# Erzeugung von Pseudozufallszahlen

Proseminar

Kryptografie und Datensicherheit

Sommersemester 2009

Mario Frank

# Übersicht

1. Allgemeines
2. Anwendungen von PRBG
3. (k,l)-bit Generatoren
4. Unterscheidbarkeit und das Nachfolgerbit-Orakel
5. Der BBS Generator
6. Probabilistische Verschlüsselung
7. Optionales
8. Quellennachweise

# Allgemeines

Betrachtung: Zahlen als Bitketten

Erzeugung von langen Bitketten  hat eine hohe Zeitkomplexität (Polynomieller Zeitaufwand)

Lösung: Eingabe einer kurzen, zufälligen Bitkette (Seed)

Expansion des Seeds auf die gewünschte Länge

Eine erzeugte Zahl soll wie zufällig aussehen.

Sieht die Bitkette zufällig aus, dann ist der Generator sicher und wird als Pseudozufallszahlengenerator (PRBG) bezeichnet.

# Anwendungen von PRBG

- Erzeugung zufälliger Primzahlen p und q für das RSA-Modul
- Erzeugung von Schlüsseln für das One-time pad
- In der Probabilistischen Verschlüsselung
- In Simulationen

# (k,l)-bit Generatoren

1. Allgemeines
2. Sicherheit eines Bit-Generators
3. Linear gekoppeltes Schieberegister
4. Shrinking Generator
5. Kongruenzgenerator
6. RSA Generator

# Allgemeines

Definition 1: Sei l ≥ k+1 mit $k,l \in \mathbb{Z}$.

Ein (k,l)-Bit-Generator ist eine Funktion
$f: (\mathbb{Z}_2)^k \to (\mathbb{Z}_2)^l$, die in polynomieller Zeit
als Funktion von k berechenbar ist.
$s_0 \in (\mathbb{Z}_2)^k$ Ist ein zufälliger Seed und die Ausgabe
$f(s_0) \in (\mathbb{Z}_2)^l$ die erzeugte Bitkette.

Da die Funktion deterministisch ist, ist die Ausgabe von f nur vom Seed abhängig.

# Sicherheit eines Bit-Generators

- Ist es unmöglich, in polynomieller Laufzeit über k und damit auch l eine von einem PRBG erzeugte Bitkette von einer wirklich zufälligen zu unterscheiden, so ist der PRBG sicher.

- Beispiel: Sei $x^l$ eine zufällige Bitkette und $z^l$ eine generierte.

  Die Häufigkeit einer 1 in $x^l$ ist durchschnittlich ½ l.

  Ist die Häufigkeit einer 1 in $z^l$ >> ½ l, so kann man $x^l$ von $z^l$ unterscheiden und der PRBG ist nicht sicher.

# Linear rückgekoppelte Schieberegister (LRS)

- LSR vom Grad k kann $2^k$ -1 verschiedene Zustände annehmen.
- Bei Eingabe eines k-bit Seeds können $2^k$-k-1 weitere Bits erzeugt werden.

- Sehr schnell aber unsicher: Rekonstruktion des Seeds schon mit 2k bekannten bits möglich .

# Shrinking Generator

Seien $K_1$ und $K_2$ zwei linear rückgekoppelte Schieberegister vom Grad $k_1$ und $k_2$ mit $K_1:(\mathbb{Z}_2)^{k_1} \rightarrow (\mathbb{Z}_2)^{l}$ und $K_2:(\mathbb{Z}_2)^{k_2} \rightarrow (\mathbb{Z}_2)^{m}$.

Als Eingabe ist ein Seed mit $|s_0| = k_1 + k_2$ nötig.

$K_1$ erzeugt dann $(a_1,a_2,...,a_l)$ und $K_2$ erzeugt $(b_1,b_2,...,b_m)$.

$(z_1,z_2,...)$ wird dann wie folgt erzeugt: $\mathbf{z_i = a_{i_k}}$

Dabei ist $i_k$ die Stelle der k-ten 1 in der Folge $(b_1,b_2,...,b_m)$.

⟶ Die Ausgabe des Generators sieht nicht linear aus.

