noch eine weitere Operation: Die Multiplikation mit einer ganzen Zahl. Diese ist auf einer elliptischen Kurve E genauso definiert wie in jeder abelschen Gruppe: Für ein $P \in E$ und ein $n \in \mathbb{N}$ ist

$$nP = \underbrace{P + \cdots + P}_{n \text{ Summanden}};$$

für $n = 0$ ist $nP = O$, und für $n < 0$ ist $nP = (-n)(-P)$.

Grundsätzlich läßt sich diese Definition auch zur Berechnung von nP verwenden; im letzten Beispiel in der vorigen Vorlesung ging es um so eine Rechnung. Bei den Anwendungen, die wir im Rest des Semesters betrachten wollen, wird aber n sehr groß sein, beispielsweise in der Größenordnung von 2^{256}, und so viele Additionen können wir heute nicht einmal mit den besten Supercomputern ausführen, selbst wenn wir mehrere Jahre Rechenzeit zulassen. Kryptologen gehen davon aus, daß ein Kryptosystems nach heutigen Standards *praktisch* sicher ist, wenn ein Gegner mindestens etwa 2^{128} Rechenoperationen durchführen muß, um es zu knacken; das Bundesamt für Sicherheit in der Informationstechnik redet sogar schon ab 2^{120} Rechenoperationen von praktischer Sicherheit.

Die Grundidee zur effizienten Berechnung von nP auch für große n ist einfach: Wenn wir etwa den Punkt $32P$ berechnen wollen, können wir natürlich 31 Additionen ausführen; schneller geht es aber, wenn wir ihn mit fünf Verdoppelungen als $2 \cdot (2 \cdot (2 \cdot (2 \cdot 2P)))$ berechnen.

Für einen beliebigen Multiplikator n bietet sich folgendes Verfahren an: Man schreibt $n = \sum_{i=0}^{k} a_i 2^i$ als Binärzahl, d.h. $a_i \in \{0, 1\}$, und berechnet durch sukzessive Verdoppelung die Punkte $2^i P$ für $i \leq k$. nP ist die Summe derjenigen $2^i P$, für die $a_i = 1$ ist. (Zur konkreten Implementierung geht das natürlich auch noch etwas effizienter; man

muß nicht alle Punkte $2^i P$ abspeichern, sondern kann sie verarbeiten, sobald sie anfallen.

Der Algorithmus von MONTGOMERY baut darauf auf, macht die Berechnung aber noch etwas effizienter, indem er in den einzelnen Rechenschritten nur die x-Koordinaten der Punkte betrachtet. Erst zum Schluß wird dann die y-Koordinate des Ergebnisses bestimmt.

Für $n = \sum_{i=1}^{k} a_i 2^i$ und $\ell \leq k+1$ bezeichne n_ℓ die Binärzahl, die aus den ℓ vorderen Binärziffern von n zusammengesetzt ist, d.h.

$$n_\ell = \sum_{i=k+1-\ell}^{k} a_i 2^{i-1-(k-\ell)} .$$

Im ℓ-ten Schritt werden die beiden Punkte $U_\ell = n_\ell P$ und $V_\ell = (n_\ell + 1)P$ berechnet.

Setzen wir formal $n_0 = 0$, $U_0 = O$ und $V_0 = P$, so lassen sich die n_ℓ rekursiv berechnen nach der Vorschrift $n_\ell = 2n_{\ell-1} + a_{k+1-\ell}$, und $n_{k+1} = n$.

Falls die neu dazugekommene Ziffer $a_{k+1-\ell}$ eine Null war, ist n_ℓ gleich $2n_{\ell-1}$, d.h. $U_\ell = 2U_{\ell-1}$ und $V_\ell = U_{\ell-1} + V_{\ell-1}$; andernfalls ist n_ℓ gleich $2n_{\ell-1} + 1$ und damit $U_\ell = U_{\ell-1} + V_{\ell-1}$ und $V_\ell = 2V_{\ell-1}$. Nach dem $(k+1)$-ten Schritt ist $U_{k+1} = nP$ und $V_{k+1} = (n+1)P$.

