Security

- LV 4121 und 4241 -

Schlüsselmittelmanagement und Zufallszahlen

Kap. 7: Schlüsselmittelmanagement und Zufallszahlen

Teil 1: Erzeugung von Zufallszahlen

- Allgemeine Aspekte und Zielsetzung
- Zufallszahlengeneratoren
- Neumann-Filter

Allgemeine Aspekte

- Im Zusammenhang mit **kryptographischen Schlüsselparameter** (Initialisierungsvektoren, Startwerte etc.) spielt das Erzeugen von Zufallszahlen (möglichst zufällig, hinreichend groß, unvorhersehbar, besondere Eigenschaften uvm.) eine zentrale Rolle.
- So basiert die **Sicherheit** aller kryptographischen Verfahren auf der Schwierigkeit, einen geheimen Schlüsselparameter zu erraten oder anderweitig zu beschaffen.
- Zufallszahlen und deren Nachbehandlung sind daher ein wesentliches kryptographisches Grundelement.
- Da die Erzeugung echter Zufallszahlen nicht unproblematisch ist, werden vielerorts Pseudozufallszahlen verwendet.

Zielsetzung:

- Es sollte sehr schwierig sein, die Output-Bits eines Pseudo-Zufallszahlengenerators oder eines echten Zufallszahlengenerators vorhersagen zu können.
- Selbst wenn eine Folge von n aufeinanderfolgenden Output-Bits a_1, a_2, ..., a_n eines Generators vorliegen, sollte es rechnerisch unmöglich sein, das (n+1)-te Bit a_{n+1} mit einer Wahrscheinlichkeit, die größer als 0.5 ist, vorherzusagen.

Pseudo-Zufallszahlengenerator:

Definition:

- Ein **Pseudozufallszahlengenerator** ist ein Algorithmus, der nach Eingabe von gewissen Initialisierungsdaten (sogenannten **seed numbers**) eine Zufallsfolge deterministisch erzeugt.
- Einen solchen randomisierten Algorithmus, der in Form eines Simulations- oder Rechenprogramms lediglich eine pseudozufällige Bitfolge liefert, nennt man **pseudo random number generator** (**PRNG**).

Der echte Zufallszahlengenerator:

Definition:

A **random bit generator** is a device that is designed to output a sequenz of statistically independent and symmetrically distributed **binary random variables**, i. e., that is designed to be the implementation of a so-called **binary symmetric source (BSS)**.

- Das Wissen der ersten n Bits einer zufälligen Folge liefert keine Information über das n + 1-te Bit.
- Eine gute Zufallsquelle stützt sich auf physikalische Zufallsereignisse wie zum Beispiel thermisches Rauschen oder radioaktiver Zerfall ab.
- Den zugehörigen Prozess nennt man **real random number generator (RRNG)**.

Nachbehandlung echter Zufallsfolgen:

Auch wenn die Zufallszahlen aus einem physikalischen Prozess stammen, muss untersucht werden, ob der zugrunde liegende physikalische Prozess **echt** zufällig ist und im Falle einer **statistisch unabhängigen** Zahlenfolge diese eine symmetrische Verteilung bezüglich der Werte „0“ und „1“ aufweist.

Der Neumann-Filter:

Der Informatikpionier **John von Neumann** schlug 1951 eine sehr effiktive Funktion **f** zur Beseitigung der Asymmetrie in einer Bitfolge vor:

$$\mathbf{f} : \{0,1\}^m \to \{0,1\}^n \text{ mit } 00 \to \varepsilon,\ 11 \to \varepsilon,\ 01 \to 0,\ 10 \to 1,$$

wobei sich **f** auf zwei aufeinanderfolgende Bits (nicht überlappende Bit-Paare) bezieht und ε für die leere Zeichenkette steht.

Eigenschaften nach Anwendung des Neumann-Filters:

- Wenn in einer Bitfolge $\mathbf{a_i \rightarrow \{0,1\}^n}$ aufeinanderfolgende Bits **statistisch unabhängig** sind und den Wert „**1**" mit der Wahrscheinlichkeit **p** annehmen, so verkürzt sich die Länge der Bit-Folge durch die Filterung um den Faktor **p (1 – p)**.
- Im Falle **p = 1/2** gehen dann etwa **3/4** aller ursprünglichen Bits verloren und für alle anderen Werte von **p** ist der **Verlust** noch höher (dies ist der Preis für die Verbesserung der Zufälligkeit).
- Da die Wahrscheinlichkeit für ein Paar „**01**" bzw. „**10**" in der ursprünglichen Bitfolge gleich **p (1 – p)** ist, ergibt sich für die Wahrscheinlichkeit $\mathbf{p_0}$ und $\mathbf{p_1}$ für den Wert **1** bzw. **0** nach der Filterung der Wert **1/2**.

Kap. 7: Schlüsselmittelmanagement und Zufallszahlen

Teil 2: Primzahlen

- Primzahlen und Fundamentalsatz der Arithmetik
- Sieb von Eratosthenes
- Primzahlentests und Primteilerzerlegung
- Primzahlhäufigkeit und Primzahldichtefunktion

Primzahlen

Definition: Es sei $p \in \mathbf{Z}$ und $p > 1$. p heißt **Primzahl** (auch unzerlegbar oder irreduziebel) $\Leftrightarrow$ Es gibt keine Teiler a von p mit $1 < a < p$.

Eine Zahl $a \in \mathbf{Z}$ mit $|a| \geq 2$ heißt **zerlegbar**, wenn $|a|$ keine Primzahl ist.

Satz:

1. Es sei $p \in \mathbf{Z}$ und $p > 1$. p ist genau dann eine Primzahl, wenn gilt:

   $$\text{Aus } p \mid (a \cdot b) \;\Rightarrow\; p \mid a \text{ oder } p \mid b\,.$$

2. $\forall\, a \in \mathbf{N}$ mit $a \geq 2 \;\Rightarrow\; \exists$ Primzahl p mit $p \mid a$.
3. Jede Zahl $a \in \mathbf{N}$ mit $a \geq 2 \;\Rightarrow\;$ a ist das Produkt endlich vieler Primzahlen, d. h. es gilt die **Primfaktorzerlegung**:

   $$a = p_1 \cdot p_2 \cdot \ldots \cdot p_k$$

Fundamentalsatz der Arithmetik

Jede natürliche Zahl a > 1 besitzt eine eindeutige Primfaktorzerlegung der Form:

$$a = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k}$$

wobei $p_1, \ldots, p_k \in \mathbf{P}$ und $a_1, \ldots, a_k \in \mathbf{N}$.