# Kongruenzgenerator

- Ein sehr schneller, aber auch unsicherer Generator.
- Nicht für die Kryptografie geeignet
- Sinnvoll für Simulationen

Algorithmus 1:

Sei $1 < m \in \mathbb{Z}$ und $1 \leq a,b \leq m-1$ .
Sei $k := 1 + \lfloor \log_2 m \rfloor$ mit
$k+1 \leq l \leq m-1$ und $s_0 \in \mathbb{N}$ mit $0 \leq s_0 \leq m-1$
das Seed.

Dann ist $s_i = (as_{i-1} + b) \bmod m$
für $1 \leq i \leq l$ und
$F(s_0) = (z_1, z_2, \ldots, z_l)$ mit $z_i = s_i \bmod 2$
und $1 \leq i \leq l$.
Dann ist f ein (k,l)-Kongruenzgenerator.

# RSA Generator

- Basiert auf das Faktorisierungsproblem
- Bedeutend sicherer als der Kongruenzgenerator und der lineare rückkopplungs Schieberegister.

Algorithmus 2:

Seien p und q zwei (k/2)-bit Primzahlen und n = pq.

Ein b sei so gewählt, dass ggt(b,φ(n))= 1.

n und b sind öffentlich und p und q sind geheim.

Sei $s_0 \in \mathbb{Z}_2^k$.

Für i ≥ 1 sei $s_{i+1} = s_i^b \bmod n$ .

und $f(s_0)=(z_1,z_2,...,z_l)$ mit

$z_i = s_i \bmod 2$ , $1 \le i \le l$

Dann ist f ein (k,l)-RSA Generator

# Unterscheidbarkeit und Bit-Orakel

1. Unterscheidbarkeit
2. Nachfolgerbit-Orakel

# Unterscheidbarkeit

Def: Seien $p_0$ und $p_1$ Wahrscheinlichkeitsverteilungen aller Bitketten der Länge l.
Dann ist $p_j(z^l)$ mit j=0,1 und $z^1 \in (\mathbb{Z}_2)^1$ die Wahrscheinlichkeit, dass $z^l$ in $p_j$ enthalten ist.

Sei dst: $(\mathbb{Z}_2)^1 \to \{0,1\}$ eine Funktion und $\epsilon > 0$

und $E_{dst}(p_j) = \sum_{z^l \in (\mathbb{Z}_2)^l : dst(z^l) = 1} p_j(z^l).$

dst wird ε-Differenzierer von $p_0$ und $p_1$ genannt, wenn

$| E_{dst}(p_0) - E_{dst}(p_1) | \geq \varepsilon$ gilt und $p_0$ und $p_1$ sind ε-differenzierbar, wenn es einen ε-Differenzierer von $p_0$ und $p_1$ gibt.

Ist $dst(z^l)$ in polynomieller Laufzeit als Funktion von l berechenbar, dann ist dst ein Differenzierer mit polynomieller Laufzeit.

# Unterscheidbarkeit (2)

- Die Wahrscheinlichkeitsverteilung $p_j$ ist definiert als

  F(x)= p[X≤x] . Dabei ist X eine zufällige Größe und $x \in \mathbb{R}$ .
- dst($z^l$) ist die Abschätzung, ob $z^l$ in $p_0$ oder $p_1$ enthalten ist.
- $E_{dst}(p_j)$ mit j=0,1 ist die erwartete Ausgabe von dst($p_j$).
- Ist dst ein randomisierter Differenzierer dann ist

$$E_{dst}(p_j) = \sum_{z^l \in (\mathbf{Z}_2)^l} (p_j(z^l) \times P[dst(z^l=1)]).$$

- Ein Differenzierer kann entscheiden , ob ein $z^l$ von einem PRBG erzeugt wurde oder zu der uniformen Warscheinlichkeitsverteilung $p_u$ gehört.

# Uniformität

- In $p_u$ haben bei einer bitkette $z^l$ die verschiedenen $2^l$ durchschnittlich dieselbe Auftrittswahrscheinlichkeit ½ l.
- Die durch einen (k,l)-bit Generator erzeugte Wahrscheinlichkeitsverteilung sei $p_f$ und f erzeuge aus einem zufälligen k-bit Seed eine l-bit Sequenz.

