Aufgabe: Beim Giant-Step-Baby-Step Algorithmus vom letzten Mal entstand folgende Liste:

$$\mathcal{L}_1 = \{(0,1), (1,175), (2,207), (3,132), (4,173), (5,84), (6,172), (7,136), (8,192), (9,4), (10,19), (11,147), (12,74), (13,11), (14,109), (15,7)\}$$

Man trage diese Elemente in eine Hashtabelle mit 23 Elementen ein und verwende die Hashfunktion $h(x) = x \bmod 23$. Bei Auftreten einer Kollision verwende man die Methode Lineares Sondieren. Die Hashtabelle soll so organisiert sein, dass man effizient nach den 2. Komponenten der Zahlenpaare suchen kann und bei einem Treffer die 1. Komponente zurückgeliefert bekommt.

Antwort: Man berechnet nacheinander folgende Hashwerte für die 2. Komponenten und trägt die Elemente in die Hashtabelle $a[0..22]$ ein. Die dazugehörigen 1. Komponenten kommen in das Array $b[0..22]$.

$h(1) = 1$, $h(175) = 14$, $h(207) = 0$, $h(132) = 17$,
$h(173) = 12$, $h(84) = 15$, $h(172) = 11$, $h(136) = 21$,
$h(192) = 8$, $h(4) = 4$, $h(19) = 19$, $h(147) = 9$,
$h(74) = 5$, $h(11) = 11^{*)}$, $h(109) = 17^{*)}$, $h(7) = 7$.

Bei den mit *) gekennzeichneten Fällen tritt eine Kollision auf. Man speichert die entsprechenden Zahlen eine (oder 2 oder 3...) Position hinter dem gehashten Arrayelement (das bereits belegt ist).

	0	1	2	3	4	5	6	7
a	207	1			4	74		7
b	2	0			9	12		15

	8	9	10	11	12	13	14	15
a	192	147		172	173	11	175	84
b	8	11		6	4	13	1	5

	16	17	18	19	20	21	22
a		132	109	19		136	
b		3	14	10		7	

Noch einige Bemerkungen zu Hashing

Bisher: $h : \mathcal{M} \longrightarrow \{0,1\}^l$

$\mathcal{M}$ ... Menge der Nachrichten

l ... relativ kleine Konstante, z.B. $l = 160 = \underbrace{2 \cdot 80}$ wegen Geburtstagsangriff

$h(m)$ ist eine Art „Fingerabdruck" der Nachricht m (MAC $\hat{=}$ message authentication code).

Manche Anwendungen benötigen zusätzlich noch einen Schlüssel $k \in \mathcal{K}$. Damit ist

$$h : \mathcal{M} \times \mathcal{K} \longrightarrow \{0,1\}^l$$

Der Schlüssel k wird zufällig gezogen und zunächst geheim gehalten. In einer späteren Runde des Protokolls wird dann typischerweise k mitgeteilt.

In der Algorithmik nennt man die Situation, dass die eigentliche Hashfunktion erst durch

einen zufälligen Schlüssel k spezifiziert wird, <u>Universales Hashing</u>. In anderen Worten, es wird per Zufall eine Hashfunktion $h : \mathcal{U} \to \{0,1\}^q$ aus einer Klasse $\mathcal{H}$ von Hashfunktionen ausgewählt: $\mathcal{H} = \{ h(k,\cdot) \mid k \in \mathcal{K} \}$

Eine besonders interessante Hashfunktion mit Schlüssel ist Folgende:

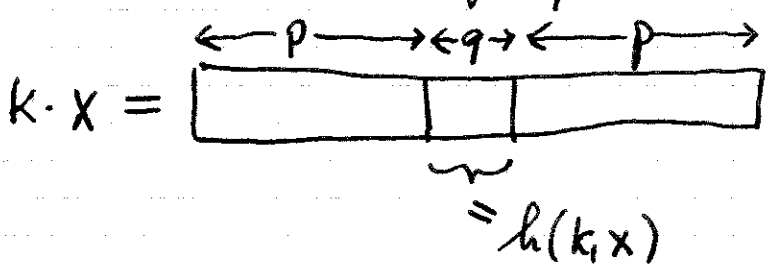
$$h : \mathcal{U} \times \mathcal{K} \to \{0,1\}^q$$