Satz:

Sei n eine **un**gerade Zahl für die auch (n – 1) / 2 **un**gerade ist. Dann gilt:

1. n prim $\Rightarrow$ $a^{(n-1)/2} = \pm 1$ für $\forall\, a \in \mathbf{Z}_n \setminus \{0\}$
2. n nicht prim $\Rightarrow$ $a^{(n-1)/2} = \pm 1$ für höchstens die Hälfte der $a \in \mathbf{Z}_n \setminus \{0\}$

(Beweis folgt aus dem Fermatschen Satz, vgl. Übung!)

Primzahlen

- Es existieren **unendlich** viele Primzahlen.
- Es existiert **kein** bekanntes effizientes Verfahren für die Zerlegung großer Zahlen in ihre Primfaktoren.
- Die Zahl 1234567891 ist eine Primzahl.
- Die Zahl 1987654321 ist keine Primzahl, sondern das Produkt von 457 und 4349353.
- $(2^{77232917} - 1)$ ist die größte bislang bekannte Primzahl. Sie hat 23.249.425 Dezimalstellen und wurde 2018 entdeckt (Jonathan Pace, US-amerik. Germantown, Tennessee).
- Für jede natürliche Zahl n gibt $\pi(n)$ die Zahl der Primzahlen $\leq$ n an:

  $$\pi(n) \approx n \,/\, (\ln(n) - 1.08366) \approx n \,/\, \ln(n) \quad \text{für großes } n.$$

Primzahlen

<u>Definition</u>: Eine natürliche Zahl n > 1 heißt Primzahl, wenn sie nur durch 1 und sich selbst ohne Rest teilbar ist.

<u>Lösungsalgorithmus</u>:

1. Wir bringen alle Zahlen von 2 bis n in ein Sieb.
2. Wir nehmen die erste (kleinste) Zahl aus dem Sieb heraus und heben sie separat auf. Danach lassen wir alle *nicht-trivialen* Vielfachen dieser Zahl durch das Sieb fallen.
3. Dann nehmen wir die nächst größere der *übriggebliebenen* Zahlen aus dem Sieb heraus und heben auch diese separat auf. Alle Vielfachen dieser Zahl lassen wir wiederum durch das Sieb fallen.
4. Wir wiederholen den 3. Schritt so oft, bis keine Zahlen mehr im Sieb sind.
5. Die separat aufgehobenen Zahlen sind die gesuchten Primzahlen.

Primzahlen

Sieb von Eratosthenes

Erzeugt Primzahlen `p` unterhalb der Schranke `maxp`.

Pseudocode:

```
for p = 2 bis maxp put Z(p) := TRUE
for p = 2 bis √maxp
   if Z(p) == TRUE then
      {  k = p * p
         do
            Z(k) := FALSE
            k := k + p
         while k <= maxp    }
```

Ausgabe:

∀ Primzahlen `p` = 2 bis `maxp`, für die `Z(p) == TRUE` ist.

Primzahlen zwischen 0 und 300: Rechenbsp.: $n = 300$ $\ln(300) = 5.7$

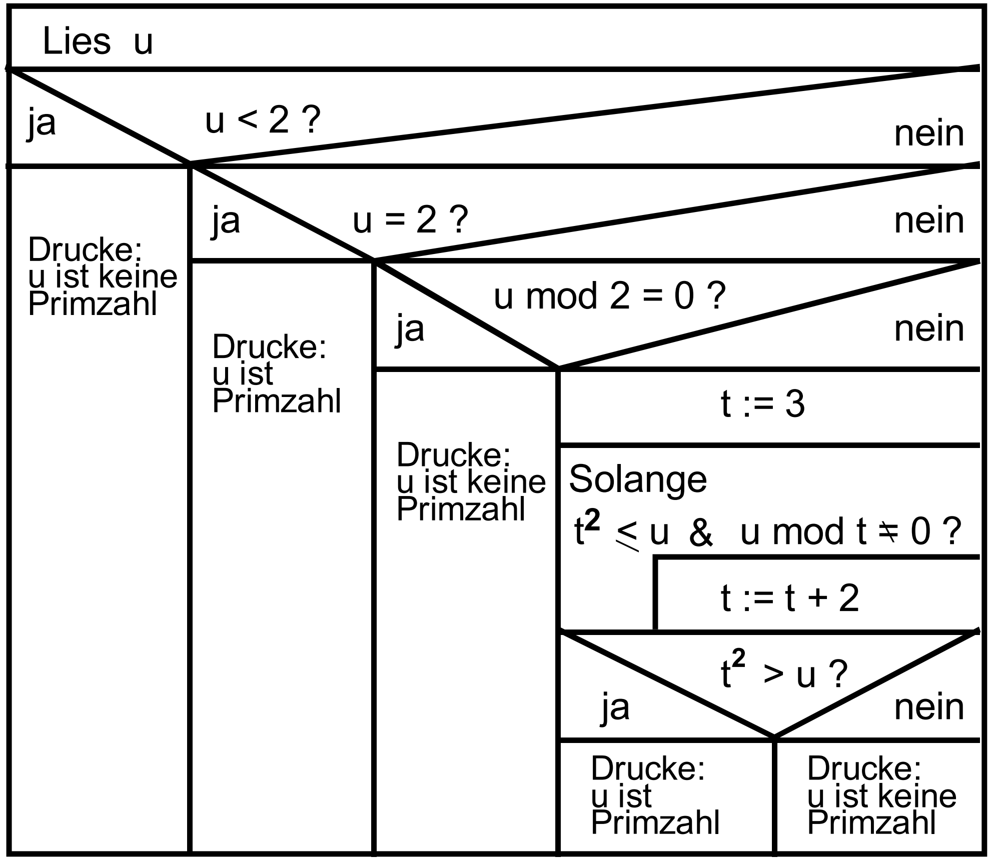
$\Rightarrow$ Es existieren 62 Primzahlen. $\Rightarrow \pi(300) \approx \underline{64}$