  Da f eine bijektive Funktion ist, gibt es also $2^k$ verschiedene Seeds und somit auch $2^k$ erzeugte Pseudozufallszahlen, die in $p_f$ mit einer Warscheinlichkeit von ½ k vorkommen.

  Die anderen $2^l – 2^k$ treten nie auf.

  ⟶ $p_f$ ist nicht uniform.

# Nachfolgerbit-Orakel

Def: Sei f ein (k,l)-bit Generator und $1 \leq i \leq l-1$.

Dann ist nbo: $(\mathbb{Z}_2)^{i-1} \rightarrow (\mathbb{Z}_2)$ mit

$nbo(z^{i-1}) = z_i$ .

Dabei sind $z^{i-1}$ die ersten i-1 bits, die f aus dem Seed $s_0$ erzeugt hat und $z_i$ das nachfolgende Bit

# Nachfolgerbit-Orakel (2)

- Satz 1: Sei ε > 0 und f ein (k,l)-bit Generator.

  Weiter sei $p_f$ die von f erzeugte Wahrscheinlichkeits-verteilung.

  Dann ist die Funktion nbo ein ε-i-tes-bit-Orakel gdw.

$$\sum_{z^{i-1} \in (\mathbb{Z}_2)^{i-1}} (p_f(z^{i-1}) \text{ x } P[z_i = \text{nbo}(z^{i-1}) | z^{i-1}]) >= \frac{1}{2} + \epsilon \ .$$

$p_f(z^{i-1})$ ist die Wahrscheinlichkeit, dass $z^{i-1}$ von f erzeugt wurde.

# Nachfolgerbit-Orakel (3)

Satz 2:

Sei nbo ein ε-i-tes-bit-Orakel für den (k,l)-bit Generator f und $p_f$ die von f auf $(\mathbb{Z}_2)^l$ erzeugte Wahrscheinlichkeitsverteilung und $p_u$ die uniforme Wahrscheinlichkeitsverteilung über $(\mathbb{Z}_2)^l$ .

Dann ist DIFF ein ε-Differenzierer von $p_f$ und $p_u$ mit polynomieller Laufzeit.

Algorithmus 3:

DIFF($z^i$)

external nbo

$z \leftarrow$ nbo($z^{i-1}$)

If $z = z^i$

then return(1)

else return (0)

# Nachfolgerbit-Orakel (4)

Satz 3: Sei dst ein ε-Differenzierer von $p_f$ und $p_u$ mit polynomieller Laufzeit, wobei $p_f$ die vom (k,l)-bit Generator f auf $(\mathbb{Z}_2)^l$ erzeugte Wahrscheinlichkeitsverteilung und $p_u$ die uniforme Wahrscheinlichkeitsverteilung über $(\mathbb{Z}_2)^l$ ist.

Dann gibt es für $1 \leq i \leq l-1$ ein $\frac{\epsilon}{l}$-i-tes-bit-Orakel mit polynomieller Laufzeit für f.

# Nachfolgerbit-Orakel (5)
# Randomisierter Algorithmus

- NBO erzeugt $z_l$
- Gibt dst 0 aus, ist $z_i$ vermutlich von f erzeugt.
- Gibt dst 1 aus, ist $z_i$ vermutlich zufällig und $1-z_i$ wäre warscheinlich das von f erzeugte bit.

Algorithmus 4:

NBO($z^{i-1}$)

external dst

Wähle $(z_i,...,z_l) \in (\mathbb{Z}_2)^{l-i-1}$ zufällig

$z \leftarrow dst(z_1,z_2,...,z_l)$

return($z+z_i$ mod 2)

# Der Blum-Blum-Shub Generator (BBS)

Algorithmus 5:

Seien p und q (k/2)-bit Primzahlen mit $p \equiv q \equiv 3 \bmod 4$ und $n = pq$, wobei n ungerade sein sollte.

Weiter sei QR(n) die Menge der quadratischen Reste modulo n

und $s_0 \in QR(n)$.

Seien für $0 \leq i \leq l-1$ $s_{i+1} = s_i^2 \bmod n$ und

$f(s_0) = (z_1, z_2, ..., z_l)$ mit $z_i = s_i \bmod 2$ und $1 \leq i \leq l$ .

Dann ist f ein (k,l)-bit Generator und wird BBS-Generator genannt.