wobei $\mathcal{U} = \{0,1\}^p$, $p > q$. $\mathcal{K} = \{0,1\}^{p+q}$

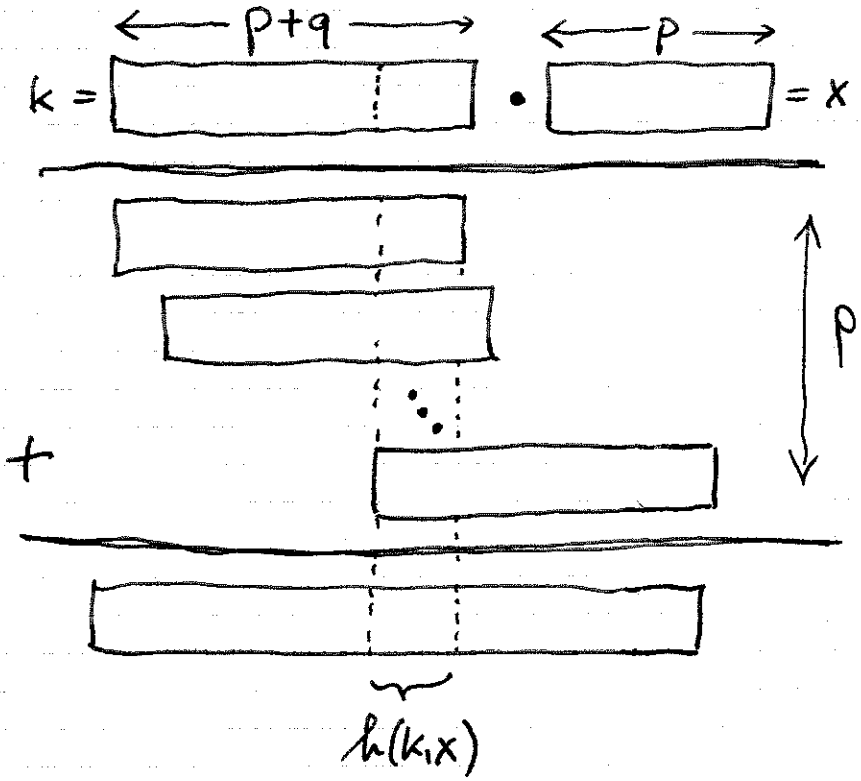
$$h(k,x) = (k \cdot x \bmod 2^{p+q}) \operatorname{div} 2^q$$

k, x: Bitstrings der Länge $p+q$ bzw. p

Das bedeutet inhaltlich Folgendes: Das Produkt $k \cdot x$ ist ein $(2p+q)$-Bit langer String. Daraus wird ein mittlerer Abschnitt der Länge q herausgeschnitten:



Nochmals dargestellt im Sinne der „Schulmultiplikation":



Aufgabe: Sei p Primzahl und a Primitivwurzel mod p. Warum ist $h: \mathbb{N} \to \{1, \ldots, p-1\}$

mit $h(x) = a^x \bmod p$

keine gute Hashfunktion (für kryptographische Zwecke)?

Antwort: Für Kollisionsresistenz sollte man kein $x \neq x'$ ermitteln können mit $h(x) = h(x')$.

Es gilt:

$$\begin{aligned} & h(x) = h(x') \\ \Leftrightarrow \quad & a^x \bmod p = a^{x'} \bmod p \\ \Leftrightarrow \quad & a^x \equiv a^{x'} \pmod p \\ \Leftrightarrow \quad & x \equiv x' \pmod{p-1} \end{aligned}$$

Somit liegen entsprechende x, x' immer im Abstand $p-1$ auseinander und sind somit leicht zu ermitteln.

Aufgabe: In einer Familie von 4 Personen, wie groß ist die Wahrscheinlichkeit, dass keine 2 Personen der Familie im selben Monat Geburtstag haben?

Antwort: Die gesuchte W'heit p ist

$p = P($2. Person im anderen Monat Geb.tag als 1. Person$)$

$\cdot P($3. Person im anderen Monat Geb.tag als 1. +2. Person$)$

$\cdot P($4. Person im anderen Monat Geb.tag als 1., 2. und 3. Person$)$

$$= \frac{11}{12} \cdot \frac{10}{12} \cdot \frac{9}{12} = \frac{55}{96} \approx 0{,}573$$