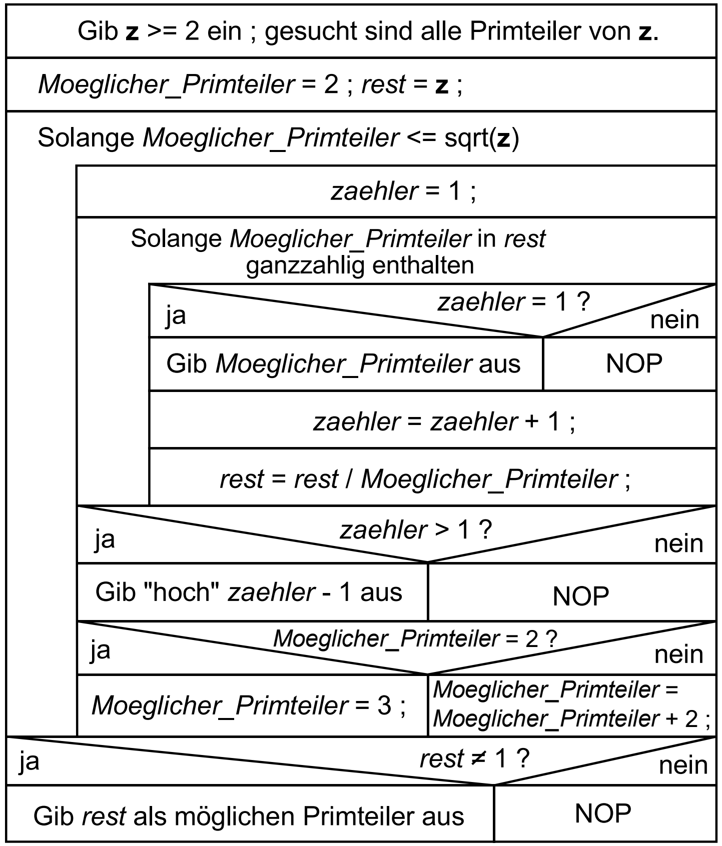
2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229
233	239	241	251	257	263	269	271	277	281
283	293								

Abschätzung:

$$\pi(n) \approx n / (\ln(n) - 1.08366) \approx n / \ln(n) \text{ für großes } n.$$

Primzahlentest und Primteilerzerlegung (deterministisch)





Statistischer Primzahlentest von Miller und Rabin (MRT)

Sei n eine ungerade Zahl für die auch $(n-1)/2$ ungerade ist (sog. *erlaubte Zahlen*):

1. Wähle Zufallszahlen $a_1, \ldots, a_k \in \{2, \ldots, n-2\}$ aus (auch *Zeugen* genannt).
2. Berechne $a_i^{(n-1)/2}$ in $\mathbf{Z}_n$ (folgt aus dem kleinen Satz von Fermat).
3. Falls alle $a_i^{(n-1)/2} = \pm 1$ bzw. 1 oder $n-1$,

   dann entscheide: n ist wahrscheinlich prim.

   sonst entscheide: n ist definitiv nicht prim.

Die Fehlerwahrscheinlichkeit des Tests ist $\leq (1/4)^k$, wenn k die Anzahl der gewählten Zeugen ist.

→ Test liefert nicht immer die korrekte Antwort!

MRT (Fortsetzung)

Der Miller-Rabin-Algorithmus kann deterministisch angewendet werden, indem alle **Basen** in einer bestimmten Menge getestet werden.

Aus $(\mathbf{Z}/n\mathbf{Z})^*$ ist a ein Zeuge für das Zusammengesetztsein von n mit

$$\forall\, a \in \{2, 3, \ldots, \min(n - 1, 2 \cdot \ln^2(n))\}.$$

Allerdings müssen in der Praxis nicht alle a bis zur Obergrenze $2 \cdot \ln^2(n)$, sondern nur eine sehr viel kleinere Menge getestet werden.

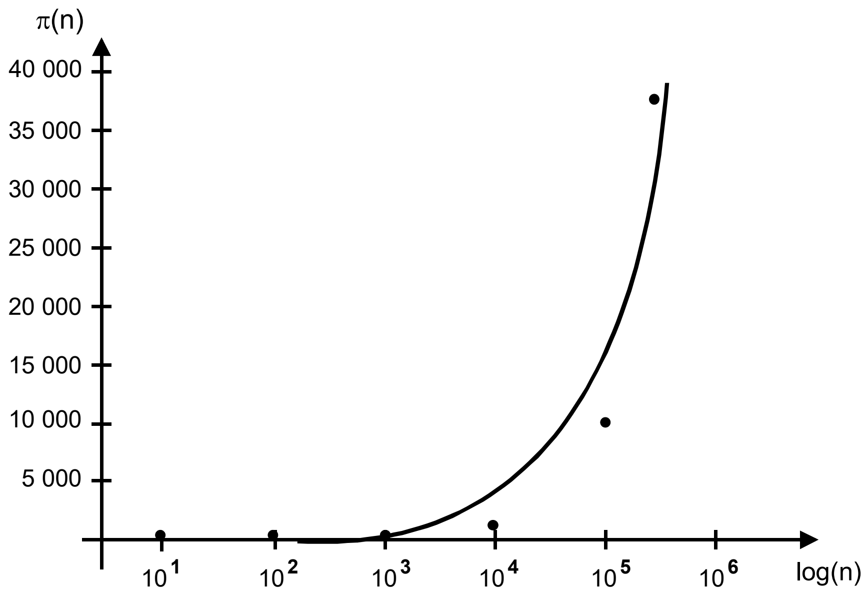
n	ungefähr	Als Zeuge zu testen sind die Zahlen
$< 2^{16}$	65.536	2, 3
$< 2^{32}$	4.294.967.296	2, 7, 61
$< 2^{64}$	$\approx 1.844\ 10^{19}$	2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37
< 3.317.044.064.679.887.385.961.981		2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41

Beispiele:

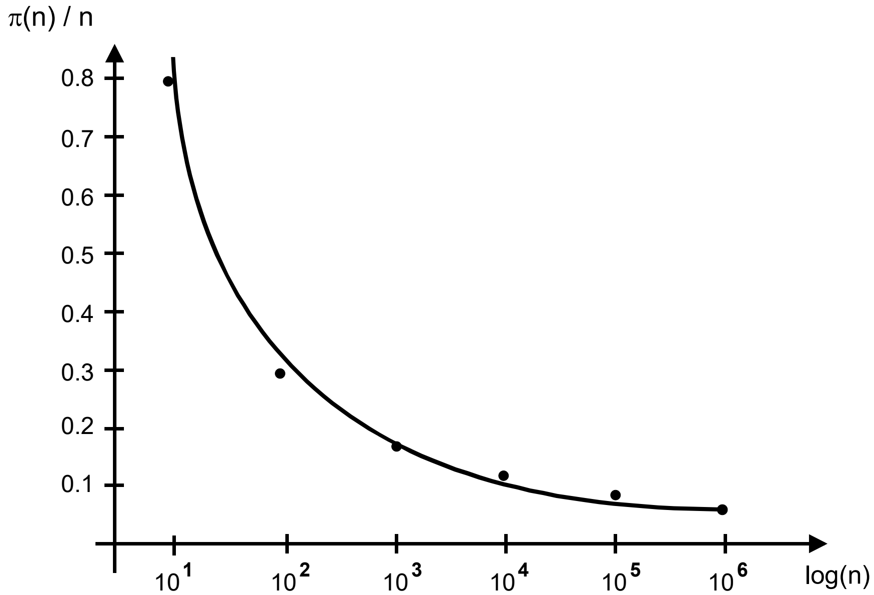
Zahl u	Primzahl ja / nein ?
15.681	nein
194.609	ja
224.711	ja
302.515.449	nein
2.147.483.647	nein

Zahl u	Zerlegung
137.917	$13 \cdot 103^2$
119.394.613	$13^2 \cdot 19^3 \cdot 103$
2 E9	$2^{10} \cdot 5^9$
183.495.637	$13^3 \cdot 17^4$

Primzahlhäufigkeit



Primzahldichtefunktion



$$\pi(n) = n \,/\, (\ln(n) - a_0) \quad \text{mit} \quad a_0 = 1.08366$$

Kap. 7: Schlüsselmittelmanagement und Zufallszahlen

Teil 3: Pseudozufallszahlengeneratoren

- Linearer Kongruenzgenerator und Rauschgenerator
- Blum-Blum-Shub-Generator (BBS)
- Lineare Schieberegister mit Rückkopplung
- Geffe-Generator
- Blum-Micali-Generator

Formel:

$$x_{n+1} = (a \cdot x_n + b) \bmod n$$

x_0 = Startwert (seed number) $\in$ **N+1**

a, b, n = Parameter (konstant) $\in$ **N+1**

x_{n+1} = Pseudozufallszahlen (n = 0, 1, 2, ...)

Parameterwahl:

Parameter	**Möglichkeit 1**	**Möglichkeit 2**
a	137153	7141
b	17	54773
n	2^{19}	259200

Ausgabe:

$$x_0, x_1, x_2, \ldots$$

Pseudozufallszahlengenerator

Formel:

$$x_{n+1} = a \cdot x_n - \text{int}\,(a \cdot x_n / b) \cdot b$$

x_0	= Startwert (seed number) $\in$ **N+1**
a, b	= Parameter (konstant) $\in$ **N+1**
int (...)	= ganzzahliger Anteil Klammerausdruck
x_{n+1}	= Pseudozufallszahlen (n = 0, 1, 2, ...)

Parameterwahl:

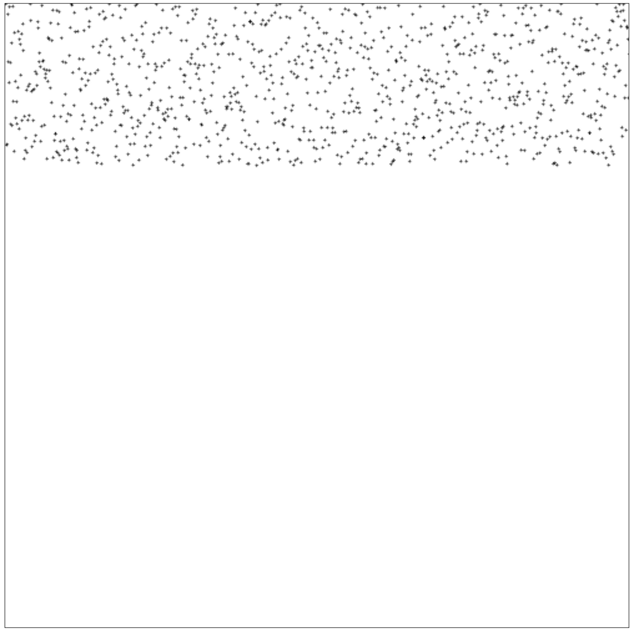
Parameter	**Möglichkeit 1**	**Möglichkeit 2**
a	23	29
b	100 000 001	1 000 001
x_0	439 147	691 156

Ausgabe:

$$x_1, x_2, x_3, \ldots$$

Linearer Kongruenzgenerator

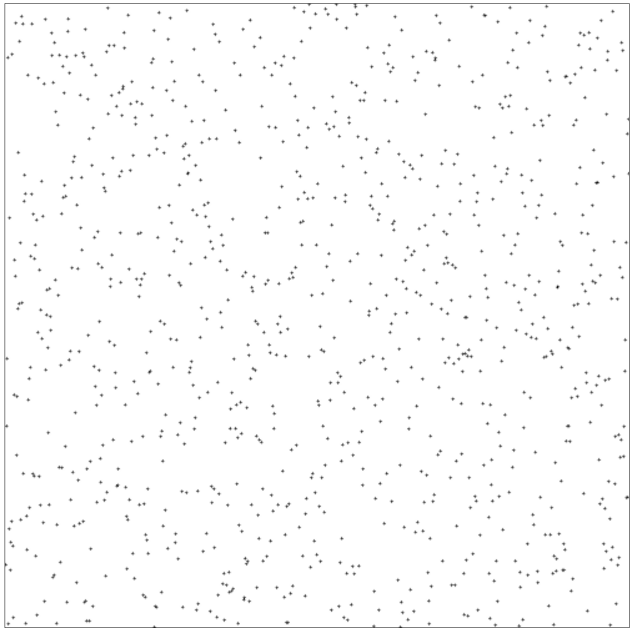
x_0 = 4711 a = 7141 b = 54773
n = 259200



x-Achse: 1, 2, 3, …, 1000
y-Achse: $x_1, x_2, x_3, \dots, x_{1000}$

Integerzahlengenerator

x_0 = 439147 a = 23 b = 100000001
(gerechnet von Marcell Dietl)



