Elliptiske kurver og kryptering

Henrik Lunding Nielsen, 20081795

25. august 2011

Abstract

This study investigates the subject of elliptic curves and how they are used in cryptography. Also an application to perform the Diffie-Hellman key exchange is included.

First, the method called *Index Calculus* is discussed. Next more general methods to find discrete logarithms is described in section 3.

Section 4 presents an application to establish a common key between two parts, Alice and Bob, and finally a method to create a digital signature using elliptic curves is described, as well as another application of elliptic curves, ECIES.

Most methods of cryptography that utilizes elliptic curves rely on the difficulty of finding discrete logs. The security of these methods can be compromised by solving these discrete logs. But if the elliptic curves and points on them are cleverly chosen (large enough), solving the discrete logs takes an enormous amount of time. In practise, its impossible to solve these discrete logs using any known method to date.

Indhold

1 Indledning

Kryptografi er nok den del af algebraen der har fundet størst praktisk anvendelse. Hver gang følsomme data sendes er der oftest en form for kryptering forbundet. Elliptiske kurver har fundet indpas i kryptering og er nu et af de primære værktøjer.

Styrken i mange kryptosystemer ligger i sværheden af det diskrete logaritmeproblem. Eg. lad p være et primtal, og a, b være heltal med $a \not\equiv 0 \not\equiv b \pmod{p}$. Antag yderligere at der findes et $k \in \mathbb{N}$ sådan at

$$a^k \equiv b \pmod{p} \tag{1}$$

Problemet er så at finde k. Hvis tallene er store kan det ikke klares i noget overskueligt tidsrum (med nogen metode vi kender i dag). Metoden kaldet *Index Calculus* er den bedst kendte til finde diskrete logaritmer over $\mathbb{F}_p$. Men denne metode virker ikke over elliptiske kurver.

2 Index Calculus

I dette afsnit beskrives den bedst kendte metode til at finde diskrete logaritmer (i $\mathbb{F}_p$). Den kræver at elementerne kan primfaktoriseres. Vi skal benytte os af Smooth Numbers, og der præsenteres den hurtige algoritme til af finde disse diskrete logaritmer.
π bliver brugt som funktionen, der giver antallet af primtal op til et givent tal, f.eks. $\pi(11) = 5$. Derudover benyttes $o(g)$-, $O(g)$- og $\Theta(g)$-notation (hvor g er positiv for store x):

$$\begin{aligned} f = \Theta(g) &\quad\Leftrightarrow\quad \exists c, d \in \mathbb{R}^+ : cg(x) \leq f(x) \leq dg(x) \quad \text{for store } x \\ f = O(g) &\quad\Leftrightarrow\quad \exists c \in \mathbb{R}^+ : |f(x)| \leq cg(x) \quad \text{for store } x \\ f = o(g) &\quad\Leftrightarrow\quad \frac{f(x)}{g(x)} \to 0 \quad \text{for } x \to \infty \end{aligned}$$

2.1 Smooth Numbers

Definition 2.1.
Antag $m \in \mathbb{N}$ og $y \in \mathbb{R}^+$.
*Hvis alle primtal, der går op i m højst er y, så siges m at være **y-smooth**.*

Og hvis $0 \leq y \leq x$, så defineres $\Psi(y, x)$ til at være antallet af y-smooth tal mindre end eller lig x:

$$\Psi(y, x) := |\{m \in \mathbb{N} | m \leq x, m \text{ } y\text{-smooth}\}| \quad for \quad 0 \leq y \leq x$$

Theorem 2.2.
Lad $y : \mathbb{R}^+ \to \mathbb{R}^+$ være en funktion af x, sådan at:

$$\frac{y}{\log x} \to \infty \quad og \quad \frac{\log x}{\log y} \to \infty \; for \; x \to \infty$$

Så gælder for stort x at:

$$\Psi(y, x) \geq x \cdot \exp\left((-1 + o(1)) \frac{\log x}{\log y} \log\log x\right)$$

Bevis. Vi ser først, at $\Psi(y, x)$ giver mening fordi $y \leq y^{\frac{\log x}{\log y}} = x$ for stort x.
Vi skriver $u := \frac{\log x}{\log y} = \lfloor u \rfloor + \delta$ hvor $\lfloor u \rfloor$ er u rundet ned, og $0 \leq \delta < 1$.
Nu deler vi så primtallene (og 1), der er højst y, op i to mængder:

$$V = \{p \text{ primtal } | p \leq y^\delta/2\} \cup \{1\} \tag{2}$$
$$W = \{p \text{ primtal } | y^\delta/2 < p \leq y\} \tag{3}$$

Bertrands postulat[1] siger, at for ethvert $m \in \mathbb{N}$ gælder at $\pi(m) - \pi(m/2) > \frac{m}{6 \log m}$. Så har vi:

$$\begin{aligned} |W| &= |\{p \text{ primtal } | y^\delta/2 < p \leq y\}| = \pi(y) - \pi(y^\delta/2) \\ &\geq \pi(y) - \pi(y/2) > \frac{1}{6}\frac{y}{\log y} = \frac{1}{6}\frac{y}{\log x}\frac{\log x}{\log y} \\ &= \frac{1}{6}\frac{y}{\log x} u \geq 2\lfloor u \rfloor \quad \text{for stort } x \text{ (da } \tfrac{y}{\log x} \to \infty) \end{aligned} \tag{4}$$

Definér nu mængden:

$$S := \{w_1 w_2 ... w_{\lfloor u \rfloor} v | w_1, w_2, ..., w_{\lfloor u \rfloor} \in W \text{ indbyrdes forskellige}, v \in V\}$$

Det ses at S indeholder $\begin{pmatrix} |W| \\ \lfloor u \rfloor \end{pmatrix} |V|$ elementer, der alle er y-*smooth* (da alle primfaktorerne $w_1, ..., w_{\lfloor u \rfloor}, v$ højst er y). Desuden er tallene heri højst x:

$$\begin{aligned} w_1 w_2 ... w_{\lfloor u \rfloor} v &\leq (\max\{w \in W\})^{\lfloor u \rfloor} \max\{v \in V\} \\ &\leq y^{\lfloor u \rfloor} \frac{y^\delta}{2} \leq y^u = y^{\frac{\log x}{\log y}} = x \end{aligned}$$

[1] Se Victor Shoup: A Computational Introduction(...), Theorem 5.8

(Her blev brugt at alle $w \in W$ højst er y, og at alle $v \in V$ højst er $\frac{y^\delta}{2}$; se (2) og (3)). Så alle $s \in S$ er i mængden $\{m \in \mathbb{N} | m \leq x, m \text{ } y\text{-smooth}\}$. Men så må $\Psi(y,x) \geq |S|$.

$$\begin{aligned}
\Psi(y,x) \geq |S| &= \binom{|W|}{\lfloor u \rfloor} |V| \\
&= \frac{|W|}{\lfloor u \rfloor} \cdot \frac{|W|-1}{\lfloor u \rfloor - 1} \cdot \frac{|W|-2}{\lfloor u \rfloor - 2} \cdot \ldots \cdot \frac{|W| - (\lfloor u \rfloor - 1)}{1} \cdot |V| \\
&\geq \left(\frac{|W|}{\lfloor u \rfloor} \right)^{\lfloor u \rfloor} \cdot |V| \geq \left(\frac{|W|}{2u} \right)^{\lfloor u \rfloor} \cdot |V| \quad \text{for stor } x \text{ (så } \lfloor u \rfloor \geq 1)
\end{aligned}$$

Anden sidste ulighed gælder fordi $\frac{|W|}{\lfloor u \rfloor} \leq \frac{|W|-1}{\lfloor u \rfloor - 1} \leq \ldots \leq \frac{|W|-(\lfloor u \rfloor - 1)}{1}$ (fordi $|W| \geq 2\lfloor u \rfloor \geq \lfloor u \rfloor$ for stort x (pr. (4))).
Da $|W| \geq \frac{1}{6} \frac{y}{\log y}$, (4), og $u = \frac{\log x}{\log y}$ kan vi omskrive lidt mere:

$$\begin{aligned}
\Psi(y,x) &\geq \left(\frac{|W|}{2u} \right)^{\lfloor u \rfloor} \cdot |V| \\
&\geq \left(\frac{y}{12u \log y} \right)^{\lfloor u \rfloor} \cdot |V| \\
&= \left(\frac{y}{12 \log x} \right)^{u-\delta} \cdot |V|
\end{aligned}$$

Når vi tager logaritmen får vi:

$$\begin{aligned}
\log(\Psi(y,x)) &\geq \log \left(\left(\frac{y}{12 \log x} \right)^{u-\delta} \cdot |V| \right) \\
&= (u-\delta)(\log y - \log 12 - \log\log x) + \log |V| \\
&= u \log y - u \log 12 - u \log\log x \\
&\quad - \delta \log y + \delta \log 12 + \delta \log\log x + \log |V| \\
&= \log x - u \log\log x + (\log |V| - \delta \log y) + O(u + \log\log x) \quad (5)
\end{aligned}$$

Chebyshev's theorem[2] siger, at $\pi(x) = \Theta\left(\frac{x}{\log(x)}\right)$. Det vil sige:

$$|V| = \pi(y^\delta/2) + 1 = \Theta\left(\frac{y^\delta/2}{\log(y^\delta/2)} \right)$$