Da MONTGOMERY in jedem Schritt sowohl eine Verdopplung als auch eine Addition zweier verschiedener Punkte durchführt, wird jede dieser Operationen für jede Ziffer genau einmal angewandt. Wenn wir stattdessen den üblichen binären Algorithmus angewandt hätten, wäre zwar auch für jede Ziffer eine Verdopplung notwendig gewesen, die sonstige Addition aber nur für jede Eins; die Gesamtanzahl der Rechenoperationen wäre also im Mittel und 25% kleiner. (Mit zusätzlichen Tricks läßt sich sogar erreichen, daß die Anzahl der sonstigen Additionen im Mittel nur ein Drittel der Ziffernzahl ist.)

Nur am Rande sei vermerkt, daß es für Anwendungen in der Kryptologie durchaus ein Sicherheitsgewinn sein kann, wenn in *jedem* Schritt sowohl eine Verdoppelung als auch eine sonstige Addition durchzuführen: Hier ist die Zahl n typischerweise etwas geheim zu haltendes, während

die Punkte P und nP einem Gegner bekannt sein können. Falls der Punkt nP von einer Chipkarte berechnet wird, erhält diese ihren Strom durch induktive Ankoppelung vom Lesegerät. Beim Bezahlen in einem unbekannten Geschäft oder Restaurant kann man nie ausschließen, daß dieses Lesegerät manipuliert ist, beispielsweise könnte es die Kurve des Stromverbrauchs der Chipkarte aufzeichnen und auswerten. In der Kryptologie spricht man hier von einer *side channel attack,* einem Angriff über einen versteckten Kanal. Da Verdopplungen einfacher zu berechnen sind als sonstige Addition, verbrauchen sie weniger Zeit und weniger Strom. Der Gegner kann somit die Abfolge von Verdoppelungen und sonstigen Additionen erkennen und damit beim „klassischen" Algorithmus die Binärziffern von N identifizieren. Beim Algorithmus von MONTGOMERY führen beide möglichen Ziffern zu sowohl einer Verdoppelung als auch einer sonstigen Addition, und man kann beides auch immer in der gleichen Reihenfolge durchführen. Somit erkennt der Gegner aus seinen Messungen lediglich die Bitlänge von n, und die ist meist ohnehin eine allgemein bekannte globale Konstante des Kryptosystems.

Der eigentliche Vorteil des Algorithmus von MONTGOMERY liegt aber darin, daß wir in den Zwischenschritten die Punkte U_ℓ und V_ℓ nicht vollständig berechnen müssen, sondern uns mit ihren x-Koordinaten begnügen können:

Die Formeln aus der letzten Vorlesung hängen natürlich von beiden Koordinaten ab, und es ist meist auch sinnvoll, sie so anzuwenden, da man ohnehin beide Koordinaten kennt. Nun ist aber y^2 über die WEIERSTRASS-Gleichung $y^2 = x^3 + ax + b$ durch x bestimmt; wenn y in einer Formel nur quadratisch vorkommt, können wir es also durch $x^3 + ax + b$ ersetzen und haben dann einen Ausdruck, der nur von x abhängt.

Bei der Formel zur Berechnung von $2P$ ist das der Fall: Hat P die Koordinaten (x_1, y_1), so hat $2P$ die x-Koordinate $m^2 - 2x_1$ mit

$$m = \frac{3x_1^2 + a}{2y_1} \qquad \text{und} \qquad m^2 = \frac{(3x_1^2 + a)^2}{4y_1^2} = \frac{(3x_1^2 + a)^2}{4(x_1^3 + ax_1 + b)} \,;$$

damit ist die x-Koordinate von $2P$ gleich

$$\frac{(3x_1^2+a)^2}{4(x_1^3+ax_1+b)} - 2x_1 = \frac{9x_1^4+6ax_1^2+a^2-8x_1^4-8ax_1^2-8bx_1}{4(x_1^3+ax_1+b)}$$
$$= \frac{(x_1^2-a)^2-8bx_1}{4(x_1^3+ax_1+b)} .$$

Man beachte, daß hier der Nenner genau dann verschwindet, wenn die rechte Seite der WEIERSTRASS-Gleichung im Punkt P verschwindet, wenn wir also einen Punkt der Form $(x_1, 0)$ haben. In diesem Fall ist $2P = O$, was wir rechnerisch daran erkennen, daß wir zur Berechnung von $2P$ durch Null dividieren müßten.

