10. Woche: Elliptische Kurven
Skalarmultiplikation und Anwendungen

Skalarmultiplikation

Eine wichtige Fundamentaloperation in vielen kryptographischen Protokollen ist die Skalarmultiplikation, d.h.

Skalarmultiplikation

Gegeben eine Gruppe $G = (G, +, 0)$, ein Element $P \in G$, und eine natürliche Zahl n, die Skalarmultiplikation $n \cdot P$ ist das n-te Vielfache von P, d.h.

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

Auf elliptischen Kurven können wir solche Operation problemlos durchführen, denn wir kennen alle nötigen Details des Gruppengesetzes.

Die naïve Berechnung von $n \cdot P$ benötigt $n - 1$ Gruppenoperationen, d.h. die Komplexität ist $\Theta(n)$.

Das ist aber zu viel. Wir wollen nun diese Operation optimieren.

Ein spezieller Fall

Sei $n = 2^\ell$. Wir wollen nun $n \cdot P$ berechnen.

Wir können einfach P ℓ-mal verdopplen:

$$P\,,\ \ 2\cdot P\,,\ \ 4\cdot P\,,\ \ \ldots\,,\ \ 2^i\cdot P\,,\ \ 2^{i+1}\cdot P\,,\ \ \ldots\,,\ \ 2^{\ell-1}\cdot P\,,\ \ 2^\ell\cdot P\ .$$

Das ist ℓ Gruppenoperationen, d.h. $\log_2 n$ Schritte.
In Komplexitätstheoretischen Notation: $\Theta(\log_2 n)$.

Problem: das geht nur für Zweierpotenzen.

Verallgemeinerung

Beobachtung: alle natürliche Zahlen lassen sich als Summen von Zweierpotenzen schreiben!
Sei

$$n = \sum_{i \in \Im} 2^i \quad \text{mit} \quad \ell := \max\{i \in \Im\} \ .$$

Dann lässt $n \cdot P$ sich wie folgt berechnen:

1. Berechne
   $P, \ 2 \cdot P, \ 4 \cdot P, \ \ldots, \ 2^i \cdot P, \ 2^{i+1} \cdot P, \ \ldots, \ 2^{\ell-1} \cdot P, \ 2^\ell \cdot P \ .$
2. Für $i \in \Im$ addiere die entsprechenden $2^i \cdot P$ zusammen.

Die Korrektheit der Methode folgt aus

$$n \cdot P = \left(\sum_{i \in \Im} 2^i \right) \cdot P = \sum_{i \in \Im} (2^i \cdot P) \ .$$

Komplexität

Zeit:

1. $\ell = \lfloor \log_2 n \rfloor$ Verdopplungen für Schritt 1.
2. $\#\Im - 1$ Additionen für Schritt 2 (um k Elementen zusammen zu addieren brauche ich nur $k - 1$ Additionen)
3. $\#\Im$ ist das Hamming-Gewicht der Binärdarstellung von n, wobei wir wissen, die höchstwertige Stelle gleich 1 ist. Also $1 \leqslant \#\Im \leqslant \ell + 1$ und da alle Stellen bis auf die höchstwertige beide Werte 0 und 1 mit gleicher Wahrscheinlichkeit annehmen, der erwartete Wert is $1 + \ell/2$

Also: Komplexität $\Theta(\log_2 n)$ und erwartet $\frac{3}{2} \log_2 n$.

Raum: Speicher für $2^i \cdot P$ mit $0 \leqslant i \leqslant \ell$ und für einen „Akkumulator" für die Summe. $\ell + 2$ Registern (gross wie die Gruppenelementen).

Die Zeit-Komplexität lässt sich nur ein wenig verbessern, die Raum-Komplexität aber noch drastisch.

Skalarmultiplikation mit einer Leiter

Idee: Akkumulator X am Anfang auf 0 setzen, $2^i \cdot P$ für $i = 1, 2, \ldots, \ell$ zu berechnen und sofort zu X addieren, falls $i \in \Im$ ist.

Skalarmultiplikation mit einer Leiter

EINGABE: $n = \sum_{i \in \Im} 2^i$ in binärer Darstellung, $\ell := \max\{i \in \Im\}$, Gruppe G und $P \in G$

1. $X \leftarrow 0$, $Q \leftarrow P$
2. For $i = 0, 1, \ldots, \ell$ do
   1. if $i \in \Im$ then $X \leftarrow X + Q$
   2. if $i \neq \ell$ then $Q \leftarrow 2 \cdot Q$

AUSGABE: $X = \sum_{i \in \Im} 2^i \cdot P = n \cdot P$

Zeitaufwand: genau wie auf vorigen Folie.