[2]Se Victor Shoup: A Comptational Introduction(...), Theorem 5.1

Så for en konstant C gælder, at når x er stor, så er:

$$\begin{aligned}
&\left(\frac{C}{2}\right)\frac{y^\delta}{\log y} = C\frac{y^\delta/2}{\log(y)} \leq C\frac{y^\delta/2}{\log(y^\delta/2)} \leq |V| \leq \frac{y^\delta}{2} \leq y^\delta \\
&\Rightarrow \frac{C}{2}\frac{1}{\log y} \leq \frac{|V|}{y^\delta} \leq 1 \\
&\Rightarrow \log\left(\frac{C}{2}\right) - \log\log y \leq \log|V| - \delta\log y \leq 0 \\
&\Rightarrow |\log|V| - \delta\log y| \leq \left|\log\left(\frac{C}{2}\right) - \log\log y\right| \\
&\qquad = \left|\frac{\log\left(\frac{C}{2}\right)}{\log\log y} - 1\right|\log\log y \\
&\qquad \leq 2\log\log y \quad \text{for stort } x
\end{aligned}$$

(obs: $\log\log y$ er positiv når x er stor nok). Vi har nu at $\log|V| - \delta\log y$ er en $O(\log\log y)$-funktion. Vi bruger dette sammen med lemma 2.3 til at konkludere at $(\log|V| - \delta\log y) + O(u + \log\log x)$ er en $o(u\log\log x)$-funktion. Vi fortsætter så udregningerne fra (5):

$$\begin{aligned}
\log(\Psi(y,x)) &\geq \log x - u\cdot\log\log x + (\log|V| - \delta\cdot\log y) \\
&\quad + O(u + \log\log x) \\
&= \log x - u\cdot\log\log x + o(u\cdot\log\log x) \\
&= \log x + (-1 + o(1))u\cdot\log\log x
\end{aligned}$$

Men dette betyder netop:

$$\Psi(y,x) \geq x\cdot\exp((-1+o(1))u\cdot\log\log x)$$

□

Lemma 2.3.
Hvis en funktion f er en $O(u + \log\log x)$- eller en $O(\log\log y)$-funktion, så er f en $o(u\log\log x)$-funktion.

Bevis. Antag først at f er en $O(u + \log\log x)$-funktion (så $|f(x)| \leq c(u + \log\log x)$ når x er stor). Så får vi at:

$$\begin{aligned}
\frac{|f(x)|}{u\log\log x} &\leq \frac{c(u+\log\log x)}{u\log\log x} \\
&= \frac{cu}{u\log\log x} + \frac{c\log\log x}{u\log\log x} \\
&= \frac{c}{\log\log x} + \frac{c}{u} \to 0 \quad \text{for } x\to\infty
\end{aligned}$$

Og dermed er $f(x) = o(u \cdot \log\log x)$

Antag nu at f er en $O(\log\log y)$-funktion. Så udledes på samme måde:

$$\begin{aligned}\frac{|f(x)|}{u\log\log x} &\leq \frac{c\log\log y}{u\log\log x} = \frac{c\log\left(\frac{\log x}{u}\right)}{u\log\log x} \\ &= \frac{c\log\log x}{u\log\log x} - \frac{c\log u}{u\log\log x} \\ &= \frac{c}{u} - \frac{\log u}{u}\frac{c}{\log\log x} \to 0 \quad \text{for } x \to \infty\end{aligned}$$

□

2.2 Algoritme SEDL

Her beskrives en algoritme, kaldet SEDL (split exponent discrete logarithm), der kan løse diskrete logaritme på subexponential tid.
Input til algoritmen er p og q primtal samt γ og α hvorom følgende gælder:

$$q|(p-1) \quad \text{og} \quad q \nmid \frac{p-1}{q} =: m \tag{6}$$

$$\gamma \in \mathbb{Z}_p^* \text{ genererer undergruppen } \{\gamma^n | 1 \leq n \leq q\} =: G \text{ af orden } q \tag{7}$$

$$\alpha \in G \tag{8}$$

Algoritmen returnerer det x som opfylder $\alpha = \gamma^x$.

Algoritmen starter således:
1) Lad y være et heltal mindre end p
(senere finder vi ud af at vælge y bedre, så algoritmen bliver hurtigere)
2) Lad $p_1, p_2, ..., p_k$ være primtallene op til y
Så nu har vi at $p_1 < p_2 < ... < p_k \leq y \leq p$.
3) Sæt endvidere $\pi_i = [p_i]_p \in \mathbb{Z}_p^*$. Så π_i er restklassen der indeholder p_i.
4) For alle $i \in \{1, 2, ..., k+1\}$ gentag (5)-(7):
5) Vælg $r_i, s_i \in \{0, ..., q-1\}$ og $\overline{\delta_i} \in \mathbb{Z}_p^*$ tilfældigt
6) Beregn $\delta_i = \overline{\delta_i}^q, m_i = \text{rep}(\gamma^{r_i}\alpha^{s_i}\delta_i)$
Her er rep() en funktion der giver representanten af klassen, der er positiv, og tættest på 0. (f.eks $\text{rep}([i]_p) = i$ hvis $0 \leq i < p$). Nu giver Lemma 2.5 at $\gamma^{r_i}\alpha^{s_i}\delta_i$ er tilfældig i $\mathbb{Z}_p^*$; så repræsentationen m_i er også tilfældig i $\{1, 2, ..., p-1\}$.
7) Forsøg at primfaktorisere m_i med $p_1, ..., p_k$. Start over fra (5) hvis det mislykkedes (med samme i)

Så nu har vi $m_i = p_1^{e_{i,1}} p_2^{e_{i,2}} ... p_k^{e_{i,k}}$ for $i = 1, 2, ..., k+1$. Dvs:

$$\begin{aligned} \gamma^{r_i}\alpha^{s_i}\delta_i = [m_i]_p &= [p_1^{e_{i,1}} p_2^{e_{i,2}} ... p_k^{e_{i,k}}]_p \\ &= [p_1]_p^{e_{i,1}} [p_2]_p^{e_{i,2}} ... [p_k]_p^{e_{i,k}} \\ &= \pi_1^{e_{i,1}} \pi_2^{e_{i,2}} ... \pi_k^{e_{i,k}} \end{aligned} \tag{9}$$

8) Lad $\nu_i = (e_{i,1}, ..., e_{i,k}) \in \mathbb{Z}^k$, og $\overline{\nu_i} = ([e_{i,1}]_q, [e_{i,2}]_q, ..., [e_{i,k}]_q) \in \mathbb{Z}_q^k$ for $i = 1, 2, ..., k+1$

$\mathbb{Z}_q^k$ er et vektorrum. Så $\overline{\nu_1}, \overline{\nu_2}, ..., \overline{\nu_{k+1}}$ er lineært afhængige. Så der findes $c_1, c_2, ..., c_{k+1} \in \{0, 1, ..., q-1\}$ hvor ikke alle er 0, sådan at:

$$[\mathbf{0}]_q = [c_1]_q \overline{\nu_1} + [c_2]_q \overline{\nu_2} + ... + [c_{k+1}]_q \overline{\nu_{k+1}} \tag{10}$$

9) Benyt udvidet Gauss-elimination på $\left(\overline{\nu_1}^T, \overline{\nu_2}^T, ..., \overline{\nu_{k+1}}^T\right)$ for først at finde nulrummet for denne matrice, og derefter et (ikke-trivielt) sæt $c_1, c_2, ..., c_{k+1}$ hvor ovenstående er opfyldt.
Husk at ovenstående (10) er en vektor med k indgange. Den j'te indgang er:

$$\begin{aligned} [0]_q &= [c_1]_q [e_{1,j}]_q + [c_2]_q [e_{2,j}]_q + ... + [c_{k+1}]_q [e_{k+1,j}]_q \\ &= [c_1 e_{1,j} + c_2 e_{2,j} + ... + c_{k+1} e_{k+1,j}]_q \\ \Rightarrow & q\mathbb{Z} \ni c_1 e_{1,j} + c_2 e_{2,j} + ... + c_{k+1} e_{k+1,j} =: e_j \end{aligned} \tag{11}$$

Og så må $c_1\nu_1 + c_2\nu_2 + ... + c_{k+1}\nu_{k+1} = (e_1, e_2, ..., e_k) \in q\mathbb{Z}^k$.

10) Definér nu $r := \sum_{i=1}^{k+1} c_i r_i$, $s := \sum_{i=1}^{k+1} c_i s_i$, $\delta := \prod_{i=1}^{k+1} \delta_i^{c_i}$.
Så får vi (ved at benytte (9)):

$$\begin{aligned} \gamma^r\alpha^s\delta = \gamma^{\sum_{i=1}^{k+1} c_i r_i} \alpha^{\sum_{i=1}^{k+1} c_i s_i} \prod_{i=1}^{k+1} \delta_i^{c_i} &= \prod_{i=1}^{k+1} \gamma^{c_i r_i} \alpha^{c_i s_i} \delta_i^{c_i} \\ = \prod_{i=1}^{k+1} \left(\gamma^{r_i}\alpha^{s_i}\delta_i\right)^{c_i} &= \prod_{i=1}^{k+1} \left(\pi_1^{e_{i,1}} \pi_2^{e_{i,2}} ... \pi_k^{e_{i,k}}\right)^{c_i} \\ = \pi_1^{\sum_{i=1}^{k+1} c_i e_{i,1}} \pi_2^{\sum_{i=1}^{k+1} c_i e_{i,2}} ... \pi_k^{\sum_{i=1}^{k+1} c_i e_{i,k}} \\ = \pi_1^{e_1} \pi_2^{e_2} ... \pi_k^{e_k} \end{aligned}$$

Vi skriver nu $H := \{x^q | x \in \mathbb{Z}_p^*\}$. Da e_i'erne var et multiplum af q (det har vi fra (11)), må $\pi_i^{e_i}$ og dermed $\pi_1^{e_1}\pi_2^{e_2}...\pi_k^{e_k}$ være et element fra H. Og vi ved også at $\delta \in H$ (da alle $\delta_i = \overline{\delta_i}^q \in H$), og så må $\gamma^r\alpha^s \in H$.
Men vi ved fra lemma 2.5 at $\gamma^{r_i}\alpha^{s_i} \in G$ og dermed $\gamma^r\alpha^s \in G$, så der må gælde at $\gamma^r\alpha^s \in G \cap H = \{1\} \Rightarrow \gamma^r\alpha^s = [1]_p$ (lemma 2.4).