x-Achse: 1, 2, 3, …, 1000
y-Achse: $x_1, x_2, x_3, \dots, x_{1000}$

BBS-Generator

1. Wähle zwei große Primzahlen p und q, die beide bei Division durch 4 den Rest 3 ergeben. D. h.

$$p \equiv q \equiv 3 \bmod 4$$

2. Berechne $n = p \cdot q$ und wähle eine Zufallszahl s, die relativ prim zu n ist. Daraus berechne die Seed-Zahl x_0:

$$x_0 = s^2 \bmod n$$

3. Berechne nun mit $i = 1$ beginnend wiederholt:

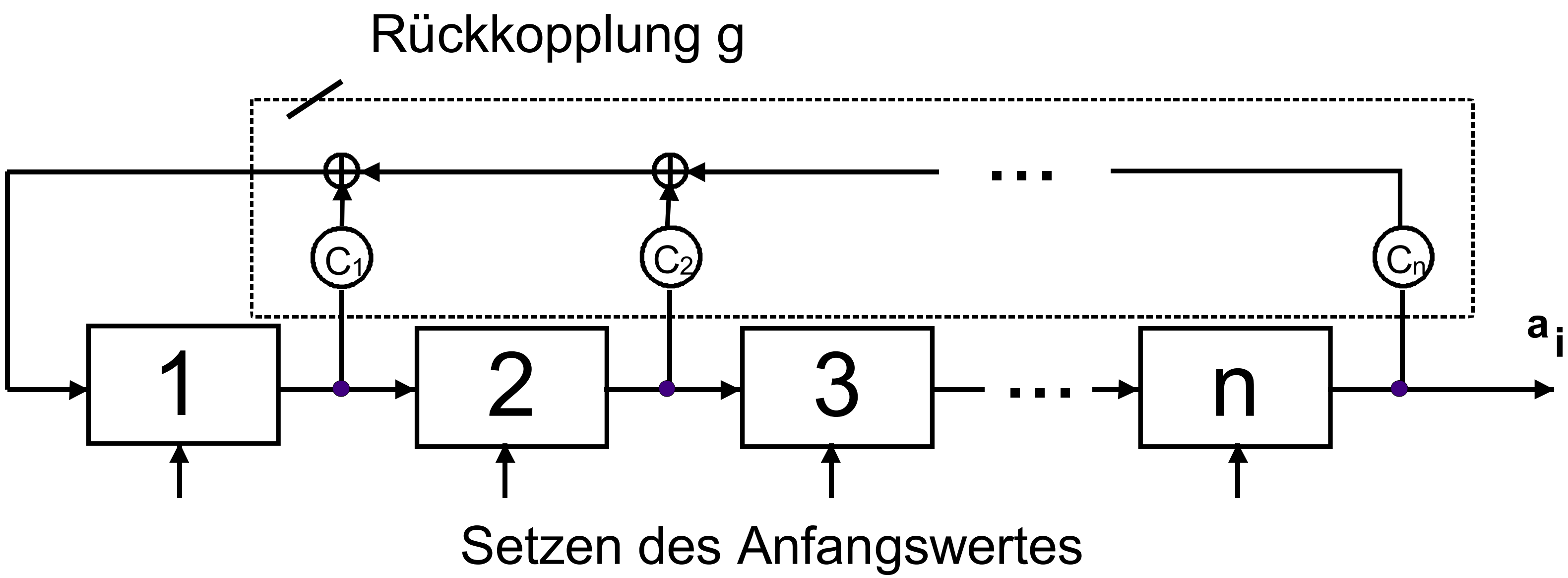
$$x_i = (x_{i-1})^2 \bmod n$$

4. Gib das folgende Bit b_i als i-tes Zufallsbit aus:

$$b_i = x_i \bmod 2 \in \{0, 1\}$$

Zufallszahlengenerator

- Je nachdem, ob die Rückkopplungsfunktion g linear oder nichtlinear ist, spricht man von linearen oder nichtlinearen Schieberegistern.
- Die maximale Periode eines n-stufigen Schieberegisters ist 2^n.
- Periodische Zufallsfolgen mit einer möglichst großen Periode bewirken gleichzeitig eine gute statistische Verteilung.
- Die Länge N des kürzesten linear-rückgekoppelten Schieberegisters, durch das die Erzeugung einer vorgegebenen Zufallszahlenfolge a ersetzt werden kann, heißt <u>lineare Komplexität</u> der Folge a.
- Wenn $2 \bullet N$ aufeinanderfolgende Output-Bits eines N-stufigen linear-rückgekoppelten Schieberegisters bekannt sind, können <u>alle</u> nachfolgenden Output-Bits vorausgesagt werden.



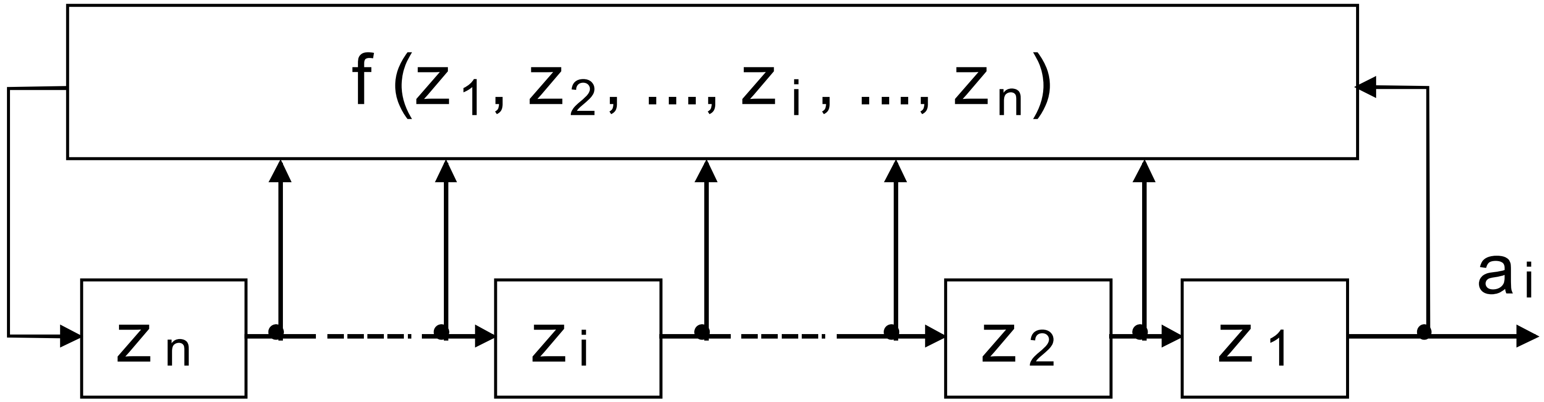
g = Rückkopplungsfunktion ($g = C_1 \cdot z_1 \oplus C_2 \cdot z_2 \oplus \ldots \oplus C_n \cdot z_n$)

n = Anzahl der Stufen

$C_i \in 0 \vee 1$; je nachdem, ob entspr. Rückführung vorhanden

$\oplus$ = XOR-Verknüpfung bzw. modulo-2-Addition

Prinzip des Linear Feedback Shift Register (LFSR)