Um sicherzustellen, dass $s_0 \in QR(n)$ wähle man $s_{-1} \in \mathbb{Z}_n^*$

und erzeuge $s_0 = s_{-1}^2$ .

# Problem der kompositen quadratischen Reste (kqR)

Sei $n \in \mathbb{Z}$ mit n = pq, p≠q und p und q unbekannt.

Weiter sei p mod 2 = q mod 2 = 1 und $x \in \mathbb{Z}_n^*$ , so dass

$(\frac{x}{n} = 1)$.

Das Problem besteht in der Frage, ob x ∈ QR(n).

Ist $(\frac{x}{n} = 1)$ , dann ist x ∈ QR(n) gdw. $(\frac{x}{p} = 1.)$

Da die Faktorisierung von n unbekannt ist, ist das Problem nicht lösbar

# Sicherheit des BBS Generators

1. Das Vorgängerbit-Orakel
2. QR-Test
3. Monte Carlo QR-Test

# Das Vorgängerbit-Orakel (VbO)

Sei f ein (k,l)-bit BBS Generator.

Dann ist $VbO(z_1,z_2,...,z_l) = z_o = s_0 \bmod 2$ mit $s_0 \in QR(n)$.

Ist die Wahrscheinlichkeit, dass VbO $z_0$ richtig bestimmt ≥ ½+ε , dann ist VbO ein ε-Vorgängerbit-Orakel, das heißt,

VbO ist ein ε-Vorgängerbit-Orakel gdw.

$$\sum_{s_0 \in QR(n)} (p_f(s_0) \text{ x } P[s_0 \bmod 2 = z_0 = VbO(z^l)|z^l]) >= \frac{1}{2} + \epsilon \ .$$

# Das Vorgängerbit-Orakel (2)

Satz 4: Angenommen, es gibt einen ε-Differenzierer von $p_f$ und $p_u$ , wobei $p_f$ die vom (k,l)-bit BBS Generator f auf $(\mathbb{Z}_2)^l$ erzeugte Wahrscheinlichkeitsverteilung und $p_u$ die uniforme Wahrscheinlichkeitsverteilung über $(\mathbb{Z}_2)^l$ ist.

Dann gibt es ein $\frac{\epsilon}{l}$ -Vorgängerbit-Orakel mit polynomieller Laufzeit für f.

# Das Vorgängerbit-Orakel (VbO)

Sei f ein (k,l)-bit BBS Generator.

Dann ist $VbO(z_1,z_2,...,z_l) = z_o = s_0 \bmod 2$ mit $s_0 \in QR(n)$.

Ist die Wahrscheinlichkeit, dass VbO $z_0$ richtig bestimmt ≥ ½+ε , dann ist VbO ein ε-Vorgängerbit-Orakel, das heißt,

VbO ist ein ε-Vorgängerbit-Orakel gdw.

$$\sum_{s_0 \in QR(n)} (p_f(s_0) \text{ x } P[s_0 \bmod 2 = z_0 = VbO(z^l)|z^l]) >= \frac{1}{2} + \epsilon \ .$$

# QR-Test

Überprüft, ob x ein quadratischer Rest ist.

Gibt der QR-Test yes aus, dann ist x ein gültiger Seed.

Gibt der QR-Test no aus, dann ist x ein Pseudo-Quadrat $x \in \widetilde{QR}(n)$ und die erzeugte Bitkette ist die, die der BBS Generator mit -x als Seed erzeugt hätte.

Algorithmus 6:

QR-TEST(x,n)

external VbO

$s_1 \leftarrow x_2 \bmod n$

$z_1 \leftarrow s_1 \bmod 2$

Errechnung von $(z_2,...z_l)$ aus $s_1$ mit dem BBS-Generator

$z \leftarrow VbO(z_1,z_2,...,z_l)$

if (x mod 2) = z

then return (yes)

else return (no)

# QR-Test (2)

Satz 5: Sei VbO ein δ-Vorgängerbit-Orakel mit polynomieller Laufzeit für den (k,l)-bit BBS Generator f.

Dann erzeugt der QR-Test die quadratische Residuosität mit einer Wahrscheinlichkeit P ≥½ + δ in polynomieller Laufzeit.