Speicherplatzbedarf: P, Q und X, also nur drei Register (konstant!). Man beachte, die Registern sind wenigstens $\lceil \log_2 |G| \rceil$ Bits gross.

Andere Schreibweise

Sei die *Ziffernentwicklung* $n = \sum_{i=0}^{\ell} d_i 2^i$ von n gegeben, wobei $d_i \in \{0, 1\}$.

Skalarmultiplikation mit einer Leiter (2. Version)

EINGABE: $n = \sum_{i=0}^{\ell} d_i 2^i$ in binärer Darstellung, Gruppe G und $P \in G$

1. $X \leftarrow 0$, $Q \leftarrow P$
2. For $i = 0, 1, \ldots, \ell$ do
   1. if $d_i = 1$ then $X \leftarrow X + Q$
   2. if $i \neq \ell$ then $Q \leftarrow 2 \cdot Q$

AUSGABE: $X = \sum_{i=0}^{\ell} d_i 2^i \cdot P = n \cdot P$

Nächst: neuer Algorithmus.

Verdopple-und-Addiere I

Sei $n = \sum_{i=0}^{\ell} d_i 2^i$ wobei $d_i \in \{0, 1\}$.

Verdopple-und-Addiere

EINGABE: $n = \sum_{i=0}^{\ell} d_i 2^i$ in binärer Darstellung, $\ell := \max\{i \in \Im\}$, Gruppe G und $P \in G$

1. $X \leftarrow 0$
2. For $i = \ell, \ell - 1, \ldots, 0$ do
   1. (if $i \neq \ell$ then $X \leftarrow 2 \cdot X$)
   2. if $d_i = 1$ then $X \leftarrow X + P$

AUSGABE: $X = \sum_{i=0}^{\ell} d_i 2^i \cdot P = n \cdot P$

Vorteile

- ein Register weniger
- öffnet die Möglichkeit, grössere Ziffernmengen zu verwenden.

Woher kommt dieser Algorithmus?

Verdopple-und-Addiere II

Falls

$$n = \sum_{i=0}^{\ell} d_i 2^i \quad \text{dann} \quad n \cdot P = d_i 2^i \cdot P$$

und wir können $n \cdot P$ so berechnen:

$$2\big(2\big(\cdots 2\big(2\big(2(d_\ell \cdot P) + d_{\ell-1} \cdot P\big) + d_{\ell-2} \cdot P\big) + \cdots + d_2 \cdot P\big) + d_1 \cdot P\big) + d_0 \cdot P$$

Verdopple-und-Addiere!
Das ist korrekt auch wenn die d_i andere Werte also $0, 1, -1$ nehmen dürfen (um das zu nutzen, muss man die möglichen Punkte $d_i \cdot P$ im Voraus kennen).

Beweis: nutze die *Linearität* von der Skalarmultiplikation, also folgendes Distributivgesetz: $a \cdot P + b \cdot P = (a + b) \cdot P$.

Man sieht per Induktion (über ℓ), dass obigen Ausdruck $= n \cdot P$ ist.

NAF: Nicht-Angrenzende Form

NAF = non-adjacent form (engl.) = nicht-angrenzende Form (de.)
Ausgangspunkt: auf EK können wir addieren und *subtrahieren*: Falls men eine Entwicklung

$$n = \sum_{i=0}^{\ell} d_i 2^i \quad \text{mit} \quad d_i \in \{0, 1, -1\}$$

hat, dann gibt es folgenden Algorithmus

Verdopple-und-Addiere-oder-Subtrahiere

EINGABE: $n = \sum_{i=0}^{\ell} d_i 2^i$ mit $d_i \in \{0, 1, -1\}$, Gruppe G und $P \in G$

1. $X \leftarrow 0$
2. For $i = \ell, \ell - 1, \ldots, 0$ do
   1. (if $i \neq \ell$ then $X \leftarrow 2 \cdot X$)
   2. if $d_i = 1$ then $X \leftarrow X + P$
   3. if $d_i = -1$ then $X \leftarrow X - P$

AUSGABE: $X = \sum_{i=0}^{\ell} d_i 2^i \cdot P = n \cdot P$

Frage: doch besser? Wann? Warum?

Berechnung der NAF

Beo: eine ungerade Zahl ist kongruent zu 1 oder -1 modulo 4.