Bei der Addition zweier Punkte $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ mit $x_1 \neq x_2$ ist die Situation schwieriger: $P_1 + P_2$ hat die x-Koordinate

$$x_+ = m^2 - x_1 - x_2 \qquad \text{mit} \qquad m = \frac{y_2-y_1}{x_2-x_1} .$$

Im Zähler von m^2 kommt außer y_1^2 und y_2^2, die wir beide über die WEIERSTRASS-Gleichung durch x_1 und x_2 ausdrücken können, noch ein Term $2y_1y_2$ vor, für den das nicht möglich ist.

Ähnlich ist es auch bei der Subtraktion, die uns zwar eigentlich nicht interessiert, die hier aber den benötigten Ausweg liefert: $P_1 - P_2$ ist die Summe von P_1 und $-P_2 = (x_2, -y_2)$; die x-Koordinate der Differenz ist somit

$$x_- = n^2 - x_1 - x_2 \qquad \text{mit} \qquad n = \frac{-y_2-y_1}{x_2-x_1} = -\frac{y_2+y_1}{x_2-x_1} .$$

Das Produkt der beiden Koordinaten ist

$$x_+x_- = (m^2-x_1-x_2)(n^2-x_1-x_2)$$
$$= (mn)^2 - (x_1+x_2)(m^2+n^2) + (x_1+x_2)^2 .$$

In den beiden Termen

$$mn = -\frac{y_2^2-y_1^2}{(x_2-x_1)^2}$$

und

$$m^2+n^2 = \frac{(y_2-y_1)^2+(y_1+y_2)^2}{(x_2-x_1)^2} = 2\cdot\frac{y_1^2+y_2^2}{(x_2-x_1)^2}$$

kommen y_1 und y_2 nur noch quadratisch vor, können also durch die zugehörigen x-Werte ausgedrückt werden:

$$mn = -\frac{(x_1^3 + ax_1 + b) - (x_2^3 + ax_2 + b)}{(x_2 - x_1)^2} = -\frac{(x_1^3 - x_2^3) + a(x_1 - x_2)}{(x_2 - x_1)^2}$$
$$= -\frac{x_1^2 + x_1x_2 + x_2^2 + a}{x_2 - x_1}$$

und

$$m^2 + n^2 = 2 \cdot \frac{x_1^3 + x_2^3 + a(x_1 + x_2) + 2b}{(x_2 - x_1)^2} .$$

Somit ist x_+x_- gleich

$$\frac{(x_1^2 + x_1x_2 + x_2^2 + a)^2}{(x_2 - x_1)^2} - (x_1 + x_2)\frac{x_1^3 + x_2^3 + a(x_1 + x_2) + 2b}{(x_2 - x_1)^2}$$
$$+\frac{(x_1 + x_2)^2(x_2 - x_1)^2}{(x_2 - x_1)^2}$$
$$= \frac{x_1^4 + x_1^2x_2^2 + x_2^4 + a^2 + 2x_1^3x_2 + 2x_1x_2^3 + 2x_1^2x_2^2 + 2ax_1^2 + 2ax_1x_2 + 2ax_2^2}{(x_2 - x_1)^2}$$
$$-2 \cdot \frac{x_1^4 + x_2^4 + x_1^3x_2 + x_1x_2^3 + ax_1^2 + 2ax_1x_2 + ax_2^2 + 2bx_1 + 2bx_2}{(x_2 - x_1)^2}$$
$$+\frac{x_1^4 + 2x_1^2 + x_2^4}{(x_2 - x_1)^2}$$
$$= \frac{x_1^2x_2^2 + a^2 + 2 - ax_1x_2 - 4bx_1 - 4bx_2}{(x_2 - x_1)^2} = \frac{(x_1x_2 - a)^2 - 4b(x_1 + x_2)}{(x_2 - x_1)^2} .$$