Dabei ist die Wahrscheinlichkeit über alle uniform zufällig gewählten x $\in$ QR(n)$_{\cup \widetilde{QR}(n)}$ gemittelt.

# Monte Carlo QR-Test

Bestimmt die quadratische Residuosität.

Für alle x $\in$ QR(n)$\cup\widetilde{QR}(n)$

gibt der MC-Algorithmus

mit P $\geq$ ½ + δ die richtige Antwort.

Algorithmus 7: MC-QR-TEST(x)

external QR-TEST

Wähle $r \in \mathbb{Z}_n^*$ zufällig

x' ← $r^2$x mod n

Wähle s $\in \{1,-1\}$ zufällig

x' ← sx' mod n

- t ← QR-TEST(x')

if ((t=yes)&(s=1)|(t=no)&(s=-1)

then return (yes)

else return (no)

# Monte Carlo QR-Test (2)

Satz 6: Angenommen, der QR-Test bestimmt die quadratische Residuosität mit einer Wahrscheinlichkeit $P \geq ½ + \delta$ in polynomieller Laufzeit richtig.

Dann ist der MC-QR-Test ein Monte Carlo Algorithmus mit polynomieller Laufzeit für komposite quadratische Reste mit einer Fehlerwahrscheinlichkeit $P_{(err)} \leq ½ - \delta$.

# Monte Carlo CR-Test (3)

Satz 7: Sei A ein unverzerrter MC-Algorithmus mit maximaler Fehlerwahrscheinlichkeit $P_{(err)} \leq ½ - \delta$.

Lässt man A n = 2m +1 mal mit dem selben x ablaufen und gibt die Antwort aus, die am häufigsten ausgegeben wurde, dann ist die Fehlerwahrscheinlichkeit

$$P_{(err)} \leq \frac{(1 - 4\delta^2)^m}{2}.$$

# Abschließend

- Nach Yao sind die vom BBS Generator erzeugten Bitfolgen kryptografisch sicher, bestehen also jedem probabilistischen, in polynomieller Laufzeit ablaufendem statistischem Test. [4]
- Die maximale Periodenlänge des BBS ist abhängig von den gewählten p und q. Ist die Periodenlänge klein, so können sich Bitfolgen innerhalb einer Bitkette wiederholen.
- $\pi(s_0)$ ist die Periodenlängen von $z^l$.
- $\pi$ ist maximal, wenn $\pi = \lambda(\lambda(n))$ gilt, wobei $\lambda(n)$ die Carmichael-Funktion ist.

# Probabilistische Verschlüsselungen

1. Definition eines Probabilistischen Kryptosystems
2. Goldwasser-Micali Public-key Kryptosystem
3. Blum-Goldwasser Public-key Kryptosystem

# Definition eines probabilistischen Kryptosystems

Ein probabilistisches Kryptosystem ist ein 6-tupel (P,C,K,E,D,R),

P - Menge der Klartexte, C – Menge der Ciphertexte

K – Der Schlüssleraum, R – eine Menge von PRBG

Seien $k \in K$, $e_k \in E$ eine öffentliche Verschlüsselungsregel und

$d_k \in D$ eine geheime Entschlüsselungsregel.

$e_k: P \times R \rightarrow C$ und $d_k: C \rightarrow P$ seien Funktionen mit

$$d_k(e_k(b,r))=b \text{ mit } b \in P \text{ und } r \in R$$

Dabei gilt insbesondere $e_k(x,r) \neq e_k(x',r)$, wenn $x \neq x'$ .

# Definition eines probabilistischen Kryptosystems (2)

Sei $\varepsilon$ ein Sicherheitsparameter.

Für alle festen $k \in K$ und für alle $x \in P$ sei $p_{k,x}$ die Wahrscheinlichkeitsverteilung auf C.

Dabei ist $p_{k,x}(y)$ die Wahrscheinlichkeit über alle zufälligen $r \in R$, dass y der Geheimtext ist, wenn k der Schlüssel und x der Klartext ist.

Seien $x,x' \in P$ mit $x \neq x'$ und $k \in K$. Dann sind $p_{k,x}$ und $p_{k,x'}$ nicht

in polynomieller Laufzeit $\varepsilon$-differzierbar.

# Goldwasser-Micali Public-key Kryptosystem

Sei n = pq, wobei p≠q und p und q ungerade und geheim sind und $m \in \widetilde{QR}(n)$.