11) Hvis $s \equiv 0 \pmod{q}$ stopper algoritmen (fejl!)

Sandsynligheden for at algorimen stopper her er $\frac{1}{q}$ (hvilket er acceptabelt da q typisk er meget stor). Dette vil dog ikke blive vist her. Se evt. Victor Shoup's: A Computational Introduction(...), Lemma 15.2.
Så antag nu, at $s \not\equiv 0 \pmod{q}$. Så hvis vi definerer $s' := s^{q-2}$, så er $[s']_q$ det inverse element af $[s]_q$ fordi $ss^{q-2} = s^{q-1} \equiv 1 \pmod{q}$ (pr Fermats lille sætning). Sagt på en anden måde, $ss' = nq + 1$ for et naturligt tal n.

$$\begin{aligned} & [1]_p = \gamma^r \alpha^s \\ \Rightarrow \quad & [1]_p = \gamma^{rs'} \alpha^{ss'} = \gamma^{rs'} (\alpha^q)^n \alpha = \gamma^{rs'} \alpha \\ \Rightarrow \quad & \alpha = \gamma^{-rs'} \end{aligned}$$

Hvor $\alpha^q = [1]_p$ følger af, at $\alpha \in G$ og $\text{ord}(G) = q$
Så $-rs'$ er den diskrete logaritme af α mht. γ

12) Returner $-rs'$

Lemma 2.4.
Hvis G er givet som i (7), og $H = \{x^q | x \in \mathbb{Z}_p^*\}$ for samme primtal p og q, der opfylder (6), så er $\text{ord}(H) = m$ *og $G \cap H = \{[1]_p\}$ (hvor $m = \frac{p-1}{q}$)*

Bevis. $\mathbb{Z}_p^*$ er cyklisk[3] og har endelig orden $p - 1$. Der gælder $\mathbb{Z}_p^* \cong \mathbb{Z}_q^* \times \mathbb{Z}_m^*$ hvor $m \nmid q$ (pr. (6)). Og der findes netop én subgruppe, som har $\frac{p-1}{q} = m$ som orden, og denne gruppe er netop $\{x^q | x \in \mathbb{Z}_p^*\} = H$ (se[4]). Så ordnen af H er m.

Alle elementer $g \in G, h \in H$ opfylder $\text{ord}(g) \mid \text{ord}(G) = q$ og $\text{ord}(h) \mid \text{ord}(H) = m$, fordi ordnen af et gruppeelement deler ordnen af gruppen, men det vil sige at et element $a \in G \cap H$ har en orden der deler både q og m. Men så skal vi hvertfald have at $\text{ord}(a) \mid \gcd(q, m) = 1$ (fordi q er et primtal og $q \nmid m$). Så ordnen af alle elementer i $G \cap H$ er 1, men det er jo kun $[1]_p$ der har orden 1. Dvs: $G \cap H = \{[1]_p\}$.

□

Lemma 2.5.
Lad p og q primtal samt γ og α opfylde (6), (7), (8) og lad $u, v \in \{0, 1, ..., q-1\}$ og $w \in \mathbb{Z}_p^*$ være tilfældige.
Så gælder der at $\gamma^u \alpha^v \in G$ og at $\gamma^u \alpha^v w^q$ er tilfældig i $\mathbb{Z}_p^*$

[3]Se Victor Shoup: A Computational Introduction(...), Theorem 7.28
[4]Se Victor Shoup: A Computational Introduction(...), Theorem 6.32(ii)

Bevis. Lemma 2.4 giver at H har orden $m := \frac{p-1}{q}$.

I første del vises, at w^q er tilfældig i H. Lad x være en generator for $\mathbb{Z}_p^*$ (der findes en, da $\mathbb{Z}_p^*$ er cyklisk). Så er $x^s = w$ for et tal s, som vi kan skrive $s = nm + l$, for heltal n, q der opfylder $0 \leq n < q$ og $0 \leq l < m$. Vi regner på w^q og bruger at $x^{n(p-1)} = [1]_q$ (fermats lille sætning):

$$w^q = x^{sq} = x^{(nm+l)q} = x^{nmq+lq} = x^{n(p-1)}x^{lq} = (x^q)^l$$

Så w^q er altså en af elementerne $(x^q)^0, (x^q)^1, ..., (x^q)^{m-1}$. Alle disse er forskellige, og de er indeholdt i $H := \{x^q | x \in \mathbb{Z}_p^*\}$. Men da $\text{ord}(H) = m$, må de præcist udgøre H.
w var tilfældig i $\mathbb{Z}_p^*$. Dermed må s være tilfældig i $\{0, 1, ..., p-2\}$ (jeg tillader mig at skrive tilfældig, selvom s afhænger af w. Det der menes er, at man kan skabe et tilfældigt s ved at lade w være tilfældig og derefter finde det tilsvarende s). Men så er l også tilfældig i $\{0, 1, ..., m-1\}$. Og så må $(x^q)^l = w^q$ være tilfældig i H.

Nu kigger vi på $\gamma^u\alpha^v = \gamma^u\gamma^z = \gamma^{u+z}$ for et tal z. Det er nok at vide at u er tilfældig i $\{0, 1, ..., q-1\}$ og at $\text{ord}(G) = q$ for at indse, at $\gamma^{u+z} = \gamma^u\alpha^v$ er tilfældig i G.

Vi får nu vha ([5]) og lemma (2.4), at afbildningen $\rho : G \times H \to \mathbb{Z}_p^*$ givet ved $\rho(g, h) = gh$ er en isomorfi og dermed injektiv. Så ρ sender over i $qm = p-1$ forskellige elementer i $\mathbb{Z}_p^*$. Men da der kun er $p-1$ elementer heri, må ρ være bijektiv. Så hvis man vælger et $g \in G$ og $h \in H$ tilfældig vil gh også være tilfældigt i $\mathbb{Z}_p^*$.
Men så er $\rho(\gamma^u\alpha^v, w^q) = \gamma^u\alpha^v w^q$ tilfældig i $\mathbb{Z}_p^*$, som skulle vises. □

2.3 Kørselstiden for SEDL

Hvad er den forventede kørselstid for algoritmen?
At teste om et tal m_i er y-smooth tager $O(k \cdot (\text{len}\, p)^c)$ tid (der er k primtal), for en konstant c, og hvis vi lader $\sigma = \frac{\textit{antal y-smooth tal mindre end p}}{p-1} = \frac{\Psi(y,p-1)}{p-1}$ være sandsynligheden for at et tilfældigt tal mindre end p er y-smooth, må vi forvente at skulle gentage testen med nye m_i'er σ^{-1} gange, før vi finder et, som er y-smooth; så vi forventer at det tager $O\left(\frac{k}{\sigma}(\text{len}\, p)^c\right)$ tid at finde et m_i, der er y-smooth. Da dette gentages for $i = 1, 2, ..., k$ må den endelige forventede kørselstid for første del af algoritmen være $O\left(\frac{k^2}{\sigma}(\text{len}\, p)^c\right)$.

[5] Se Victor Shoup: A Computational Introduction(...), Theorem 6.25

Næste del af algoritmen er en (udvidet) gauss-elimination af matricen:

$$\begin{pmatrix} [e_{1,1}]_q & [e_{2,1}]_q & \dots & [e_{k+1,1}]_q \\ [e_{1,2}]_q & [e_{2,2}]_q & \dots & [e_{k+1,2}]_q \\ \dots & \dots & \dots & \dots \\ [e_{1,k}]_q & [e_{2,k}]_q & \dots & [e_{k+1,k}]_q \end{pmatrix}$$

Vi finder derved nulrummet for ovenstående matrice (som ikke kun består af $\mathbf{0}$). Så vi tager et ikke-trivielt $(c_1, c_2, ..., c_q)$ fra nulrummet, sådan at $c_1[\bar{\nu}_1]_q + c_2[\bar{\nu}_2]_q + ... + c_{k+1}[\bar{\nu}_{k+1}]_q = 0$. Denne Gauss-elimination tager $O(k^3 \cdot (\operatorname{len} p)^c)$ tid, da vi skal regne på elementer fra $\mathbb{Z}_q$. ($O((\operatorname{len} p)^c)$ er tiden det tager at finde en representant for $[\bar{\nu}_i]_q$). Den forventede kørselstid for algoritmen skal være positiv, så vi kan skrive:

$$E[Z] \leq \left(\frac{k^2}{\sigma} + k^3\right) O((\operatorname{len} p)^c) = \left(\frac{k^2}{\sigma} + k^3\right) (\operatorname{len} p)^{O(1)}$$

Lad os nu antage at vi i starten valgte $y < p$ således:

$$y(p) = \exp\left((\log p)^{\lambda + o(1)}\right) \quad \text{for } \lambda \in]0, 1[\tag{12}$$

Vi kan yderligere antage at p er stor, og så følger kørselstiden (som funktion af p) af theorem 2.8. Til dette skal vi bruge lemma 2.7:

Lemma 2.6. *(sublemma til 2.7)*
Hvis et tal y skrives i n-talssystemet som $x_1x_2...x_k$, så gælder at:

$$\log((\operatorname{len} y)^{O(1)}) = O(1) \log\log y + o(1)$$

Bevis.
Hvis y er skrevet i n-talssystemet må der gælde at $\operatorname{len} y = k = \lfloor \log_n y \rfloor + 1$ hvor $\lfloor \times \rfloor$ er en funktionen der runder ned til nærmeste hele tal. Vi kan så omskrive $\operatorname{len} y$:

$$\begin{aligned} \operatorname{len} y &= \lfloor \log_n y \rfloor + 1 \\ &= \log_n y - (\log_n y - \lfloor \log_n y \rfloor) + 1 \\ &= \log_n y + O(1) = (\log_n y)\left(1 + \frac{O(1)}{\log_n y}\right) \\ &= (\log_n y)(1 + o(1)) \end{aligned}$$

Lad nu f være følgende $O(1)$-funktion: $f(y) := 1 - \frac{\log\log n}{\log\log y}$. Så får vi: $\log(\log_n y) = \log\left(\frac{\log y}{\log n}\right) = \log\log y - \log\log n = f(y)\log\log y = \log((\log y)^{f(y)}) \Rightarrow \log_n y =$

$(\log y)^{f(y)} = (\log y)^{O(1)}$. Dette benytter vi til at regne videre (og husker samtidig at $x^c \to 1$ når $x \to 1$ hvis c er en konstant (eller en $O(1)$-funktion)):

$$\begin{aligned}
\operatorname{len} y &= (\log y)^{O(1)}(1+o(1)) \\
\Rightarrow (\operatorname{len} y)^{O(1)} &= (\log y)^{O(1)O(1)}(1+o(1))^{O(1)} \\
&= (\log y)^{O(1)}\,(1+o(1)) \\
\Rightarrow \log\left((\operatorname{len} y)^{O(1)}\right) &= \log\left((\log y)^{O(1)}(1+o(1))\right) \\
&= \log\left((\log y)^{O(1)}\right) + \log(1+o(1)) \\
&= O(1)\log\log y + o(1)
\end{aligned}$$

□

Lemma 2.7.
Lad p være et stort primtal ($log(log(p)) \geq 1$), og lad y og σ opfylde:

$$y < p \tag{13}$$

$$\log y = (\log p)^{\lambda+o(1)} \quad \textit{for et } \lambda \in]0,1[\tag{14}$$

$$\sigma = \frac{\Psi(y, p-1)}{p-1} \tag{15}$$

Lad så $k = \pi(y)$ være antallet af primtal op til y. Hvis vi så har en algoritme med forventet kørselstid

$$E[Z] \leq \left(\frac{k^2}{\sigma} + k^3\right)(\operatorname{len} p)^{O(1)} \tag{16}$$

Så gælder der om kørselstiden at:

$$E[Z] \leq \exp\left((1+o(1))\max\left(\frac{\log p}{\log y}\log\log p + 2\log y, 3\log y\right)\right) \tag{17}$$

Bevis.
Vi benytter (14) samt lemma 2.6, så:

$$\begin{aligned}
\frac{\log\left((\operatorname{len} p)^{O(1)}\right)}{\log y} &= \frac{O(1)\log\log p + o(1)}{(\log p)^{\lambda+o(1)}} \to 0 \quad \text{for } p \to \infty \\
\frac{\log\left((\operatorname{len} p)^{O(1)}\right)}{\frac{\log p}{\log y}} &= \frac{O(1)\log\log p + o(1)}{(\log p)^{1-\lambda-o(1)}} \to 0 \quad \text{for } p \to \infty
\end{aligned}$$

Sammenholdes dette, får vi:

$$\frac{\log\left((\operatorname{len} p)^{O(1)}\right)}{\min\left(\log y, \frac{\log p}{\log y}\right)} \to 0 \quad \text{for } p \to \infty$$
$$\Rightarrow \log\left((\operatorname{len} p)^{O(1)}\right) = o\left(\min\left(\log y, \frac{\log p}{\log y}\right)\right) \tag{18}$$

Dette resultat skal vi bruge senere.

Se nu på $k = \pi(y)$ (ifølge antalgelsen). Chebyshev's theorem giver os at $k = \pi(y) = \Theta\left(\frac{y}{\log y}\right)$; dvs. $\exists c, d : c\frac{y}{\log y} \leq k \leq d\frac{y}{\log y}$ hvis y er stor nok. Se samtidig at for et z konstant er $\log\left(z\frac{y}{\log y}\right) = \log z + \log y - \log\log y = (\log y)\left(\frac{\log z + \log y - \log\log y}{\log y}\right) = (\log y)(1 + o(1))$, og derfor får vi:

$$\log\left(c\frac{y}{\log y}\right) \leq \log k \leq \log\left(d\frac{y}{\log y}\right)$$
$$\Rightarrow (\log y)(1 + o(1)) \leq \log k \leq (\log y)(1 + o(1))$$
$$\Rightarrow \log k = (\log y)(1 + o(1)) \tag{19}$$

(Her er alle tre $o(1)$-funktioner forskellige).
Dette resultat skal også bruges senere.

Antagelse (14), giver os at $\log y = (\log p)^{\lambda + o(1)}$. Dvs:

$$\frac{y}{\log p} = \frac{y}{(\log y)^{1/(\lambda + o(1))}} = \left(\frac{y^{\lambda + o(1)}}{\log y}\right)^{\frac{1}{\lambda + o(1)}} \to \infty \quad \text{for } p \to \infty$$
$$\frac{\log p}{\log y} = \frac{\log p}{(\log p)^{\lambda + o(1)}} = (\log p)^{1 - \lambda - o(1)} \to \infty \quad \text{for } p \to \infty$$

Så gælder ifølge Theorem 2.2 at:

$$\Psi(y, p) \geq p \cdot \exp\left((-1 + o(1))\,\frac{\log p}{\log y}\log\log p\right)$$

Pr. antagelse (13) er $y < p$, og da p er et primtal kan p ikke være y-smooth. Så kan (15) omskrives, ved også at benytte ovenstående udtryk for

$\Psi(y,p)$:

$$\begin{aligned}\sigma &= \frac{\Psi(y,p-1)}{p-1} = \frac{\Psi(y,p)}{p-1} \geq \frac{\Psi(y,p)}{p} \\ &\geq \frac{p \cdot \exp\left((-1+o(1))\frac{\log p}{\log y}\log\log p\right)}{p} \\ &= \exp\left((-1+o(1))\frac{\log p}{\log y}\log\log p\right) \qquad (20)\end{aligned}$$

Nu regner vi på $\frac{k^2}{\sigma}$ og k^3. Vi starter ud med at bruge (19) og (20), og definerer $u = 2\log y + \frac{\log p}{\log y}\log\log p$, $v = 3\log y$ og $w = \log y + \frac{\log p}{\log y}\log\log p$ (Mest for at gøre senere udregninger overskuelige):

$$\begin{aligned}\frac{k^2}{\sigma} &= \frac{\exp(2\log k)}{\sigma} \qquad (21)\\ &\leq \frac{\exp(2(1+o(1))\log y)}{\exp\left((-1+o(1))\frac{\log p}{\log y}\log\log p\right)} \\ &= \exp\left(2(1+o(1))\log y - (-1+o(1))\frac{\log p}{\log y}\log\log p\right) \\ &= \exp\left(u + o(1)\log y + o(1)\frac{\log p}{\log y}\log\log p\right) \\ &\leq \exp(u + o(1)w) \qquad (22)\\ k^3 &= \exp(3\log k) = \exp(3(1+o(1))\log y) \\ &= \exp(v + o(1)\log y) \\ &\leq \exp(v + o(1)w) \qquad (23)\end{aligned}$$

Bemærk at ved tredje lighedstegn skiftede en $o(1)$-funktion fortegn. Og ved anden ulighed er der en helt ny $o(1)$-funktion, valgt til at være maximum af de to foregående. Nu kan vi begynde at omskrive den forventede kørselstid, (16):

$$E[Z] \leq \left(\frac{k^2}{\sigma} + k^3\right)(\operatorname{len} p)^{O(1)} \leq 2\max\left(\frac{k^2}{\sigma}, k^3\right)(\operatorname{len} p)^{O(1)}$$

Ved at vælge en større $O(1)$-funktion kan vi fjerne 2-tallet. Herefter indsættes

(22) og (23):

$$\begin{aligned}
E[Z] &\leq \max\left(\frac{k^2}{\sigma}, k^3\right)(\operatorname{len} p)^{O(1)} \\
&\leq \max(\exp(u + o(1)w), \exp(v + o(1)w))(\operatorname{len} p)^{O(1)} \\
&= \exp(\max(u + o(1)w, v + o(1)w))(\operatorname{len} p)^{O(1)} \\
&\leq \exp(\max(u, v) + o(1)w)(\operatorname{len} p)^{O(1)} \\
&= \exp\left(\max(u, v) + o(1)w + \log\left[(\operatorname{len} p)^{O(1)}\right]\right) \\
&= \exp\left(\max(u, v)\left(1 + \frac{o(1)w + \log\left[(\operatorname{len} p)^{O(1)}\right]}{\max(u, v)}\right)\right) \\
&\leq \exp\left(\max(u, v)\left(1 + \frac{o(1)w + \log\left[(\operatorname{len} p)^{O(1)}\right]}{u}\right)\right) \qquad (24)
\end{aligned}$$

Ved tredje ulighed vælges igen en ny $o(1)$-funktion, som maksimum af de to gamle. Lad os nu vise, at $\frac{o(1)w}{u}$ og $\frac{\log\left((\operatorname{len} p)^{O(1)}\right)}{u}$ er $o(1)$-funktioner. Den første er let - indsæt blot hvad u og w er defineret til:

$$\frac{o(1)w}{u} = \frac{\log y + \frac{\log p}{\log y}\log\log p}{2\log y + \frac{\log p}{\log y}\log\log p} \cdot o(1) \leq o(1) \to 0 \quad \text{for } p \to \infty$$