Der Nenner verschwindet wieder genau dann, wenn $x_1 = x_2$ ist, wenn also (da wir $P_1 \neq P_2$ vorausgesetzt haben) $P_1 = -P_2$ ist und damit $P_1 + P_2 = O$. Auch hier also erkennen wir an einer erforderlich werdenden Division durch Null, daß das Ergebnis gleich O ist.

Damit haben wir das Produkt x_+x_- als Funktion von x_1 und x_2 ausgedrückt; wenn wir in einer speziellen Situation auch x_- so ausdrücken können, läßt sich der gesuchte Wert x_+ aus x_1 und x_2 berechnen.

Bei MONTGOMERYs Algorithmus sind wir in so einer speziellen Situation: Wir führen in jedem Schritt eine Verdoppelung aus, von der wir

bereits wissen, daß die x-Koordinate des Ergebnisses nur von der x-Koordinate des zu verdoppelnden Punkts abhängt, und wir berechnen die Summe $U_{\ell-1} + V_{\ell-1}$. Wegen $U_{\ell-1} = (\ell-1)P$ und $V_{\ell-1} = \ell P$ ist $V_{\ell-1} - U_{\ell-1} = P$, und diesen Punkt kennen wir natürlich.

Ist also $P = (x_1, y_1)$, und bezeichnen u_ℓ, v_ℓ die x-Koordinaten von U_ℓ und V_ℓ, so ist die x-Koordinate von $U_\ell + V_\ell$ nach obiger Formel gleich

$$\frac{(u_\ell v_\ell - a)^2 - 4b(v_\ell - u_\ell)}{(v_\ell - u_\ell)^2 \cdot x_1} \,.$$

Als Endergebnis wollen wir freilich nicht nur die x-Koordinate von nP, sondern auch die y-Koordinate. Zum Glück liefert uns MONTGOMERY nicht nur die x-Koordinate u_{k+1} von nP, sondern auch die x-Koordinate v_{k+1} von $(n+1)P$, und damit können wir die gesuchte y-Koordinate y_* von nP bestimmen: Wenn wir $(n+1)P$ berechnen als $nP+P$, so erhalten wir für die x-Koordinate nach der Additionsformel

$$v_{k+1} = \left(\frac{y_* - y_1}{u_{k+1} - x_1}\right)^2 - u_{k+1} - x_1 \,.$$

Auflösen nach $(y_+ - y_1)^2$ führt auf die Gleichung

$$(y_* - y_1)^2 = (v_{k+1} + u_{k+1} + x_1)(u_{k+1} - x_1)^2 \,.$$

Da $nP = (u_{k+1}, y_*)$ auf der Kurve liegt, ist

$$(y_* - y_1)^2 = y_*^2 - 2y_1y_* + y_1^2 = u_{k+1}^3 + au_{k+1} + b - 2y_1y_* + y_1^2 \,,$$

und damit ist

$$y_* = \frac{u_{k+1}^3 + au_{k+1} + b + y_1^2 - (v_{k+1} + u_{k+1} + x_1)(u_{k+1} - x_1)^2}{2y_1} \,.$$