Berechnung der NAF

Eingabe: $n \in \mathbb{Z}$,

1. $z \leftarrow n$
2. $\ell \leftarrow 0$
3. while $z \neq 0$ do
   1. if z even then
      1. $d_\ell \leftarrow 0$
   2. else
      1. Sei $d_\ell \in D$ mit $d_\ell \equiv z \ mod(4)$
   3. $z = \frac{z-d_\ell}{2}$
   4. if $z \neq 0$ then $\ell = \ell + 1$
4. Ausgabe: $(\sum_{i=0}^{\ell} d_i 2^i)$ mit $d_i \in \{0, 1, -1\}$ und $d_i = 0 \vee d_{i+1} = 0$

Korrektheit (falls Terminiert)

Es ist $n = (d_\ell\, d_{\ell-1} \ldots d_1\, d_0)_2 = (\underbrace{d_\ell \ldots d_1}_{\frac{n-d_0}{2}}\ \underbrace{d_0}_{=0})_2$

Also: wenn man weiss, wie man mindestwertige Ziffer von einer Eingabe bestimmt, kann man *rekursiv* die ganze Entwicklung bestimmen.

- Falls n gerade, dann reduzieren wir modulo 2 die Gleichheit $\sum_{i=0}^{\ell} d_i 2^i = n$: es ist $d_0 \equiv 0 \bmod 2$, also $d_0 = 0$.
- Falls n ungerade, dann $d_0 \equiv 1 \equiv -1 \bmod 2$: ist d_0 nicht eindeutig?
- Reduziert man modulo 4, so erhält man $n \equiv d_0 + 2\,d_1 \equiv$ **entweder** 1 **oder** $-1 \bmod 4$ ist.
- Falls $n \equiv 1 \bmod 4$, dann setzen wir $d_0 = 1$.
  Also ist $n - 1 \equiv 0 \bmod d$, und $(n-1)/2 \equiv 0 \bmod 2$.
  Nun ist d_1 die mindestwertige Ziffer von $(n-1)/2$ und es muss $d_1 = 0$.
- Analog, falls $n \equiv -1 \bmod 4$, dann $d_0 = -1$ und $d_1 = 0$.

Ein rekursiver Algorithmus „klebt" einfach d_0 (bestimmt wie oben) und die Entwicklung von $\frac{n-d_0}{2}$ zusammen; ferner, da $d_0 \neq 0 \Rightarrow d_1 = 0$, ist auch $d_i \neq 0 \Rightarrow d_{i+1} = 0$, also ist seine Ausgabe Korrekt.
Der Algorithmus auf vorigen Folie ist eine iterative Version davon. □

Terminierung

OBdA $n > 0$. Für $n = 1$ Algorithmus terminiert sofort korrekt.

Falls $n > 1$, dann

$$\frac{n - d_0}{2} < n \quad \Leftrightarrow \quad -\frac{d_0}{2} < \frac{n}{2} \quad \Leftrightarrow \quad -d_0 < n$$

und das ist immer erfüllt.

Also wird n durch strikt kleinere Zahlen ersetzt: ergibt Folge absteigender Zahlen.

Diese Folge absteigender Zahlen muss halten.

Analyse I

Wie groß ist der Vorteil durch die NAF?.
Länge: Die Zahl z hat genau dann $\ell + 1$ Stellen, wenn $d_\ell = 1$ und $\forall t > l: \ d_t = 0$ gilt. Die Entwicklung hat *Länge* $\boldsymbol{\ell + 1}$ hat.

Das (Hamming)Gewicht von $n = \sum_{i=0}^{\ell} d_i 2^i$ ist $\boldsymbol{w(n) = \#\{d_i | d_i \neq 0\}}$

Die „Dichte“ einer Entwicklung ist gegeben als: $\boldsymbol{d(n) = \frac{w(n)}{\ell + 1}}$

Satz: Dichte der Binärdarstellung und der NAF

Der Erwartungswert der Dichte der binären Darstellung einer zufällig gewählten natürlichen Zahl n ist $\mathbb{E}(d(bin(n))) = \frac{1}{2}$.
Der Erwartungswert der Dichte der NAF einer zufällig gewählten natürlichen Zahl n ist $\mathbb{E}(d(NAF(n))) = \frac{1}{3}$.

Also: Komplexität reduziert von $\frac{3}{2}\ell$ (erwartet) Gruppenoperationen auf $\frac{4}{3}\ell$ (erwartet). 11.1% weniger Gruppenoperationen.
Wenn nur Additionen gezählt: von $\frac{1}{2}\ell$ auf $\frac{1}{3}\ell$, also 33.3% weniger.