Til $\frac{\log\left((\operatorname{len} p)^{O(1)}\right)}{u}$ (som er positiv) skal vi bruge (18) (og u's definition):

$$\begin{aligned}
\frac{\log\left((\operatorname{len} p)^{O(1)}\right)}{u} &= \frac{o\left(\min\left(\log y, \frac{\log p}{\log y}\right)\right)}{2\log y + \frac{\log p}{\log y}\log\log p} \\
&\leq \frac{o\left(\min\left(\log y, \frac{\log p}{\log y}\right)\right)}{\log y + \frac{\log p}{\log y}} \\
&\leq \frac{o\left(\min\left(\log y, \frac{\log p}{\log y}\right)\right)}{\min\left(\log y, \frac{\log p}{\log y}\right)} \to 0 \quad \text{for } p \to \infty
\end{aligned}$$

(ved første ulighed bruges antagelsen $log(log(p)) \geq 1$)

Dermed er $\frac{o(1)w + \log\left((\operatorname{len} p)^{O(1)}\right)}{u}$ en $o(1)$-funktion, og (24) kan skrives:

$$E[Z] \leq \exp(\max(u, v)(1 + o(1)))$$

Indsættes udtrykkene fra u og v's definition får vi (17). Dette slutter beviser for lemma 2.7. □

Som udtryk for SEDL's kørselstid har vi nu:

$$E[Z] \leq \exp\left((1+o(1))\max\left(\frac{\log p}{\log y}\log\log p + 2\log y, 3\log y\right)\right) \qquad (25)$$

Vi vil minimere max()-funktionen. Hvis vi lader $\mu := \log y$, $A := \log p \log\log p$, $S_1 := \frac{A}{\mu} + 2\mu$ og $S_2 := 3\mu$, så skal vi bare minimere $\max(S_1, S_2)$:

$$\frac{dS_1}{d\mu} = -\frac{A}{\mu^2} + 2 = 0 \Rightarrow \mu_{min} = \sqrt{\frac{A}{2}}$$
$$\left.\frac{d^2S_1}{d\mu^2}\right|_{\mu_{min}} = \frac{2A}{\mu_{min}^3} > 0 \Rightarrow \mu_{min} \text{ er et minimum}$$

Så $S_1(\mu_{min}) = \frac{A}{\sqrt{\frac{A}{2}}} + 2\sqrt{\frac{A}{2}} = \sqrt{8}\sqrt{A}$ er den mindste værdi S_1 antager. Og da $S_2(\mu_{min}) = 3\sqrt{\frac{A}{2}} = \sqrt{\frac{9}{2}}\sqrt{A} < S_1(\mu_{min})$, må $\max(S_1, S_2)$ også være minimeret ved μ_{min}. Nu kan vi isolere y:

$$\log y = \mu_{min} = \frac{1}{\sqrt{2}}\sqrt{A} = \frac{1}{\sqrt{2}}\sqrt{\log p \log\log p}$$
$$\Rightarrow y = \exp\left(\frac{1}{\sqrt{2}}(\log p \log\log p)^{1/2}\right) \qquad (26)$$

men y skulle have formen $\exp\left((\log p)^{\lambda+o(1)}\right)$ (12). Dette indser vi at den har ved at vælge et λ og en $o(1)$-funktion f sådan at $\exp\left((\log p)^{\lambda+f}\right) = \exp\left(\frac{1}{\sqrt{2}}(\log p \log\log p)^{1/2}\right)$.
Start ud med at vælge $\lambda = \frac{1}{2}$ og f:

$$f := \frac{\log\left(\frac{1}{\sqrt{2}}(\log\log p)^{1/2}\right)}{\log\log p}$$

f er positiv for store p, og vi må have at $f \leq \frac{\log\log\log p}{\log\log p} \to 0$ for $p \to \infty$. Dvs f er en $o(1)$-funktion. Derudover gælder:

$$f = \frac{\log\left(\frac{1}{\sqrt{2}}(\log\log p)^{1/2}\right)}{\log\log p}$$
$$\Rightarrow \quad \log\left((\log p)^f\right) = f \cdot \log\log p = \log\left(\frac{1}{\sqrt{2}}(\log\log p)^{1/2}\right)$$
$$\Rightarrow \quad (\log p)^f = \frac{1}{\sqrt{2}}(\log\log p)^{1/2}$$
$$\Rightarrow \quad \exp\left((\log p)^{1/2+f}\right) = \exp\left(\frac{1}{\sqrt{2}}(\log p \log\log p)^{1/2}\right)$$

Så den er god nok! Indsæt y (26) i max-funktionen:

$$\begin{aligned}
&\max\left(\frac{\log p}{\log y}\log\log p + 2\log y, 3\log y\right)\\
&= \max\left(\sqrt{2}(\log p\log\log p)^{1/2} + \sqrt{2}(\log p\log\log p)^{1/2}, \frac{3}{\sqrt{2}}(\log p\log\log p)^{1/2}\right)\\
&= \max\left(2\sqrt{2}(\log p\log\log p)^{1/2}, \frac{3}{\sqrt{2}}(\log p\log\log p)^{1/2}\right)\\
&= 2\sqrt{2}(\log p\log\log p)^{1/2}
\end{aligned}$$

Ved at indsætte dette i (25) får vi:

Theorem 2.8.
Den forventede kørselstid for algoritme SEDL opfylder:

$$E[Z] \leq \exp\left((2\sqrt{2}+o(1))(\log p\log\log p)^{1/2}\right) \tag{27}$$

Hvis vi i (12) antog at y havde en anden form, ville vi ikke kunne være sikker på at (25) holder. Vi har kun fundet en øvre grænse for tiden, men det er en øvre grænse der er bedre end den tid nedenstående algoritmer tager.

3 Andre algoritmer til diskrete logaritmer

Hvis G er en mere generel gruppe af orden højst N, samt $P, Q \in G$ hvor $P^k = Q$ for et k kan vi ikke bruge ovenstående metode til at finde k. Algoritmerne i dette afsnit er i stand til at finde logaritmer i disse grupper, men de er til gengæld ikke lige så hurtig som Index Calculus. Da det er elliptiske kurver vi ser på (hvis komposition skrives '+') skriver vi fremover $kP = Q$ istedet for $P^k = Q$.

3.1 Baby step, Giant step

Denne algoritme kan bruges hvis G er cyklisk. For simpelhed skyld antages at P er en generator for G (ellers kan vi erstatte G med undergruppen genereret af P (Q er netop også i denne undergruppe)) Denne algoritme kræver $\sqrt{N}$ tid og $\sqrt{N}$ plads. Først skal man finde et N der er større eller lig ordnen af G (man kender højst sandsynligt ikke ordenen af G). Når vi regner med elliptiske kurver kan vi bruge Hasse's theorem:

Theorem 3.1 (Hasse's)**.**
Hvis E er en elliptisk kurve over $\mathbb{F}_q$, så gælder at ordnen af gruppen $E(\mathbb{F}_q)$ opfylder:

$$|q + 1 - \text{ord}\,(E(\mathbb{F}_q))\,| \leq 2\sqrt{q}$$

For et bevis, se *Lawrence C. Washington: Elliptic Curves, Number Theory and Cryptography, Second edition* s. 97-102.

Dette giver os at:

$$q+1-\text{ord}(E(\mathbb{F}_q)) \geq -2\sqrt{q} \Rightarrow q+2\sqrt{q}+1 \geq \text{ord}(E(\mathbb{F}_q))$$
$$q+1-\text{ord}(E(\mathbb{F}_q)) \leq 2\sqrt{q} \Rightarrow q-2\sqrt{q}+1 \leq \text{ord}(E(\mathbb{F}_q))$$

Det fremgår heraf at vi kan vælge N større end ordnen af gruppen ved $N := \lceil q+2\sqrt{q}+1 \rceil$. Ydermere gælder der så at:

$$\frac{N}{ord(E(\mathbb{F}_q))} \geq \frac{\lceil q+2\sqrt{q}+1 \rceil}{q+2\sqrt{q}+1} \to 1 \quad \text{for } q \to \infty$$
$$\frac{N}{ord(E(\mathbb{F}_q))} \leq \frac{\lceil q+2\sqrt{q}+1 \rceil}{q-2\sqrt{q}+1} \to 1 \quad \text{for } q \to \infty$$

Så der må ligeledes gælde at $\frac{N}{ord(E(\mathbb{F}_q))} \to 1$ for $q \to \infty$, så $N \sim ord(E(\mathbb{F}_q))$ (N er asymptotisk lig med ordnen af $E(\mathbb{F}_q)$)

Så for eliptiske grupper kan vi let finde et N der er mindst ordnen - endda et N der er asymptotisk lig med ordnen af gruppen. For andre cykliske grupper er det ikke sikkert det er lige så let. Man kan her blive nød at gætte. Hvis gættet er for lavt er der en risiko for at algoritmen ikke returnerer logaritmen, men så kan man blot lave et nyt gæt (her er det smartest blot at fordoble ens første gæt). At N er for lav får ikke algoritmen til at returnere en forkert logaritme, men det kan være den slet ikke returnerer noget.