Frage: Jeder Mensch ist an einem von 7 möglichen Wochentagen geboren (also $m=7$). Die Zufallsvariable X kann die Werte $2, 3, \ldots, 8$ annehmen und ist so definiert, dass $X=k$ gilt, wenn bei k zufälligen Personen mit den Geburtstagswochentagen $t_1, t_2, \ldots, t_k$ die ersten $k-1$ voneinander verschieden sind und t_k schließlich mit einem von $t_1, \ldots, t_{k-1}$ übereinstimmt. Mit anderen Worten, X ist die kleinste Anzahl, bei der zum ersten Mal eine „Dublette" auftritt.
Gib die Wahrscheinlichkeiten $P(X=2), P(X=3), \ldots, P(X=8)$ an und berechne den Erwartungswert

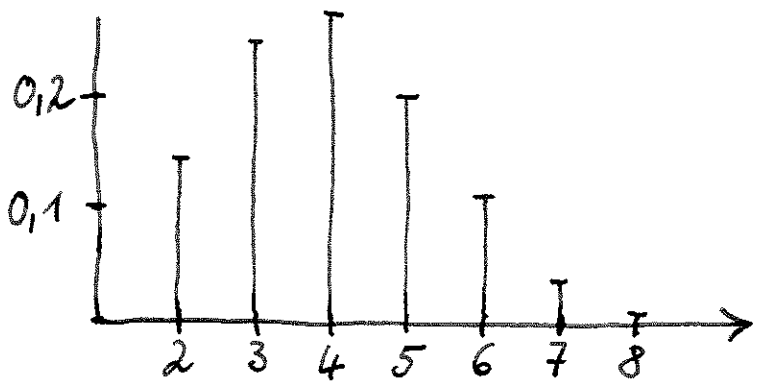
$$E(X) = \sum_{k=2}^{8} k \cdot P(X=k)$$

Antwort: $P(X=2) = \frac{1}{7}$, $P(X=3) = \frac{6}{7} \cdot \frac{2}{7}$,

$P(X=4) = \frac{6}{7} \cdot \frac{5}{7} \cdot \frac{3}{7}$, $P(X=5) = \frac{6}{7} \cdot \frac{5}{7} \cdot \frac{4}{7} \cdot \frac{4}{7}$,

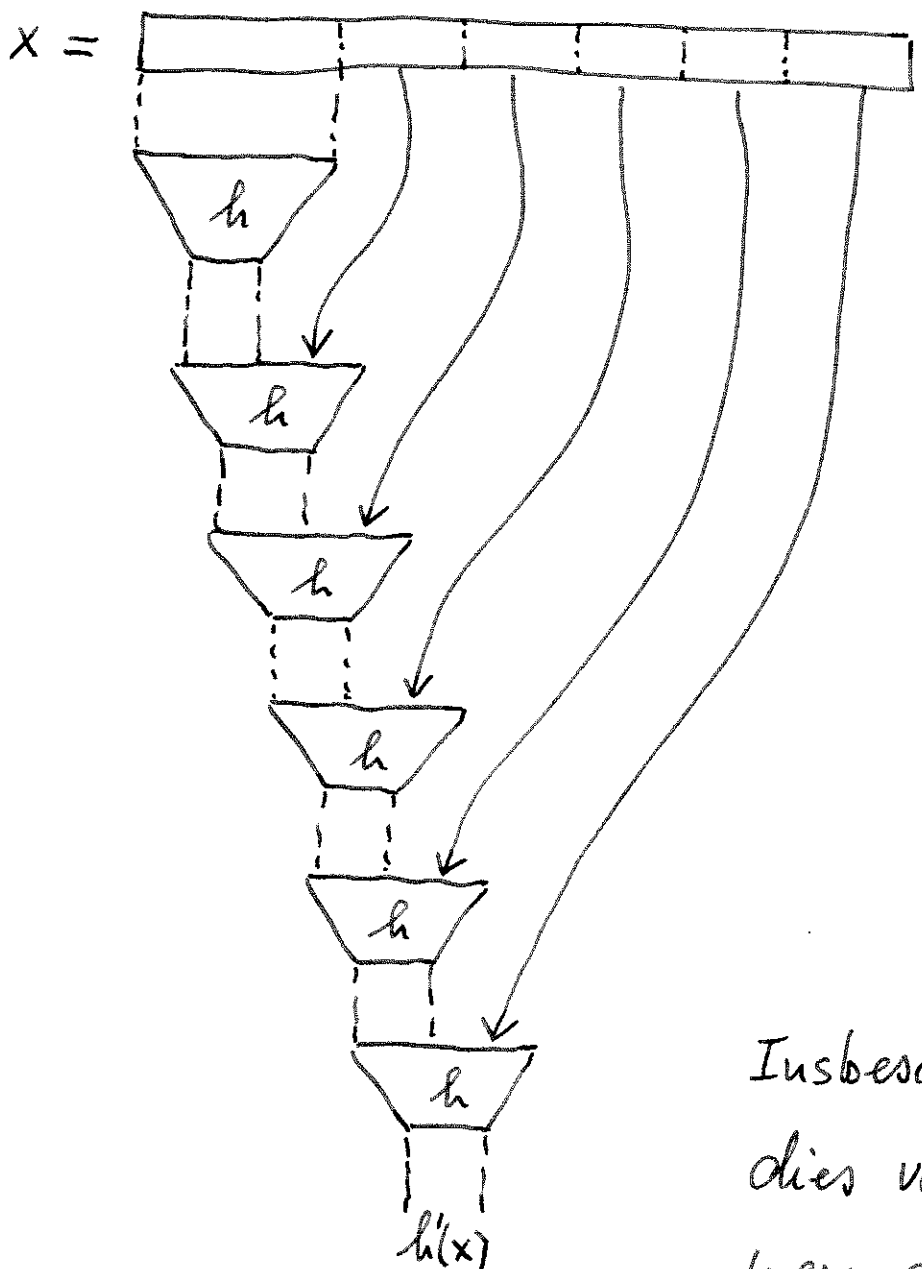
$P(X=6) = \frac{6}{7} \cdot \frac{5}{7} \cdot \frac{4}{7} \cdot \frac{3}{7} \cdot \frac{5}{7}$, $P(X=7) = \frac{6}{7} \cdot \frac{5}{7} \cdot \frac{4}{7} \cdot \frac{3}{7} \cdot \frac{2}{7} \cdot \frac{6}{7}$,