Weiter seien n und m öffentliche ganze Zahlen, P = {0,1},

C=R= $\mathbb{Z}_n^*$ und k = {(n,p,q,m)}

Verschlüsselung: $e_k(x,r) = m^x r^2 \bmod n$

Entschlüsselung: dk(y) = 0, wenn $y \in QR(n)$

1, wenn $y \in \widetilde{QR}(n)$,

mit x = 0 oder 1 und r,y $\in \mathbb{Z}_n^*$ .

# Goldwasser-Micali Public-key Kryptosystem (2)

- Sind $p \equiv 3 \pmod 4$ und $q \equiv 3 \pmod 4$, kann man für $m = -1$ wählen. Dadurch würde die Berechnung von $m^x$ keine Exponentierung benötigen, was die Verschlüsselung effizienter macht.
- Starke Datenexpansion
- Der Algorithmus ist nur sicher gegen Faktorisierung, wenn n eine 1024 bit Zahl ist.
- Daturch ist der Geheimtext aber 1000 mal größer als der Klartext.

# Blum-Goldwasser Public-key Kryptosystem

Sei n = pq, wobei p≠q und p und q geheime Primzahlen mit p≡q≡3(mod 4) seien. Die Zahl n sei öffentlich und

P = $(\mathbb{Z}_2)^l$, C = $(\mathbb{Z}_2)^l$ x $\mathbb{Z}_n^*$ und R = $\mathbb{Z}_n^*$.

K sei K = (n,p,q) , $x \in (\mathbb{Z}_2)^l$ und $r \in \mathbb{Z}_n^*$.

Verschlüsselung:

1. Berechne $z^l$ mit dem BBS Generator aus $s_0 = r$
2. Berechne $s_{l+1} = s_0^{2^{l+1}} \bmod n$
3. Berechne $y_i = (x_i + z_i) \bmod 2$ für $1 \le i \le l$
4. $e_k(x,r) = (y_1,\ldots,y_l,s_{l+1})$

# Blum-Goldwasser Public-key Kryptosystem

Entschlüsselung:

1. $a_1 = (( p + 1 ) /4)^{l+1} \bmod (p-1)$
2. $a_2 = (( q + 1 ) /4)^{l+1} \bmod (q-1)$
3. $b_1 = s_{l+1}^{a_1} \bmod p$
4. $b_2 = s_{l+1}^{a_2} \bmod q$
5. Verwende den chinesischen Restsatz, sodass

   $r \equiv b_1 \pmod p$ und $r \equiv b_2 \pmod q$
6. Berechne $z^l$ mit dem BBS Generator aus $s_0 = r$
7. Berechne $x_i = (x_i + z_i) \bmod 2$ mit $1 \leq i \leq l$
8. Der Klartext ist $x = (x_1, x_2, ..., x_l)$

# Optionales
# Die Kolmogorov-Komplexität

- Ein Maß für die Strukturiertheit einer Zeichenkette.
- Ist eine Zeichenkette strukturiert, so kann sie komprimiert werden
- Ist eine Zeichenkette unstrukturiert, so sieht sie zufällig aus und ist nicht komprimierbar

# Die Kolmogorov-Komplexität (2)

- Definition 1: Für jedes Wort $x \in (\Sigma_2)^*$ ist die Kolmogorov-Komplexität K(x) des Wortes x die binäre Länge des kürzesten Pascal-Programms, das x generiert.

- Definition 2: Ein Wort $x \in (\Sigma_2)^*$ heißt zufällig, falls $K(x) \geq |n|$.

  Eine Zahl n heißt zufällig, falls

  $K(n) = K(n_2) \geq \lceil \log_2(n+1) \rceil - 1$

# Quellennachweise


- [1]Cryptography - Theory and Practice, Stinson, 3. Auflage, Chapman & Hall
- [2]The Art of Computer Programming,Volume 2: Seminumerical Algorithms, 2. Auflage, Donald E. Knuth, Addison- Wesley Publishing Company
- [3]Theoretische Informatik, Juraj Hromkovič, 3. Auflage, Teubner Verlag
- [4]www1.informatik.uni-erlangen.de/tree/Lehre/SS07/BBS.ps