Analyse II

Binärdarstellung: Jede Ziffer der Binärenentwicklung einer zufällig gewählten ganzen Zahl ist mit gleicher Wahrscheinlichkeit 0 oder 1.

NAF: Mit Wahrscheinlichkeit $1/2$ ist eine Zahl gerade also ist ihre niederwertigste Ziffer $d_0 = 0$, und $\frac{n}{2}$ verhält sich wieder wie eine zufällig gewählte ganze Zahl (mutige Annahme).

Ist n hingegen ungerade, so ist die niederwertigste Ziffer $d_0 \in \{1, \bar{1}\}$ und $d_1 = 0$ und $\frac{n-(d_0+2d_1)}{4}$ verhält sich wie eine zufällig gewählte ganze Zahl.

Modellieren wir obigen Prozess:
Man konstruiere eine zufällige NAF, in dem man aus einer Urne mit zwei Kärtchen, die mit 0 und $0\star$ beschriftet sind, t-Mal ein Kärtchen mit zurücklegen zieht. $\star$ steht für „ungleich Null“.

Bei $\frac{t}{2}$ (erwartet) Ziehungen wird 0 gezogen, bei den restlichen $\frac{t}{2}$ (erwartet) $0\star$.

Somit ergibt sich eine erwartete Länge von $1 \cdot \frac{t}{2} + 2 \cdot \frac{t}{2} = \frac{3t}{2}$ und ein erwartetes Gewicht von $\frac{t}{2}$.

Daher ist die erwartete Dichte der NAF $\mathbb{E}(d(NAF(n))) = \frac{t/2}{3t/2} = \frac{1}{3}$. □

Effizientere Rekodierung

Sei $w \in \mathbb{N}$, $w \geqslant 1$, ein Parameter.
Die w-NAF ist definiert als eine Ziffernentwicklung (zur Basis 2) $n = \sum_{j=0}^{\ell} d_j 2^j$ mit folgenden Eigenschaften:

(w-NAF-1) Entweder $d_j = 0$ oder d_j ist ungerade.

(w-NAF-2) Falls $d_j \neq 0$, dann $d_{j+1} = \cdots = d_{j+w-1} = 0$.

Um eine solche Darstellung zu erhalten wird normalerweise eine der folgenden Ziffernmengen verwendet:

1. $D \subseteq \mathbb{N}_0$ also: $D = \{0\} \cup \{1, 3, 5, \ldots, 2^w - 1\}$
2. $D = -D$ also: $D = \{0\} \cup \pm\{1, 3, 5, \ldots, 2^{w-1} - 1\}$

Die erste Ziffernmenge ist erforderlich, falls die Inversion von Gruppenelementen nicht effizient ist (allgemeine Gruppen).

Die zweite Ziffernmenge wird bei elliptischen Kurven verwendet, weil man somit Platz und Laufzeit sparen kann (später).

Allgemeiner: D besteht aus 0 und Vertreter aller Restklassen von $\mathbb{Z}$ modulo 2^w, die ungerade sind.

Die w-NAF

Recodierung der w-NAF

Eingabe: $n = \sum d_i 2^i \in \mathbb{N}$ mit $d_i \in D$ und
$d_i = 0 \vee d_i \neq 0 \Rightarrow d_i = d_{i+1} = \ldots = d_{i+w-1} = 0$

1. $z \leftarrow n$
2. $\ell \leftarrow 0$
3. while $z \neq 0$ do
   1. if z gerade then $d_\ell \leftarrow 0$
   2. else
      1. Sei $d_\ell \in D$ mit $d_\ell \equiv z \ mod(2^w)$
   3. $z = \frac{z - d_\ell}{2}$
   4. if $z \neq 0$ then $\ell = \ell + 1$

Ausgabe: „$(\sum_{i=0}^{\ell} d_i 2^i)$”

Verwendung der w-NAF

Verdopple-und-Addiere-oder-Subtrahiere mit w-NAF

EINGABE: $n = \sum_{i=0}^{\ell} d_i 2^i$ mit $d_i \in D$. Gruppe G und $P \in G$

1. $X \leftarrow 0$
2. für alle $d \in D \cap \mathbb{N}_{\geqslant 1}$, berechne und spreichere $d \cdot P$
3. For $i = \ell, \ell - 1, \ldots, 0$ do
   1. (if $i \neq \ell$ then $X \leftarrow 2 \cdot X$)
   2. if $d_i \neq 0$ then $X \leftarrow X + \mathit{Vorzeichen}(d_i)|d_i| \cdot P$

AUSGABE: $X = \sum_{i=0}^{\ell} d_i 2^i \cdot P = n \cdot P$

Übungen: (i) Korrektheit der w-NAF Entwicklung (also: Korrektheit des Ergebnisses und Terminierung des Algorithmus); (ii) zeigen, dass die erwartete Dichte $\frac{1}{w+1}$ ist; (iii) was ist der Vorteil der 2. Ziffernmenge bei elliptischen Kurve ?