$P(X=8) = \frac{6}{7} \cdot \frac{5}{7} \cdot \frac{4}{7} \cdot \frac{3}{7} \cdot \frac{2}{7} \cdot \frac{1}{7}$



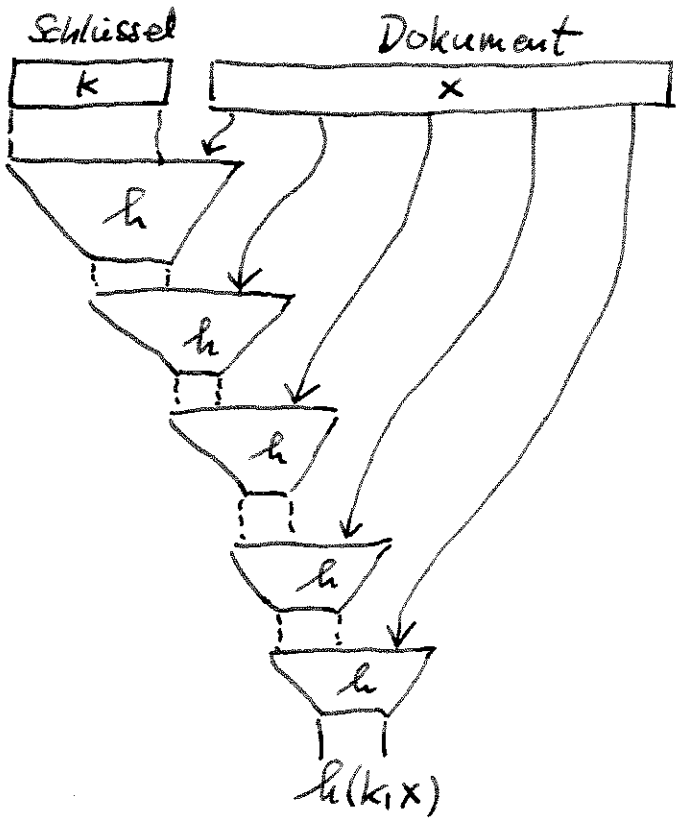
$E(X) = 4,02$

Anmerkungen zu Hashfunktionen: Meist werden Hashfunktion definiert in der Form $h: \{0,1\}^p \to \{0,1\}^q$ wobei $p > q$. Um daraus eine anwendbare Hashfunktion $h': \{0,1\}^* \to \{0,1\}^q$ zu machen, wird die Funktion h iterativ angewandt:

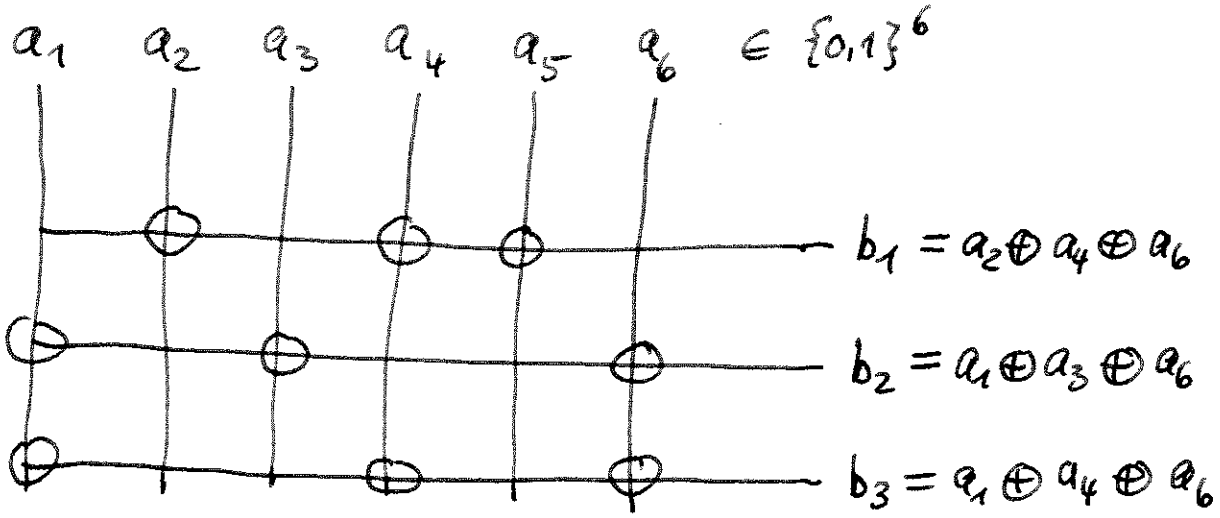


Insbesondere ist dies von Vorteil, wenn die Bits von x iterativ nacheinander entstehen. Diese Vorgehensweise kann „online" angewandt werden.

Diese Methode kann auch verwendet werden, um eine Hashfunktion mit Schlüssel k zu realisieren:



Aufgabe: Das Folgende ist eine Hashfunktion h:
$\{0,1\}^6 \to \{0,1\}^3$:



$\oplus$ bedeutet: 

Wenn man dieses Beispiel auf genügend große $p > q$, also $h: \{0,1\}^p \to \{0,1\}^q$, verallgemeinert: Wie steht es mit der Kollisionsresistenz?

Antwort: Will man weitere x mit $h(x) = y$ finden, so muss man ein lineares Gleichungssystem über $GF(2)$ lösen, was effizient (polynomial in p, q) möglich ist.

Will man die Konstruktion kollisionsresistent(er) machen, so sollte man einige $\oplus$-Operationen durch nicht-lineare wie $\wedge$ bzw $\vee$ ersetzen.

$$GF(2) = (\{0,1\}, \oplus, \wedge)$$

Der probabilistische ρ-Algorithmus von Pollard hat wie Babystep-Giantstep ebenfalls Laufzeit $O(\sqrt{n}) = O(2^{m/2})$, benötigt aber wesentlich weniger Speicherplatz.

Die Aufgabe: den diskreten Log. berechnen

Gegeben: n (Primzahl), a (Prim. Wurzel) und $y < n$.

Gesucht: x so dass $y = a^x \bmod n$

Das Ziel ist, zwei verschiedene Darstellungen

$$z = a^s \cdot y^t$$

$$z' = a^{s'} \cdot y^{t'}$$

zu finden, die zueinander kongruent sind:

$$z \equiv z' \pmod{n}$$

Wegen des Geburtstagsproblems gelingt dies mit hoher W'keit, indem man ca. $\sqrt{n}$ viele solche z-Werte ermittelt:

$$z_1 = a^{s_1} y^{t_1}, \quad z_2 = a^{s_2} y^{t_2}, \quad \ldots, \quad z_{\sqrt{n}} = a^{s_{\sqrt{n}}} \cdot y^{s_{\sqrt{n}}}$$

Dann gilt: $a^s \cdot y^t \equiv a^{s'} \cdot y^{t'} \pmod n$

$a^s \cdot a^{xt} \equiv a^{s'} \cdot a^{x \cdot t'} \pmod n$

$a^{s+xt} \equiv a^{s'+xt'} \pmod n$

$s + xt \equiv s' + xt' \pmod{n-1}$

$x \cdot (t - t') \equiv s' - s \pmod{n-1}$

Diese Kongruenzgleichung kann man nach x auflösen! (Vorausgesetzt: $t \neq t'$)