Algoritmen kører således:

1. Vælg (eller gæt) et N, der er mindst ordnen af gruppen (ved eliptiske kurver, som beskrevet ovenfor)
2. Sæt $b = \lceil \sqrt{N} \rceil$ og $R = b(-P)$
3. Beregn, gem og sorter[6] en liste med elementerne $\{0, P, 2P, ..., (b-1)P\}$ Først gemmes det neutrale element $0 \cdot P$ ($= \infty$ for elliptiske kurver) i starten af listen, og herefter lægges P til igen og igen (baby steps) indtil vi har hele listen - så vi benytter hele tiden det forrigt udregnede element: $(i-1)P + P = iP$.
4. Beregn, gem og sorter en liste med elementerne $\{Q, Q+R, Q+2R, Q+3R, ..., Q+(b-1)R\}$. Metoden er ligesom før: Start med Q, og læg R til igen og igen (giant steps)

[6] Ved elliptiske kurver kan man f.eks. sortere via x-koordinaterne

5. Gennemløb listerne for at lede efter efter to ens.
Hvis vi i 5. skridt finder et match, så har vi i og j så $iP = Q + jR$. Da $Q = kP$ og $R = b(-P)$ kan vi omskrive til $iP = kP + jb(-P) = (k - jb)P$. Dvs vi har $k = i + jb$ (hvor $0 \leq i, j < b$), som vi returnere i skridt 6.
Hvis vi i 5. skridt ikke finder et match er det fordi N ikke er stor nok. Der vil altid være et match hvis N er mindst ordnen af gruppen: Vi kan nemlig skrive $Q = kP = (k_1 b + k_0)P$ for nogle heltal k_0, k_1 der opfylder $0 \leq k_0, k_1 < b$ (det kan vi fordi $k \in \{0, 1, ..., N-1\}$ (vi antage at k er mindre end ordnen af gruppen)). Men det medfører at $k_0 P = Q - k_1 bP = Q + k_1 R$. Og $k_0 P$ og $Q + k_1 R$ findes netop i første henholdsvis anden liste (så der vil altså være et match)

6. Returner $i + jb$ (kun hvis der fandtes et match. Ellers skal der laves et nyt gæt og startes forfra - man kan her beholde de to lister og arbejde videre med dem, så man ikke starter forfra igen)

I 2. skridt skal man finde den inverse til P. I $E(\mathbb{F}_q)$ er dette nemt - elementet skal blot spejles i x-aksen. Ved generelle grupper kan det måske være et problem. Hvis ikke man kan klare dette på $O(b)$ tid ryger idéen med algoritmen i vasken. Men hvis det lykkedes kræver alle skridt i algoritmen højst et antal udregninger der er proportional til $b = \lceil \sqrt{N} \rceil$.

3.2 Pollard's ρ-metode

Pollards ρ-metode er *probabilistisk*, hvilket vil sige, at der er meget høj sandsynlighed for at den giver et resultat på den krævede tid ($O(\sqrt{N})$). Baby- og giantstep-metoden er derimod *deterministisk*, så man er garanteret på at få et resultat på $O(\sqrt{N})$ tid.

Idéen med denne metode er, at 'spring' rundt fra element til element (vha en funktion der opfører sig 'tilfældigt'), og hvert nyt element skal kunne udtrykkes som $aP + bQ$ for nogle $a, b \in \mathbb{N}$. Hvis fremover N er ordnen af G, skal der højst N spring til, før vi har en kollision med et tidligere element, som vi kan opdage, hvis vi har gemt alt indtil dette punkt. Vi har så to udtryk for det samme element og: $aP + bQ = cP + dQ$. Dvs $(a + bk)P = (c + dk)P \Rightarrow a + bk \equiv c + dk \pmod{N}$ hvorefter k findes (mere om det senere).
Det lyder som om ovenstående tager både $O(N)$ tid og plads, men rent faktisk tage det kun $O(\sqrt{N})$ tid og plads (probabiblistisk). Derudover kan den krævede plads komme helt ned på $O(1)$ ved at benytte en metode udviklet af R. W. Floyd, men lad os nu først beskrive metoden i dybden:

Vi skal bruge en funktion f, der opfører sig tilfældigt over G - Vi har ikke tid eller plads til på forhånd at lave N terningekast for at finde ud af hvad f skal sende hvert $x \in G$ over i. Vi gør det næstbedste: definerer f vha en gaffelfunktion som f.eks. denne:

$$f(x) := \begin{cases} x + M_1 & \text{hvis } x \in S_1 \\ x + M_2 & \text{hvis } x \in S_2 \\ \quad ... & \quad ... \\ x + M_s & \text{hvis } x \in S_s \end{cases}$$

hvor $S_1, S_2, ..., S_s$ er en (nogenlunde ligelig) disjunkt opdeling af G, og $M_i := a_i P + b_i Q$ for nogle tilfældigt valgte a_i og b_i (som vi gemmer til senere). Intuitivt kan man se, at jo større s er, jo mere tilfældig vil f se ud over G. Et godt valg af s viser sig at være omkring 20 ([7]). Et større s er stort set bare spild af plads da vi jo skal holde $a_1, a_2, ..., a_s, b_1, b_2, ..., b_s$ gemt under hele algoritmen, samt nemt kunne finde ud af hvilken af S_i'erne et element er i. Man kan også have forgreninger som f.eks. $x + x$ eller andre udtryk, der ser tilfældige ud - Men vi skal efter springet kunne udtrykke $f(x)$ som $uP + vQ$ for nogle $u, v \in \mathbb{N}$. Det lader sig gøre i den ovenstående definerede f, fordi vi fra starten ved at $x = u'P + v'Q$ for nogle $u', v' \in \mathbb{N}$; og så er $f(x) = u'P + v'Q + M_i = u'P + v'Q + a_i P + b_i Q = (u' + a_i)P + (v' + b_i)Q$ for det i hvor $x \in S_i$.

Vi starter et sted, f.eks. ved ∞ eller et tilfældigt $P_0 \in G$ som vi kan udtrykke $P_0 = \alpha_0 P + \beta_0 Q$. Og så udfører vi spring med f: $P_1 = f(P_0), P_2 = f(P_1), P_3 = f(P_2), ...$ Da der kun er N forskellige elementer i G vil vi på et tidspunkt springe tilbage til et tidligere element, så for nogle $i_0, j_0 \in \mathbb{N}$ vil $P_{i_0} = P_{j_0}$.
Det forventes at $i_0 < j_0 = O(\sqrt{N})$. Dette er sammenligneligt med fødselsdagsparadoxet: Hvis der er 23 mennesker i et rum, er der over 50% sandsynlighed for at to har fødselsdag samme dag. I vores tilfælde er N meget større end 365, og så forventes det at j_0 faktisk er $O(\sqrt{N})$. Fødselsdagsparadoxet eller dens generalisation vil ikke blive beskrevet nærmere her.

Men hvis ikke vi vil gemme data om hvert eneste element vi har besøger indtil vi finder et match, skal vi gøre noget andet. Vi benytter fortsat at $i_0 < j_0$ er de mindste tal hvor $P_{i_0} = P_{j_0}$. Der gælder for alle $i \geq i_0$ (vi skriver

[7] Se Lawrence C. Washington: Elliptic curves, Number Theory and Cryptography, second edition, s. 148

$i = i_0 + l$) at:

$$P_{i_0+l} = (\underbrace{f \circ f \circ ... \circ f}_{l \text{ gange}})(P_{i_0}) = (\underbrace{f \circ f \circ ... \circ f}_{l \text{ gange}})(P_{j_0}) = P_{j_0+l}$$
$$\Rightarrow P_i = P_{i_0+l} = P_{j_0+l} = P_{(i_0+l)+(j_0-i_0)} = P_{i+(j_0-i_0)} \quad \text{for } i \geq i_0$$
$$\Rightarrow P_i = P_{i+n(j_0-i_0)} \quad \text{for alle } i \geq i_0 \text{ og } n \in \mathbb{N}$$

(hvor sidste udtryk følger af at benytte udtrykket i midten flere gange)
Så vi går faktisk i ring herfra. Floyd's metode udnytter dette til at gøre algoritmen mindre pladskrævende: Istedet for at gemme data om alle punkter vi støder på, husker vi bare på to elementer hele tiden: P_i og P_{2i} (og hvordan de kan udtrykkes $uP + vQ$), sådan at hvert nyt skridt i algoritmen beregner $P_{i+1} = f(P_i)$ og $P_{2(i+1)} = f(f(P_{2i}))$. Når i er større end i_0 og er et multiplum af $j_0 - i_0$ gælder at $P_{2i} = P_{i+n(j_0-i_0)} = P_i$ og vi har et match! Det første i der opfylder dette er også højst j_0 (fordi der findes et tal mellem i_0 og j_0 som $j_0 - i_0$ går op i).
Så algoritmen kræver (probabilistisk) $O(\sqrt{N})$ tid og kun lidt plads.