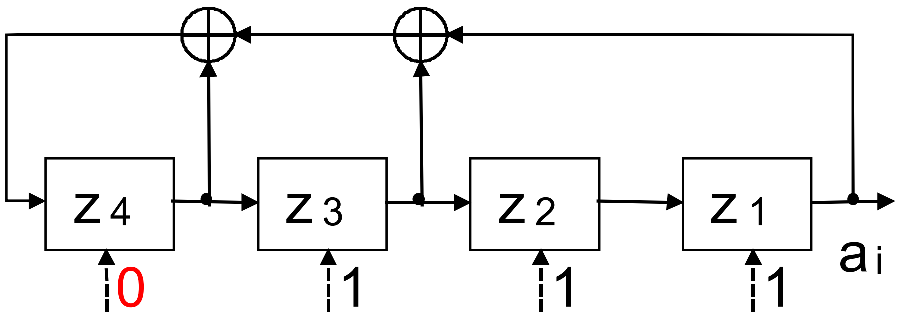


- In den Zellen z_1 bis z_n des n-stufigen LFSR können die Binärwerte 0 oder 1 gespeichert werden.
- Bei jedem Berechnungsschritt werden die Inhalte der Zellen z_n bis z_2 nach rechts geschoben.

- Der Zelleninhalt z_n wird dabei durch den Wert der binärwertigen Funktion $f(z_1, z_2, ..., z_i, ..., z_n)$ ersetzt.
- Der Zelleninhalt z_1 geht verloren und kann als binäre Pseudozufallsziffer a_i betrachtet werden.
- Damit die maximale Periodenlänge erreicht wird, wird bei LFSR die Rückkopplung durch speziell ausgewählte Zelleninhalte realisiert.
- Die Verknüpfung der rückgekoppelten Zelleninhalte geschieht durch XOR-Bildung bzw. Addition modulo 2.
- Liegt bei einem n-stufigen LFSR eine Ausgabefolge von 2n Bit vor, so lässt sich das Rückkopplungsnetzwerk rekonstruieren.
- Das Finden eines n-stufigen LFSR mit **maximaler Periode** lässt sich zurückführen auf das Finden eines primitiven Polynoms vom Grad n.

Pseudozufallszahlengeneratoren

Beispiel:



$\rightarrow$ Lineare Transformation

$$T = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

LFSR:

- 1. Spalte von T bestimmt die Positionen der Rückkopplung (hier 4, 3 und 1).
- Restliche Spalten beschreiben Verschiebung um eine Position nach rechts (Einheitsmatrix!).

Startvektor: $x = (0, 1, 1, 1) \Rightarrow$

Folgevektoren: (0, 0, 1, 1), (1, 0, 0, 1), (0, 1, 0, 0), (1, 0, 1, 0), (1, 1, 0, 1), (1, 1, 1, 0), (0, 1, 1, 1), (0, 0, 1, 1), ...

$\Rightarrow a_i = \{1, 1, 1, 0, 0, 1, 0, ...\}$, $d = 7$.

Diskussuion des Beispiels:

- Die Zustandsmenge **X** eines LFSR der Länge n lässt sich durch 0-1-Vekoren x der Form $x = (b_n, b_{n-1}, \ldots, b_2, b_1)$ darstellen.
- Die Funktion f kann als linaere Tranformation $f(x) = x \cdot T$ aufgefasst werden, wobei T ist eine binäre nxn-Matrix ist.
- Alle anfallenden Operationen werden modulo 2 ausgeführt – dies entspricht einer binären XOR-Verknüpfung.
- Bezeichnet x den Initialvektor des LFSR, so wird die Zustandsfolge x, $x \cdot T$, $x \cdot T^2$, $x \cdot T^3$, ... generiert.
- Die maximal erreichbare Periodenlänge beträgt $d = 2^n - 1$.

Zufallszahlengenerator

Geffe-Generator

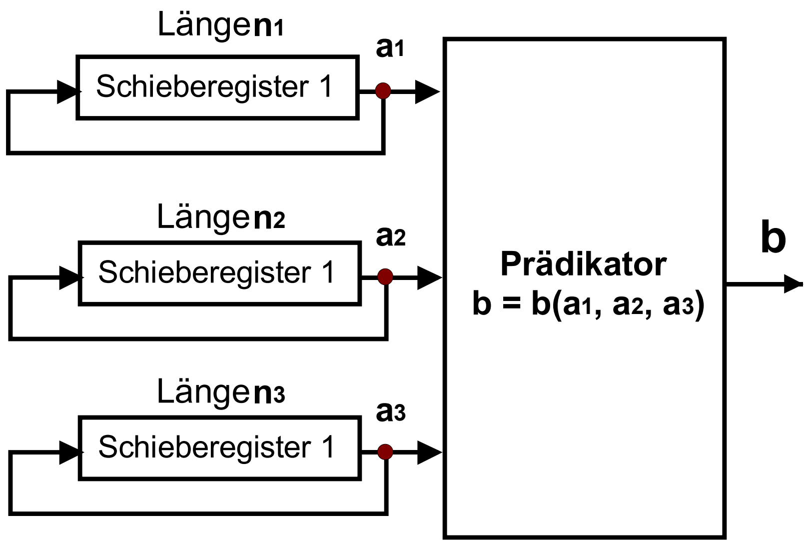
Der Geffe-Generator besteht aus drei zueinander primitiven, nicht-linear-rückgekoppelten Schieberegistern der Länge n_1, n_2 und n_3, d. h. ihre Produktdarstellungen enthalten <u>keine</u> gemeinsamen Faktoren.