Ist das gut?

Der Einfachheit halber betrachten wir nur den Fall $D \subseteq \mathbb{N}_0$ also: $D = \{0\} \cup \{1, 3, 5, \ldots, 2^w - 1\}$. In diesem Fall, mit positiven Skalaren, gibt es nur Additionen. Um alle $d \cdot P$ mit $d = 3, 5, \ldots, 2^{w-1}$ brauchen wir 2^{w-1} Operationen (man rechnet zuerst $2 \cdot P$ und dann fährt man weiter in Zweierschritten fort).

Man braucht w, derart dass

$$\ell + 2^{w-1} + \frac{1}{w+1}\ell$$

minimiert wird. Für w gilt $w_{\text{min}} \approx \log_2(\ell) - 2\log_2(\log_2(\ell))$ (cfr. Knuth, ACP 2).

Erdös hatte bewiesen: *Für fast alle ganze Zahlen n, die minimale Anzahl von Operationen, die nötig ist, um $n \cdot P$ zu rechnen, ist* $\lambda(n) + \lambda(n)/\lambda(\lambda(n))$ *wobei* $\lambda(n) = \lfloor \log_2(n) \rfloor$.

Mit der w-NAF, obiger Ziffernmenge und $w = w_{\text{min}}$, erreicht man den fast optimalen Wert $\lambda(n) + (1 + o(1))\lambda(n)/\lambda(\lambda(n))$.

Sei $\ell = 256$. Anzahl Ops mit Binärdarstellung: $256 + \frac{1}{2}256 = 384$; Mit NAF=2-NAF: $256 + \frac{1}{2}256 = 341.333$; Mit w-NAF: für $w = 3, 4, 5, 6$ ist diese Anzahl 324, 315.2, 314.7, 324.7. **Min mit 5-NAF!**

Kryptographische Anwendungen von Elliptischen Kurven.

Diffie-Hellman Schlüsselaustausch (1976)

Öffentliche Parameter: Zyklische Gruppe G (z.B. $\leqslant E(\mathbb{F}_q)$) der Ordnung ℓ (Primzahl), mit Generator P.

Protokoll Diffie-Hellman Schlüsselaustausch

EINGABE: G, P

1. Alice wählt $\alpha \in [1..\ell - 1]$ zufällig und schickt $\alpha \cdot P$ an Bob.
2. Bob wählt $\beta \in [1..\ell - 1]$ zufällig und schickt $\beta \cdot P$ an Alice.
3. Alice berechnet $\alpha \cdot (\beta \cdot P)$, Bob analog berechnet $\beta \cdot (\alpha \cdot P)$.

Gemeinsamer geheimer DH-Schlüssel: $(\alpha\beta) \cdot P$.

- Angreifer Eve erhält P, $\alpha \cdot P$, $\beta \cdot P$.
- **Sicherheit:** Eve kann $(\alpha\beta) \cdot P$ nicht
  - berechnen; (Diffie-Hellman-Problem DH-Problem);
  - von einem $\gamma \cdot P$ mit zufälligem γ unterscheiden (Decisional Diffie-Hellman-Problem, DDH-Problem).

Das DH-Problem und Diskrete Logarithmen

Weiteres Problem: Discrete-Log Problem: Gegeben P und $Q \in \langle P \rangle$, finde t, derart dass $Q = t \cdot P$.

Wer das DL-Problem lösen kann, kann das DH-Problem lösen (trivial). Die Probleme sind für elliptische Kurven unter bestimmten Annahmen äquivalent.

- U.M. Maurer and S. Wolf: Diffie-Hellmann oracles: in Advances in cryptology - CRYPTO '96, Lect. Notes in Computer Science Vol. 1109(1996) pp. 268–282, Springer-Verlag.
- A. Muzereau, N.P. Smart, F. Vercauteren: The equivalence between the DHP and DLP for elliptic curves used in practical applications, LMS J. Comput. Math., 7(2004), 50–72.

