4.4 Beispiele und praktische Überlegungen

Der BBS-Generator ist also perfekt *unter einer „vernünftigen", aber unbewiesenen Annahme,* nämlich der Quadratrest-Vermutung. Wir wissen aber nichts Konkretes:

- Wie groß muss man die Bitzahl des Parameters m wählen?
- Welches sind die schlechten Werte für den Modul m?
- Wie groß ist der Anteil der schlechten Startwerte bei gegebenem m?
- Wieviele Bits am Stück darf man verwenden bei gegebenem Modul und Startwert, d. h., welche Wahl für das Streckungspolynom g ist angemessen?

Die hergeleiteten Aussagen sind qualitativ, nicht quantitativ. Die tatsächliche Qualität der erzeugten Zufallsbits, sei es für statistische oder kryptographische Anwendungen, kann bis auf weiteres nur empirisch beurteilt werden. Man kann davon ausgehen, dass für Moduln, die sich den gegenwärtigen Faktorisierungsalgorithmen noch sicher entziehen, also etwa ab 2048 Bit Länge, bei zufälliger Wahl des Moduls und des Startwerts die Gefahr extrem gering, auf jeden Fall vernachlässigbar, ist, eine „schlechte" Bitfolge zu erzeugen. Die bewiesenen Ergebnisse lassen allerdings auch zu, dass man die Menge M aller Blum-Zahlen von vorneherein um bekannte schlechte Fälle verkleinert, um die Quadratrest-Vermutung zu retten.

Hier stellt sich auch die Frage, wie groß in Abhängigkeit vom Startwert die Gefahr ist, einen kurzen Zyklus zu erzeugen. Nach den Ergebnissen von Shparlinski – siehe das Literaturverzeichnis zur Vorlesung – ist diese Gefahr vernachlässigbar. Dort gibt es auch nichttriviale untere Schranken für das Linearitätsprofil des BBS-Generators.

Als weitere Frage drängt sich auf: Darf man, um die praktische Verwertbarkeit des Generators zu verbessern, in jedem Iterationsschritt mehr als nur ein Bit verwenden? Wenigstens 2? Diese Frage wurde von Vazirani/Vazirani und unabhängig von Alexi/Chor/Goldreich/Schnorr teilweise, aber auch wieder nur qualitativ, beantwortet: Wenigstens $O(^2\log {}^2\log m)$ der niedrigsten Bits sind „sicher". Je nach Wahl der Konstanten, die in dem „O" steckt, muss man die Bitzahl des Moduls genügend groß machen und auf empirische Erfahrungen vertrauen. Entscheiden wir uns für genau $^2\log {}^2\log m$ Bits. Hat dann m 2048 Bits, also etwa 600 Dezimalstellen, so kann man also in jedem Schritt 11 Bits verwenden. Um $x^2 \bmod m$ zu berechnen, wenn m eine n-Bit-Zahl ist, braucht man $(\frac{n}{32})^2$ Multiplikationen von 32-Bit-Zahlen und anschließend ebensoviele Divisionen „64 Bit durch 32 Bit". Bei $n = 2048$ sind das $2 \cdot (2^6)^2 = 8192$ solcher elementaren Operationen, um 11 Bits zu erzeugen, also etwa 800 Operationen pro Bit. Lineare Kongruenzgeneratoren im 32-Bit-Bereich brauchen pro Schritt zwei

elementare Operationen; selbst wenn man jeweils nur 20 Bits verwendet, hat man noch einen Geschwindigkeitsvorteil mit dem Faktor 8000, allerdings bei wesentlich geringerer „Zufallsqualität“ der Bits.

In der Literatur werden einige weitere Pseudozufallsgeneratoren betrachtet, die nach ähnlichen Prinzipien funktionieren wie der BBS-Generator. Stets wird von vorneherein ein nichtkonstantes Polynom $g \in \mathbb{N}[X]$ festgelegt, das die Anzahl der maximal auszugebenden Bits spezifiziert.

Der RSA-Generator (Shamir). Man wählt einen zufälligen Modul m, der ein Produkt zweier großer Primzahlen p, q ist, und einen Exponenten d, der teilerfremd zu $\varphi(m) = (p-1)(q-1)$ ist, ferner einen zufälligen Startwert $x = x_0$ im Zustandsraum $\mathbb{M}_m$. Die interne Transformation ist $T_m(x) = x^d \bmod m$. Man bildet also $x_i = x_{i-1}^d \bmod m$ und gibt das letzte Bit oder auch die letzten $\lfloor {}^2\log {}^2\log m \rfloor$ Bits aus, bis zu insgesamt $g(\lfloor {}^2\log(m) \rfloor)$ Bits. Wenn dieser Zufallsgenerator (mit m als Parameter) nicht perfekt ist, dann gibt es einen effizienten Algorithmus zum Brechen der RSA-Verschlüsselung. Der Rechenaufwand ist größer als beim BBS-Generator in dem Maße, wie das Potenzieren mit d aufwendiger als das Quadrieren ist; ist d zufällig gewählt, so ist der Zeitbedarf $\mathrm{O}(n^3 \cdot g(n))$, da das Potenzieren mit einer n-Bit-Zahl im Vergleich zum schlichten Quadrieren in jedem Schritt den Aufwand mit dem Faktor n vergrößert.

Der Index-Generator (Blum/Micali). Man wählt als Modul zufällig eine große Primzahl p und bestimmt dazu eine Primitivwurzel a. Ferner wählt man einen zufälligen Startwert $x = x_0$, teilerfremd zu $p-1$. Dann bildet man $x_i = a^{x_{i-1}} \bmod p$ und gibt das erste oder die ersten $\lfloor {}^2\log {}^2\log p \rfloor$ Bits aus, bis zu insgesamt $g(\lfloor {}^2\log(p) \rfloor)$ Bits. (Mit den letzten Bits geht's genauso.) Die Perfektheit dieses Zufallsgenerators (mit den Primzahlen als Indexmenge) beruht auf der Vermutung, dass diskrete Logarithmus mod$p$ hart ist. Auch hier ist der Zeitbedarf pro Bit $\mathrm{O}(n^3)$.

Der elliptische Index-Generator (Kaliski). Er funktioniert wie der Index-Generator, nur dass man die Gruppe $\mathbb{M}_p = \mathbb{F}_p^\times$ durch eine elliptische Kurve über dem Körper $\mathbb{F}_p$ ersetzt (eine solche Kurve ist auf kanonische Weise eine endliche Gruppe).