<u>Periode P:</u>

$$P = \mathrm{kgV}\left(2^{n_1} - 1,\ 2^{n_2} - 1,\ 2^{n_3} - 1\right)$$

<u>Lineare Komplexität N:</u>

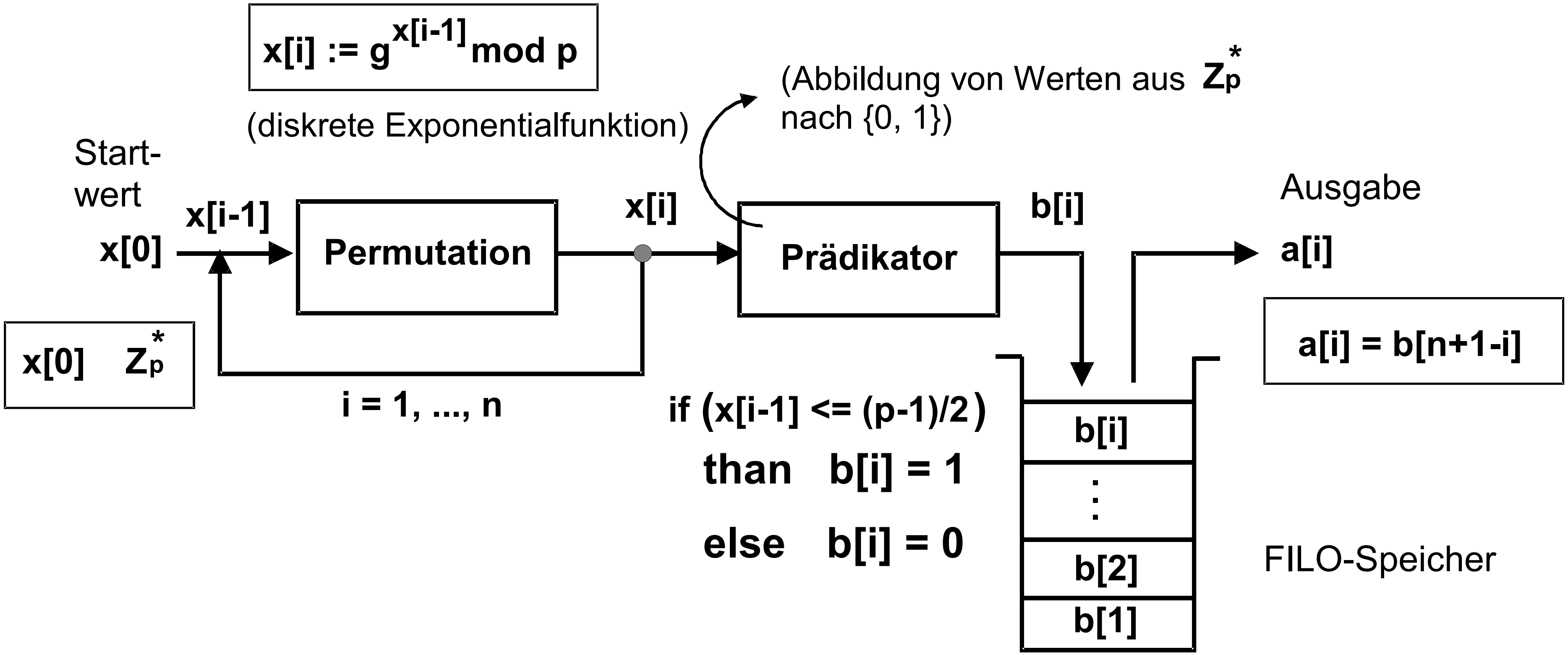
$$N = n_3 * n_1 + (n_1 + 1) * n_2$$



<u>Prädikator:</u> $b = a_3 \oplus \overline{a_1} * (a_2 \oplus a_3)$

Zufallszahlengenerator

Blum-Micali-Generator



Notation:

- p (Modulus) sei eine Primzahl; $p \in \mathbf{P}$
- n ist die Ausgabelänge; $n \in \mathbf{N}$
- $\mathbf{Z_n}$* ist eine **multiplikative Gruppe** (des Restklassenrings $\mathbf{Z_n}$) bestehend aus den Elementen von $\mathbf{Z_n}$, die zu n teilerfremd sind.

$$\mathbf{Z_n}^* := \left\{ a \in \mathbf{Z_n} \setminus \{0\} \mid ggT(a, n) = 1 \right\}$$

- g sei **Primitivwurzel** aus $\mathbf{Z_p}$* **;** Zahl g sollte so gewählt werden, dass sie eine möglchst große Untergruppe von $\mathbf{Z_p}$ erzeugt (vgl. DL-Problem).
- X[0] ist Saat aus $\mathbf{Z_p}$*

Blum-Micali-Generator

Ablauf:

```
var  x[0:n]   : array  of  integer ;
     b[1:n]   : array  of  {0, 1} ;

for  i = 1  to  n  do
    begin                                                /* Berechne nächsten Zwischen- */
        x[i] = g^(x[i–1]) mod p ;                        /* wert durch Permutation von x[i-1] */
        if  (x[i–1] ≤ (p – 1)/2)  then   b[i] = 1        /* Berechne das Prädikat mit */
                                  else   b[i] = 0        /* Hilfe von x[i - 1] */
    end ;
```

Output:

```
for  i = 1  to  n  do
  begin
        a[i] = b[n + 1 – i] ;                            /* Ausgabe in umgekehrter Reihenfolge */
        output a[i] ;
    end ;
```

Typische **Beurteilungskriterien** sind:

- Periode P
- Lineare Komplexität N
- Statistische Eigenschaften wie
  - Häufigkeitsverteilung von Bits und Bitgruppen
  - Korrelation zwischen Bitfolgen (z. B. zwischen Klar- und Schlüsseltexten)
  - Mittelwerttests
  - Abstände zwischen dem zweimaligen Auftreten eines Bitmusters
  - uvm.