Istedet for at benytte Floyd's metode kan man vælge kun at gemme u og v for nogle specifikke elementer $P_i = uP + vQ$, som skiller sig ud (f.eks. at 2^l går op i u og v for et $l \in \mathbb{N}$ (derved behøver man heller ikke gemme de sidste l bits af u og v, da de alligevel er 0)). Ved at gøre dette skal algoritmen højst udføre 2^l ekstra skridt, men tilgengæld reduceret pladskravet med en faktor 2^{-l}. Denne metode gør det også muligt at fordele arbejdet mellem flere processorer vha Pollards λ-metode, som bygger videre på ovenstående ρ-metode. Der er nemlig ikke noget i vejen for at flere processorer kan starte hver deres sted i gruppen, og springe rundt derfra og så sende de specielle elementer til en hovedcomputer, hvis eneste job er at sammenligne alle elementerne den får tilsendt indtil der sker et match - f.eks. mellem $\bar{P} = uP + vQ$ tilsendt fra én computer, og $\bar{P}' = u'P + v'Q$ tilsendt fra en anden computer. Så er $uP + vQ = u'P + v'Q$

Når vi nu har u, v, u' og v' sådan at $uP + vQ = u'P + v'Q$ (uanset hvilken metode vi brugte) så må $(u + vk)P = (u' + v'k)P$ (ved at indsætte $Q = kP$). Dvs, da P er en generator for G af orden N:

$$\begin{aligned} u + vk &\equiv u' + v'k \pmod{N} \\ \Rightarrow u - u' &\equiv (v' - v)k \pmod{N} \end{aligned} \tag{28}$$

Så vi skal finde k så $(v' - v)k + Ny = u - u'$ (for et $y \in \mathbb{N}$). Dette kan vi gøre med Euklids udvidede algoritme hvis den største fælles divisor mellem

$v' - v$ og N er 1. Hvis $d := gcd(v' - v, N)$ er større end 1 må vi istedet finde k' der løser $u - u' \equiv (v' - v)k' \pmod{\frac{N}{d}}$ (her gælder der hvertfald at $gcd(v' - v, \frac{N}{d}) = 1$). Herefter må en af elementerne k', $2k'$, ..., dk' være det k der opfylder (28). Vi tjekker dem en efter en for at finde ud af hvilken en det er. (d er typisk lille, så det tager ikke så lang tid)

4 Program: Diffie-Hellman nøgleudveksling

4.1 Teori

Lad os sige, at vi har to personer, Alice og Bob, som vil finde frem til en hemmelig fælles nøgle, men de har ingen 'hemmelige' kommunikationsveje hvorigennem de kan blive enige om en sådan nøgle, uden at blive overvåget. Diffie-Hellmans nøgleudveksling med elliptiske kurver løser dette problem på følgende måde:

1. Alice og Bob sender information til hinanden (offentligt) indtil de er blevet enige om to ting: En elliptisk kurve E (over et legeme $\mathbb{F}_q$), hvor det er svært at løse diskrete logaritmer, samt et punkt P på denne kurve. Alle kan se disse informationer.

2. Herefter finder de hver især på et hemmeligt tal, f.eks. finder Alice på a, og Bob på b

3. Alice udregner $P_a = aP$ og Bob $P_b = bP$

4. Alice sende P_a til Bob, og Bob sender P_b til Alice

5. Nu kan Alice udregne $abP = aP_b$, ligesom Bob kan det ved $abP = bP_a$

Både Alice og Bob kender nu abP, hvor eventuelle lyttere kender P_a, P_b, P samt E. Men udfra disse data er det svært at regne sig frem til abP. De metoder (som vi kender), der kan klare det, benytter sig af at man kan beregne den diskrete logaritme. Index Calculus kan ikke bruges, og de andre metoder beskrevet i sektion 3 tager alt for lang tid hvis Alice og Bob har valgt en ordentlig elliptisk kurve.

Derfor har Alice og Bob nu hemmelig viden, abP, som ingen andre kender. Herudfra kan de lave deres nøgle. Det vedlagte program udfører diffie-Hellman nøgleudveksling. Der benyttes ikke projektive koordinater til at repræsentere punkter, hvorfor hver addition kræver en inversion. Dette er den mere teoretiske (og lidt mindre effektive) måde at regne på.

4.2 Klasser

Programmet benytter sig af fire klasser:

Natural

Klassen Natural repræsenterer et tal fra $\mathbb{N}_0$ som en (2^{32})-adic expansion. Til dette benyttes et array af unsigned long ints, som hver kan repræsentere et tal mellem 0 og $2^{32}-1$. Arrayet vokser, hvis tallet bliver større. Til at holde styr på arrayets længde benyttes en unsignet short int, der kan tage værdier mellem 0 og $2^{16}-1$. (så selve datastrukturen tillader tal op til $(2^{32})^{2^{16}}-1$, selvom funktionerne til denne klasse ikke er afprøvet eller testet med sådanne astronomiske tal). Til denne klasse findes der metoder til addition, subtraktion, multiplikation, heltalsdivision og modulus m.m. med andre Naturals (og alm. integers). Bl.a. findes også en metode gcd til at finde største fælles divisor.

Whole

Whole representerer et tal fra $\mathbb{Z}$. Det er en udvidelse af Natural, og indeholder også en boolean til at bestemme fortegn. Hertil findes også metoder til addition, subtraktion osv. Disse metoder benytter sig af Naturals metoder og behandler selv fortegnet. Der findes desuden en funktion euclid(), der tager to Wholes x og y. Metoden finder løsningen til gcd(x,y) = ax+by; sætter x=a, y=b og returnerer gcd.

Resid

Resid repræsenterer en residueklasse. Det er en udvidelse af Natural, og indeholder også en pointer til en anden Natural, som er modulus til residueklassen. Der findes en inverse-funktion, der benytter sig af euklid-metoden til at give et svar. Derudover findes en funktion til at opløfte til en potens (hertil benyttes kvadrering en masse gange)

ECP

Klassen repræsenterer et punkt på en elliptisk kure (elliptic curve point). Hvert punkt har pointers til de Resids A og B som beskriver kurvens punkter ved $y^2 = x^3 + Ax + B$. Derudover har den to Resids x og y (selve punktet) og en boolean til at bestemme om punktet findes ved uendelig. Den udfører addition hvortil der skal beregnes en invers. Derudover findes en metode til at multiplicere et punkt med et tal.

4.3 Opsætning

Programmet kan findes på den vedlagte CD samt på `http://n144.dk`. Forhåbentlig kan programmet startes 'out of the box'. Hvis ikke kan det være nødvendigt at hente Microsoft Visual C++ 2010 Redistributables:
`http://www.microsoft.com/download/en/details.aspx?id=5555`

4.4 Eksempel på anvendelse

1. Alice finder et primtal q og en elliptisk kurve $E_{A,B}$ over $\mathbb{F}_q^*$ bestående af alle (x, y) fra $\mathbb{F}_q^* \times \mathbb{F}_q^*$ hvor $y^2 = x^3 + A \cdot x + B$. Derudover finder hun et punkt P herpå. Dette kan hun evt. gøre i samarbejde med Bob. Dataene indtastes. (For at teste programmet kan man vælge et lille primtal og elliptisk kurve ved at klikke 'Brug test-værdier')

2. Alice klikker 'Generér a' for at vælge et tilfældigt a. Alternativt kan hun indtaste et selv og vælge Værktøjer>'Opdater' (Herefter beregnes Pa)

3. Alice vælger Filer>'Gem offentlig data', og gemmer i en fil kaldet "Offentlig data til Bob.public", denne fil sender hun til Bob over netværket.

4. (Valgfri) Hvis det tager lang tid før Alice får svar, kan hun gemme sin hemmelige nøgle a ved at vælge Filer>'Gem hemmelig nøgle a', og gemme i en fil kaldet "Alices nøgle.secret". Filen skal aldrig sendes over netværket. Vælger Alice ikke at gemme den hemmelige nøgle skal hun have programmet åbent indtil Bob svarer (eller huske den hemmelige nøgle i hovedet)

5. Bob vælger Filer>'Åbn offentlig data', og vælger 'Offentlig data til Bob.public', som han har fået af Alice. Informationer om q, den elliptiske kurve og punktet herpå indlæses, og punktet der for Allice hed Pa, hedder nu for Bob Pb. (Generelt er b modpartens (ukendte) nøgle, hvor a er ens egen)

6. Bob vælger et hemmelig tal ligesom Alice gjorde i 2)

7. Bob kan nu se den fælles hemmelige nøgle Pab og kan gemme denne (lokalt) via Filer>'Gem fælles nøgle Pab'. Men Alice kan stadig ikke udregne nøglen, så Bob skal vælge filer>'Gem alt offentlig data', og gemme i en fil kaldet "Offentlig data til Alice.public", så Alice kan få Pa (hvor a er Bobs hemmelige nøgle). Denne fil sendes til Alice.

8. Alice åbner denne fil via Filer>'Åbn offentlig data'.

9. (Evt) Hvis Alice har haft programmet lukket undervejs skal hun nu indlæse sit hemmelige tal a, som hun har gemt lokalt i 4). Dette foregår via Filer>'Åbn hemmelig nøgle a'. Ellers kan hun indtaste tallet fra hukommelsen og trykke Værktøjer>opdater. Hvis Alice slet ikke har haft programmet lukket undervejs skal hun ikke foretage sig noget.

10. Alice kan nu også se den fælles hemmelige nøgle Pab

4.5 Kendte problemer

(eller blot ting at være opmærksom på)

- Programmet kan ikke selv generere store primtal eller elliptiske kurver med punkter af store ordner, som skal bruges til nøgleudvekslingen. Algoritmerne hertil kræver at man kan primfaktorisere store tal, og at inkludere dette i programmet ville være for stort et projekt. Så hertil skal man bruge andre programmer (eller vælge test-værdierne).

- Da programmet ikke kender ordnen af punktet P kan den ikke generere et (jævnt fordelt) tilfældigt tal a mellem 0 og $ord(P)$. Istedet vælges a som et tilfældigt tal mellem 0 og q. Bemærk at ordnen af P deler ordnen af kurven, og ordnen af kurven højst er $2\sqrt{q}$ fra $(q+1)$ pr. Hasse's theorem.

- Indtil andet er givet, antages at Pb og a er uendelige.

- Alt input foregår i HEX, dvs med 0-9 og A-F

5 Andre anvendelser i kryptering

5.1 ElGamal Digital signatur

Bob vil gerne være sikker på at et digitalt dokument er underskrevet af Alice. Det skal ikke være muligt for Eve at kopiere underskriften og lægge den til andre dokumenter. Man kan sikre dette på nedenstående metode. Den underskrevne besked vil blive synlig for alle, men underskriften kan ikke forfalskes, og det underskrevne dokument kan ikke ændres uden at det kan opdages.