Diskrete Logarithmen

Shoup-Nechaev, in black-box Gruppen primer Ordnung ℓ hat ein Algorithmus, dass das DLP mit Wahrscheinlichkeit $> 1/2$ löst, Komplexität $\Theta(\sqrt{\ell})$

Für vorsichtig gewahlte Kurven hat aber der beste bekannte Algorithmus zum lösen des DLPs Komplexität $O(\sqrt{\ell})$, wobei ℓ der größte Primfaktor der Gruppenordnung ist (Algorithmus von Silver-Pohling-Hellman - im Grunde genommen, eine Anwendung des chinesischen Restsatzes).

Infolgedessen wollen wir Kurven und Körper mit $\#E(\mathbb{F}_q)$ „quasi" eine Primzahl.

ECDSA I - Parameter

Variante des Signaturalgorithmus DSA mit ellitischen Kurven.
IEEE-Standard.

ECDSA:

Parameter (öffentlich): Kurve $E(\mathbb{F}_p)$, Punkt $P \in E(\mathbb{F}_p)$ der Ordnung ℓ (ℓ ist eine grosse Primzahl).

Privater Schlüssel: eine ganze Zahl α im Intervall $[2 \ldots \ell - 2]$

Öffentlicher Schlüssel: der Punkt Q mit $\alpha \cdot P = Q$.

$H(\cdot)$ ist eine vorhandene, feste Hashfunktion.

ECDSA II - Signaturerzeugung

Signaturerzeugung:
EINGABE: eine Nachricht m, Alices privater und öffentlicher Schlüssel α und Q.
AUSGABE: Alices elektronische Signatur (r, s)

1. A(lice) wählt $k \in [2 \ldots \ell - 2]$ zufällig.
2. A berechnet $k \cdot P = (x_1, y_1)$ und $r = x_1 \bmod \ell$. Falls $r = 0$ dann geht sie zu Schritt 1 zurück, da sonst die Signatur s, die in Schritt 3 berechnet wird, nicht vom privaten Schlüssel α abhängig wäre.
3. A berechnet $k^{-1} \bmod \ell$ und den Hashwert $H(m)$ der Nachricht m. Dann bildet sie $s = k^{-1}(H(m) + xr) \bmod \ell$.
4. Falls $s = 0$, beginnt sie wieder bei Schritt 1. Sonst sendet A die Nachricht m zusammen mit der Signatur (r, s) an B(ob).

ECDSA III - Signaturverifikation

Signaturverifikation:
EINGABE : Die Nachricht m, eine Signatur (r, s), Alices öffentlicher Schlüssel Q
AUSGABE : Die Signatur ist „korrekt“ oder „falsch“

1. B prüft, ob $r, s \in [1 \ldots \ell - 1]$.
2. B berechnet $w = s^{-1} \bmod \ell$ und $H(m)$. Dann bildet er $u_1 = H(m)w \bmod \ell$ und $u_2 = rw \bmod \ell$.
3. B ermittelt $u_1 \cdot P + u_2 \cdot Q = (x_0, y_0)$.
4. Bob entscheidet sich für „korrekt“(also akzeptiert die Signatur als gültig) genau dann, wenn $x_0 \bmod \ell = r$ ist, sonst für „falsch“.

ECDSA IV - Korrektheit und Sicherheit

Zur Korrektheit: Wir nehmen an, dass Alice die Signaturerzeugung korrekt ausgeführt hat. Dann berechnet Bob im dritten Schritt der Signaturverifikation

$$u_1 \cdot P + u_2 \cdot Q = (H(m)w \bmod \ell) \cdot P + (rw \bmod \ell) \cdot Q = (x_0, y_0) \ .$$

Nun gilt aber $\alpha \cdot P = Q$. Also erhält Bob

$$\begin{aligned}(H(m)w \bmod \ell) \cdot P + (rw \bmod \ell)\alpha \cdot P & \\ = ((H(m)w + rw\alpha) \bmod \ell) \cdot P & \\ = (w(H(m) + r\alpha) \bmod \ell) \cdot P = k \cdot P \ . &\end{aligned}$$

Deshalb muss $x_0 = r \bmod \ell$ gelten.

Zur Sicherheit: Falls ein beliebiger Benutzer C den diskreten Logarithmus berechnen kann, erhält er aus P und Q den geheimen Schlussel α und kann damit Signaturen von A fälschen.