Wenn wir den Algorithmus von MONTGOMERY wirklich nur mit x-Werten durchführen wollen, haben wir Probleme mit der Initialisierung, da ja $U_0 = O$ der unendlichferne Punkt ist. Deshalb beginnen wir besser erst mit U_1 und V_1. Wir können natürlich annehmen, daß in der Binärdarstellung $n = \sum_{i=0}^{k} a_i 2^i$ die erste Ziffer $a_k = 1$ ist. Somit sind wir im ersten Schritt immer im zweiten Fall, d.h. $U_1 = U_0 + V_0 = P$

und $V_1 = 2V_0 = 2P$. Mit $P = (x_1, y_1)$ ist die x-Koordinate u_1 von U_1 somit x_1; die von V_1 ist

$$v_1 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 = \frac{(x_1^2 - a)^2 - 8bx_1}{4(x^3 + ax_1 + b)} .$$

Praktisch funktioniert der Algorithmus von MONTGOMERY damit folgendermaßen:

Gegeben sind ein Punkt $P = (x_1, y_1)$ auf der elliptischen Kurve mit Gleichung $y^2 = x^3 + ax + b$ sowie eine natürliche Zahl $n = \sum_{i=0}^{k} a_i 2^i$ mit $a_i \in \{0, 1\}$ und $a_k = 1$; gesucht ist der Punkt nP.

Schritt 1: Initialisiere $u_1 = x_1$, v_1 wie oben und $n_1 = 1$.

Schritt ℓ, $\ell = 2, \ldots, k + 1$: Setze $n_\ell = 2n_{\ell-1} + a_{k+1-\ell}$. Falls $a_{k+1-\ell}$ verschwindet, setze

$$u_\ell = \frac{(u_{\ell-1}^2 - a)^2 + 8bu_{\ell-1}}{4(u_{\ell-1}^3 + au_{\ell-1} + b)}$$

und

$$v_\ell = \frac{(u_{\ell-1}v_{\ell-1} - a)^2 - 4b(v_{\ell-1} + u_{\ell-1})}{(v_{\ell-1} - u_{\ell-1})^2 x_1} ,$$

andernfalls, für $a_{k+1-\ell} = 1$ also, setze

$$v_\ell = \frac{(v_{\ell-1}^2 - a)^2 + 8bv_{\ell-1}}{4(v_{\ell-1}^3 + av_{\ell-1} + b)}$$

und

$$u_\ell = \frac{(u_{\ell-1}v_{\ell-1} - a)^2 - 4b(v_{\ell-1} + u_{\ell-1})}{(v_{\ell-1} - u_{\ell-1})^2 x_1} .$$

Schritt $k + 2$: Setze

$$y_* = \frac{u_{k+1}^3 + au_{k+1} + b - y_1^2 - (v_{k+1} + u_{k+1} + x_1)(u_{k+1} - x_1)^2}{2y_1} .$$

Der Punkt nP hat die Koordinaten (u_{k+1}, y_*).

Dieser Algorithmus ist nicht nur interessant zur praktischen Berechnung höherer Vielfacher eines Punktes, sondern liefert uns wegen seiner

speziellen Struktur auch ein theoretisches Ergebnis: Angenommen, wir starten mit einem unbestimmten Punkt $P = (x_1, y_1)$, betrachten x_1 und y_1 also als Variablen. Wenn wir für v_2 die zweite Form der Initialisierung wählen, folgt induktiv, daß alle u_ℓ, v_ℓ rationale Funktionen von x_1 sind; insbesondere läßt sich die x-Koordinate von nP darstellen als eine rationale Funktion $r_n(x_1)$ in x_1. Die y-Koordinate berechnen wir als Quotient aus einer rationalen Funktion in x_1 dividiert durch $2y_1$. Da

$$\frac{1}{y_1} = \frac{y_1}{y_1^2} = \frac{y_1}{x_1^3 + ax_1 + b}$$

ist, können wir y_* auch ausdrücken als Produkt aus y_1 und einer rationalen Funktion. Damit haben wir gezeigt:

Satz: Zu jedem $n \in \mathbb{N}$ gibt es rationale Funktionen

$$r_N(x_1), s_N(x_1) \in k(x_1)$$

derart, daß das n-fache des Punktes $P = (x_1, y_1)$ die Koordinaten $\big(r_n(x_1),\ ys_n(x_1)\big)$ hat.

■