Alice vælger først en offentlig nøgle, nemlig sættet (E, q, f, P, Q) hvor E er en elliptisk kurve over $\mathbb{F}_q$. $f : E \to \mathbb{Z}$ er en funktion, som har en stor værdimængde, og hvor kun få input sendes over i hvert output. Derudovere er $P \in E$ et punkt med stor orden N (helst stor primorden), og $Q = aP$ hvor a er et tal, Alice holder hemmeligt. For at underskrive en besked m gør hun følgende:

1. Alice bruger om nødvendigt en offentlig kendt kryptografisk hash-funktion H på beskeden, hun vil underskrive, til at generere et tal $H(m)$, der er mindre end N.

2. Alice vælger et tilfældigt og hemmeligt k med $\gcd(k, N) = 1$ (så hun i næste skriodt kan finde k^{-1}) og beregner $R = kP$

3. Beregner $s \equiv k^{-1}(H(m) - af(R)) \pmod{N}$

Den underskrevne besked er nu (m, R, s). For at verificere underskriften skal Bob downloade Alices offentlige nøgle og teste om $f(R)Q + sR = H(m)P$. I så fald konkluderer han at underskriften er gyldig. Vi har nemlig at:

$$sR = \left(k^{-1}(H(m) - af(R))\right)(kP) = H(m)P - af(R)P \tag{29}$$

(her er det vigtigt at s var ækvivalent modulo N, ellers vil $NR \neq \infty$, og første lighed vil ikke gælde). Vi kan så se at:

$$f(R)Q + sR = f(R)aP + H(m)P - af(R)P = H(m)P$$

Første lighed i (29) gælder netop når afsenderen er Alice, da hun jo kender a. Lad os nu sige at Eve vil prøve at bruge Alice's underskrift på et dokument m. Hun skal finde på s og R så $f(R)Q + sR = H(m)P$. Hvis hun først vælger R, skal hun finde s ved at løse $sR = H(m)P - f(R)Q$, hvilket kræver at hun kan løse diskrete logaritmer. Hvis hun istedet vælger s først, skal hun løse $f(R)Q + sR = H(m)P$ for R, hvilket ser mindst lige så svært ud (selvom problemet ikke er analyseret i samme grad). En anden bemærkning er, at det kan være der findes en metode, der skaber et sæt (s, R) der passer til beskeden m. Men en sådan (effektiv) metode er altså ikke opdaget endnu (hvis den overhovedet findes).

Det er meget vigtigt at Alice ikke genbruger det samme tal k til to forskellige beskeder, for så kan Eve i sidste ende udregne a; Først og fremmest kan Eve med det samme spotte om det er brugt samme k til to forskellige meddelelser, for så vil R også være ens for de to meddelelser. Derfra kan Eve finde a ved at behandle de to sæt (m_1, R, s_1) og (m_2, R, s_2):

1. Eve kan opskrive:

$$\begin{aligned} s_1 k &\equiv H(m_1) - af(R) \pmod{N} \\ s_2 k &\equiv H(m_2) - af(R) \pmod{N} \end{aligned}$$

   Subtrahér disse, og vi får:

$$(s_2 - s_1)k \equiv H(m_2) - H(m_1) \pmod{N} \tag{30}$$

2. Eve vil så løse $k(s_2 - s_1) + nN = H(m_2) - H(m_1)$ for k (hvor $n \in \mathbb{Z}$). Hun kan hverfald løse $u(s_2 - s_1) + vN = gcd(s_2 - s_1, N)$ via Euklids udvidede algoritme. Og så kan hun nemt finde k og n da (vi definerer

her $d := gcd(s_1 - s_1, N)$ og $h = H(m_2) - H(m_1)$):

$$k(s_2 - s_1) + nN = d\frac{k(s_2 - s_1) + nN}{d} = (u(s_2 - s_1) + vN)\frac{h}{d}$$
$$= \left(\frac{uh}{d}\right)(s_2 - s_1) + \left(\frac{vh}{d}\right)N$$

(obs: d går op i $k(s_2 - s_1) + nN$). Så nu har Eve én løsning til (30).

3. Eve finder de andre ved at lægge $\frac{iN}{d}(s_2 - s_1) = \frac{i(s_2 - s_1)}{d}N$ til på venstre side i (30) (for $i = 0, 1, 2..., d - 1$). Alle disse tal er multiplum af N, så det ændrer ikke højresiden i (30). Vi ser at alle koefficienter $\frac{iN}{d}$ er forskellige, og mindre end N, så vi finder d forskellige løsninger til (30) (der er heller ikke flere). Ét af disse k'er vil opfylde $P = kQ$. Alice prøver dem alle igennem til hun finder det rigtige k (skridt 2 og 3 er meget nemmere hvis N er et primtal - så er det blot euklids algoritme)

4. Eve kan nu skrive $ks \equiv m - af(R) \pmod N$ hvor a er den eneste ubekendte. Hun går altså bare igang med at løse $ks + nN = m - af(R)$, eller:
$$af(R) + nN = m - ks$$
Dette problem er det samme som i 2. og 3. skridt. Hvis N er et primtal er det nemt. Hvis ikke får Eve d forskellige a'er. Hun prøver dem alle indtil hun finder et hvor $Q = aP$

5. Eve kender nu Alices hemmelige nøgle a og kan bruge hendes signatur på elektroniske dokumenter.

En vigtig note omkring den kryptografiske hash-funktionen H er, at givet y skal det være svært at kunne lave en ny meddelelse m, så $H(m) = y$. Ellers ville Eve kunne ændre det underskrevne dokument, uden at det kan opdages. Ligeledes skal det også være svært når man er givet m_1, at finde et m_2 så $H(m_1) = H(m_2)$.

5.2 ECIES

ECIES (Elliptic curve integrated encryption scheme) er en metode til at sende krypterede beskeder til en modtager. Der benyttes en symmetrisk enkryption, ENC_k så nøglen k hertil skal være svær (under DL) for tredje personer at finde, men det skal stadig være nemt for Bob. Man kunne alene benytte Diffie-Hellmans nøgleudveksling til at etablere den fælles nøgle mellem Alice og Bob, men for at sikre sig mod chosen ciphertext-attacks, kan man bruge

ECIES der benytter den fælles nøgle P_{ab} anderledes: Udover ENC_k bruger ECIES to funktioner - en til udledning af nøgler (Key Derivation Function, KDF), og en til at afgøre om den krypterede besked er gyldig (Message Authentication Code, MAC). KDF tager et punkt, som kun Alice og Bob kender (P_{ab}) og giver to nøgler, k_1 og k_2. MAC tager en nøgle (k_2) og en krypteret meddelelse (C) og returnere et tal, der kan bruges til at tjekke om den krypterede meddelelse (C) er gyldig (dvs. at den ikke er ændret undervejs). Disse antages i det følgende til at være (kryptografiske) hash-funktioner.

Først etablerer Alice og Bob den fælles nøgle P_{ab} med Diffie-Hellmans nøgleudveksling.

Alice vælger en elliptisk kurve $E_{A,B}$ over $\mathbb{F}_q$, samt et punkt $P \in E_{A,B}$ herpå med stor orden N (evt primtal). Hun vælger så en hemmelig nøgle, a hvor $1 \leq a < N$, og beregner $P_a = Pa$. Den offentlige nøgle er sættet $(q, E_{A,B}, N, P, P_a)$

Alice skal offentlig disse data for at Bob kan kryptere sin meddelelse m så kun Alice (og Bob selv) kan dekryptere den.

Alice, der vil afsende m gør nu følgende:

1. Beregner k_1 og k_2 med $\text{KDF}(P_{ab})$ (f.eks. opfatter resultatet af KDF som k_1 efterfulgt af k_2)
2. Beregner $C = \text{ENC}_{k_1}(m)$ og $t = \text{MAC}(C, k_2)$
3. Sender C og t til Bob.

Bobs tur:

1. Han beregner k_1 og k_2 med $\text{KDF}(P_{ab})$ ligesom Alice gjorde det.
2. Tester om $t = \text{MAC}(C, k_2)$. Hvis ikke afvises C
3. Dekrypterer C til m ved at bruge ENC_{k_1}'s tilsvarende dekrypteringsfunktion.

Kun når man kender P_{ab} kan man finde k_1 og k_2, så Eve kan ikke dekryptere C. Hvis ikke Bob ikke udfører andet skridt, hvor han kan afvise C, har Eve mulighed for at ændre C undervejs uden at det vil blive opdaget. Med verificeringsskridtet er Eve nød til også at ændre t således at $t = \text{MAC}(C, k_2)$ er opfyldt efter hun har ændret C. Men den eneste måde at finde k_2 er ved $\text{KDF}(P_{ab})$.

De kryptografiske hash-funktioner KDF {og MAC} skal nemlig (ligesom i sektion 5.1) opfylde at givet k_2 $\{t\}$ er det svært at finde $Q \in E_{A,B}$ $\{k_2\}$ der opfylder $k_2 = \text{KDF}(Q)$ $\{t = \text{MAC}(C, k_2)\}$

6 Bibliografi

- Victor Shoup: A Computational Introduction to Number Theory and Algebra, Version 2
- Lawrence C. Washington: Elliptic Curves, Number Theory and Cryptography, Second Edition, Chapman & Hall/CRC
- Richard Crandall, Carl Pomerance: Prime Numbers, A Computational Perspective, Second Edition, Springer
- Prof. Jean-Jacques Quisquater, Dr. François Koeune, ECIES, Security Evaluation of the Encryption Scheme and Primitives, July 2002