Es soll wieder ein Pseudozufallsgenerator f eingesetzt werden, um sukzessive die Folge aufzubauen:

$$(s_0, t_0) \xrightarrow{f} (s_1, t_1) \xrightarrow{f} (s_2, t_2) \xrightarrow{f} (s_3, t_3) \xrightarrow{f} \cdots$$

ergibt	ergibt	ergibt	ergibt
$z_0 =$	$z_1 =$	$z_2 =$	$z_3 =$
$a^{s_0} \cdot y^{t_0}$	$a^{s_1} \cdot y^{t_1}$	$a^{s_2} y^{t_2}$	$a^{s_3} \cdot y^{t_3}$

Im Gegensatz zum ρ-Algorithmus für das Faktorisieren steht hier kein Pseudozufallsgenerator „von der Stange" zur Verfügung, um der komplexen Situation

der (s_i, t_i) und der sich daraus ergebenden $z_i = a^{s_i} \cdot y^{t_i}$ gerecht zu werden.

Um den „Cycle Detection Trick" anwenden zu können, muss die Funktion $f: (s_i, t_i) \longmapsto (s_{i+1}, t_{i+1})$ so gestaltet sein, dass aus

$$z_i = a^{s_i} \cdot y^{t_i} \equiv z_j = a^{s_j} \cdot y^{t_j} \pmod{n}$$

$\downarrow f$

folgt dass $z_{i+1} = a^{s_{i+1}} \cdot y^{t_{i+1}} \equiv z_{j+1} = a^{s_{j+1}} \cdot y^{t_{j+1}} \pmod{n}$

Ausserdem sollten die z_i-Werte einigermaßen zufällig aussehen, so dass das Geburtstagsphänomen eintritt. Pollard gelingt dieses Kunststück folgendermaßen:

Zerlege $\{1, 2, \ldots, n-1\}$ in 3 etwa gleich-große Mengen, z.B. $M_1 = \{1, \ldots, \lfloor \frac{n}{3} \rfloor\}$

$$M_2 = \{\lfloor \tfrac{n}{3} \rfloor + 1, \ldots, \lfloor \tfrac{2n}{3} \rfloor\}$$

$$M_3 = \{\lfloor \tfrac{2n}{3} \rfloor + 1, \ldots, n-1\}$$

Die gesuchte Funktion f bildet das Paar (s,t) ab auf (s',t') wobei

$$s' = \begin{cases} 2 \cdot s & , \text{falls } z \in M_1 \\ s+1 & , \text{falls } z \in M_2 \\ s & , \text{falls } z \in M_3 \end{cases}$$

$$t' = \begin{cases} 2 \cdot t & , \text{falls } z \in M_1 \\ t & , \text{falls } z \in M_2 \\ t+1 & , \text{falls } z \in M_3 \end{cases}$$

alle Berechnungen können mod $n-1$ reduziert werden.

Hierbei ist $z = a^s \cdot y^t$. Man kann diese Umrechnung $f: (s,t) \mapsto (s',t')$ auch umrechnen in Bezug auf z:

$$z' = \begin{cases} z^2 & , \text{falls } z \in M_1 \\ z \cdot a & , \text{falls } z \in M_2 \\ z \cdot y & , \text{falls } z \in M_3 \end{cases}$$

diese Berechnungen können mod n reduziert werden.

Zahlenbeispiel: $n = 227$; $a = 5$; $y = 86$

Zerlegung: $M_1 = \{1, \ldots, 75\}$, $M_2 = \{76, \ldots, 151\}$,

$M_3 = \{152, \ldots, 226\}$

	i	s_i	t_i	z_i
	0	1	1	203
$i =$	1	1	2	206
	2	1	3	10
	3	2	6	100
	4	3	6	46
	5	6	12	73
	6	12	24	108
	7	13	24	86
	8	26	48	132
$j =$	9	27	48	206
	10	27	49	10

Periode der Länge 8

Nun gilt: $1 + 2 \cdot x \equiv 27 + 48x \pmod{226}$

$-26 \equiv 46x \pmod{226}$

$46x \equiv 200 \pmod{226}$

Es gibt 2 Lösungen: $x = 24$, $\underline{\underline{137}}$.

Der Cycle Detection Trick, der nur die Werte (z_k, z_{2k}) miteinander vergleicht, wird statt bei $(i, j) = (1, 9)$ fündig bei $(k, 2k) = (8, 16)$.