Kap. 7: Schlüsselmittelmanagement und Zufallszahlen

Teil 4: Schlüsselmanagement

- Der Diffie-Hellman-Schlüsselaustausch (DH)
- Schlüsselhierarchie und Schlüsselklassen

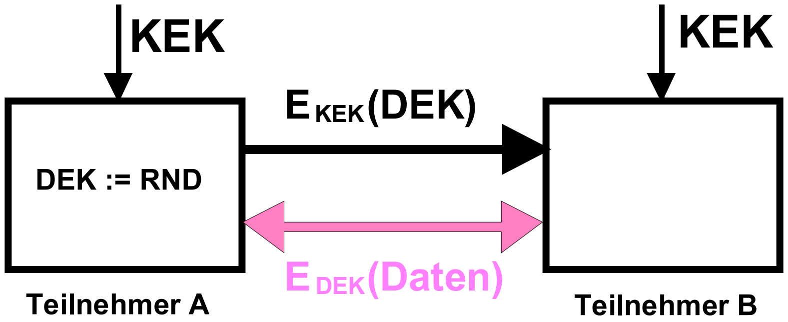
Generelle Anforderungen:

- Es muß gewährleistet sein, daß der ausgetauschte Schlüssel nur den befugten Teilnehmern bzw. Prozessen zugänglich ist.
- Die auszutauschenden Schlüssel müssen den befugten Teilnehmern unverändert und fehlerfrei zur Verfügung stehen.
- Bereits benutzte Schlüssel dürfen kein zweites Mal verwendet werden.
- Schlüsselaustauschprotokolle dürfen den Schlüsselaustausch nicht merklich verzögern.
- Der Schlüsselabsprache muß eine Authentifikation der Kommunikationspartner vorausgehen.
- Empfangsbestätigung und Verifikation des abgesprochenen Schlüssels sind in das verwendete Protokoll zu integrieren.

- Der für die Nachrichtenverschlüsselung verwendete Schlüssel (sog. Session Key oder Data Encryption Key **DEK**) sollte möglichst häufig wechseln, damit keine Analysen oder eingespielte Wiederholungen möglich sind.
- Der zur Verschlüsselung anderer Schlüssel verwendete Schlüssel heißt Master Key oder Key Encryption Key, kurz **KEK**.
- So wird in der Praxis mit dem KEK zunächst ein DEK verschlüsselt, mit dem anschließend der Datentransfer gesichert wird.
- Schließlich ist der Device Key (**DK**) ein ausgezeichneter, gerätespezifischer KEK, der im Rahmen der Geräteinitialisierung eingebracht oder hardwaremäßig im Gerät gespeichert ist.
- Damit verschiedene Angriffsmöglichkeiten unterbunden werden, ist es sinnvoll, den DEK von beiden Seiten gleichberechtigt zu bestimmen.

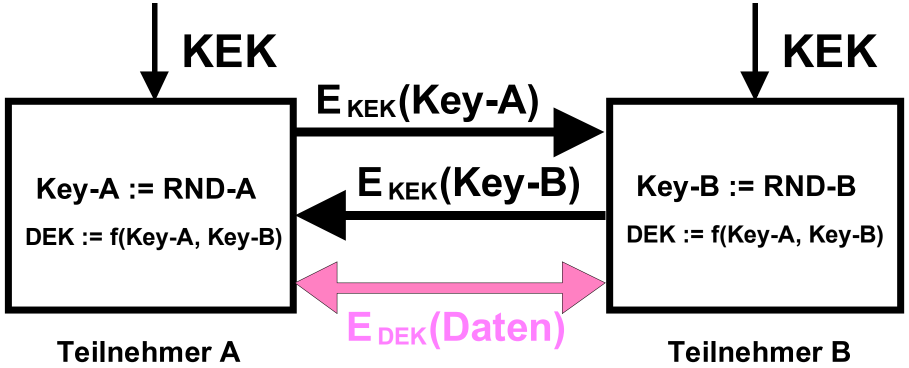
Gegenseitige Schlüsselabsprache

KEK und DEK



E = Symmetrisches Verfahren (z. B. DES)

KEK zur Verschlüsselung von Teilschlüsseln



$$N = \binom{n}{2} = \frac{n\,(n-1)}{2} \sim n^2$$

Diffie-Hellmann-Verfahren

In ihrer bedeutenden Arbeit haben **W. Diffie** und **M. Hellman** 1976 u. a. ein **asymmetrisches** Verfahren zur Schlüsselabsprache vorgestellt.

Vorteil: Wie bei asymmetrischen Verfahren üblich, müssen beide Kommunikationspartner von der Schlüsselvereinbarung über keinen gemeinsamen geheimen Schlüssel verfügen.

Nachteil: In der Schlüsselvereinbarung erfolgt keine Authentifikation, d. h. die Kommunikationspartner wissen nicht, mit **wem** sie den Schlüssel vereinbaren.

(Abhilfe schafft hier der Einsatz von Zertifizierungsinstanzen.)

	Sender **A**	Übertragungsweg	Empfänger **B**
	↓	↓	↓
wählt:	$\mathbf{a \in \{0, 1, p-2\}}$	unsicher bzw.	$\mathbf{b \in \{0, 1, p-2\}}$
	(**a** → geheim)	ungesichert	(**b** → geheim)
berechnet:	$\alpha = g^{\mathbf{a}}$ *mod* p		$\beta = g^{\mathbf{b}}$ *mod* p
austauschen:	α ––––– nach B ——————————>		
		<––––– nach A ——————————	β
berechnet:	$K_A = \beta^{\mathbf{a}}$ *mod* p		$K_B = \alpha^{\mathbf{b}}$ *mod* p

weil: $\beta^a \bmod p = (g^b)^a \bmod p = g^{ba} \bmod p = (g^a)^b \bmod p = \alpha^b \bmod p$

gilt: $\mathbf{K_A = K_B := K}$ (geheimer "Session" Key)

Sicherheit:
- Geheimhaltung von **a** bzw. **b**, die jedoch nicht ausgetauscht werden
- Lösung des diskreten Log-Problems, um von α(β) auf a(b) zu schließen

Initialisierung:	$p \in \mathbf{P}$;	öffentlich, beliebig	**P** = Primzahlen
	$2 \le g \le p-2$; $g \in \mathbf{N}$;	öffentlich, Primitivwurzel mod p	**N** = natürliche Zahlen

Schlüsselklassen

Schlüsselklasse	Benennung	Schlüssellänge*)	Lebensdauer*)
1	Session Key DEK	64 Bit	< 1 Tag
2	Master Key KEK	128 Bit	≈ 1 Monat
3	Device Key	128 Bit	≈ 2 Jahre

*) beispielhaft, abhängig vom konkreten Anwendungsfall