10 Weitere kryptographische Primitive

Secret Sharing Durchsetzen eines Mehr-Augen-Prinzips:
Aufteilen eines Geheimnisses K in n Teilgeheimnisse so, dass

- $t \leq n$ Teilgeheimnisse benötigt werden, um K zu erhalten,
- $t - 1$ Teile aber keinerlei Informationen über K liefern.

Heißt $(t, n)-$Schwellwert-Schema.

Einsatz: z.B. Sicherer Schlüsselspeicher, Zugriffskontrolle

Beispiel. Einfach aber unsicher:

$$K = (\underbrace{K_1, \ldots, K_{32}}_{=TK_1}, \underbrace{K_{33}, \ldots, K_{64}}_{=TK_2}, \underbrace{K_{65}, \ldots, K_{96}}_{=TK_3}, \underbrace{K_{97}, \ldots, K_{128}}_{=TK_4}$$

Kenntnis über 3 Teilgeheimnisse liefern 96 Bit des Schlüssels.

Beispiel. Einfach und sicher:

1. Wähle zufällig $n - 1$ Werte $TK_1, \ldots, TK_{n-1} \in \{0, 1\}^{128}$.
2. Setze $TK_n := TK_1 \oplus \cdots \oplus TK_{n-1} \oplus K$.

Es gilt:

1. $TK_1 \oplus \cdots \oplus TK_n = K$.
2. Kennt man weniger als n TG, erhält man keine Inf. über K (Erinnerung: One-time-pad)

Aber: Nur $(n, n)-$Schwellwertschema.

Shamirs Secret Sharing: Verstecke K in einem Polynom:

- Betrachte $K \in \{0,1\}^{128}$ als nat. Zahl: $K = \sum_{i=1}^{128} K_i \cdot 2^{i-1}$.
- Wähle $t-1$ Werte $a_1, \ldots, a_{t-1}$.
- Betrachte Polynom $f(x) = K + a_1 x^1 + a_2 x^2 + \cdots a_{t-1} x^{t-1}$.
  Dann gilt $f(0) = K$.
- Erzeugen der Geheimnisse:
  - Person 1 $\longleftarrow f(1)$
  - Person 2 $\longleftarrow f(2)$
  - $\vdots$
  - Person $n \longleftarrow f(n)$
- Rekonstruktion des Polynoms/des Schlüssels K:
  - Fundamentalsatz der Algebra: Polynom $(t-1)-$Grades wird durch t Punkte $(x_i, f(x_i))_{i \leq t}$ eindeutig festgelegt.
    Es werden also t der n zuvor berechneten Werte benötigt.
  - Lagrange-Formel: $f(x) = \sum_{i=1}^{t} f(x_i) \prod_{\substack{j=1 \\ j \neq i}}^{t} \frac{x - x_j}{x_i - x_j}$.
  - $K = f(0)$.
- Sicherheit:
  - $t-1$ oder weniger Werte liefern keinerlei Information über K:
  - Angreifer kennt $t-1$ Werte $(1, f(1), \ldots, t-1, f(t-1))$.
  - $K = f(0) = \sum_{i=1}^{t} f(i) \underbrace{\prod_{\substack{j=1 \\ j \neq i}}^{t} \frac{0-j}{i-j}}_{=:I_i} = \underbrace{\sum_{i=1}^{t-1} f(i) I_i}_{\text{Dem Angreifer bekannt}} + \underbrace{f(t)}_{\text{nicht bekannt}} \underbrace{I_t}_{\text{bekannt}}$.
  - $f(t)$ kann der Angreifer nicht abschätzen, also auch K nicht.

Zufallszahlengeneratoren

Einsatz für Schlüsselgenerierung, Zufall für kr. Verfahren (DSA, DH)

Wichtig: Güte des Zufalls, d.h. Entropie/Unvorhersagbarkeit
n Bit bedeutet: Wahrscheinlichkeit Zufallswert zu erraten ist $1/2^n$.

Wir unterscheiden physikalische und deterministische Generatoren

Physikalische Zufallszahlengeneratoren

Zufall aus physikalischen Rauschquellen

- Impulsschwankungen elektromagnetischer Schaltungen,
- radioaktiver Zerfall,
- Atmosphärenrauschen.

Probleme:

- nicht immer verfügbar,
- häufig sehr langsam
  Zeit, bis genug Zufall gesammelt wurde.
- häufig Schiefen (mehr 1 als 0), also keine Gleichverteilung
  Deterministische Nachbearbeitung notwendig.

Beispiel. Erzeugt wird Zufallsfolge $x_1, \ldots, x_n \in \{0, 1\}$

- Entropie pro Bit : Ideal 1, d.h. $\Pr(x_i = 1) = 1/2$
- Häufig Schiefen: Beispiel $\Pr(x_i = 0) = 0,2$. Damit $\Pr(x_i = 1) = 0,8$.
- Entropie von x_i :
  - Wahrscheinlichkeit x_i zu erraten: $0,8$.
  - Ausgedrückt in Entropie: Finde t mit $1/2^t = 0,8$?
    Also $2^t = 1,25$, d.h. $t = 0,322 \approx 1/3$.
  - Deterministische Nachbearbeitung:
    Aus 300 Bit mit Entropie 100 mache 100 Bit mit Entropie 100.

Deterministische Zufallszahlengeneratoren

Berechnen aus einem Zufallswert fester Länge, dem *Seed*,
eine pseudozufällige Bitfolge praktisch beliebiger Länge.

Achtung: Entropie kann durch det. Nachbearbeitung nicht erhöht werden.

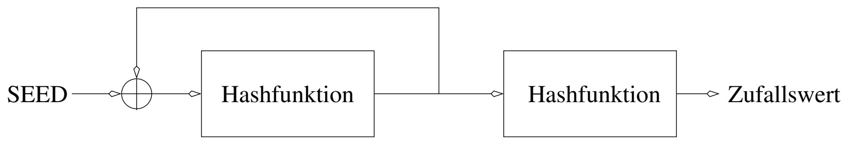

Abbildung 6: Ein einfacher Pseudozufallszahlengenerator

Weiteres Beispiel: Sichere Blockchiffre im Counter-Mode

Sicherheitforderungen:

- Entropieerhaltung,
- Aus Teil der Zufallsfolge dürfen nicht berechnet werden können
  - Vorgänger
  - Nachfolger

Übung: Zeigen Sie, dass die obigen beiden Beispiele diese umsetzen.

Seedgenerierung

Bereits gesehen: Über physikalische Zufallszahlengeneratoren.

Am Computer: Zusammensetzen verschiedener **unabhängiger** Zufallszahlen

Beispiel (GNU/Linux). Funktion /dev/random

- Untersucht vom BSI: Liefert mind. 100 Bit Entropie.
- Nutzt verschiedene Ereignisse wie Systemzeit, Nutzeraktivität usw.

Beispiel (Windows). Kombination verschiedener Systemaufrufe:

1. ReadTimeStampCounter(): Prozessorzyklen seit Systemstart
   Bei Taktfrequenz ≥ 1 GHz: 2^{30} verschiedene Werte pro Sekunde.

2. KeQuerySystemTime(): Aktuelle Systemzeit
   Auflösung 100 ns, d.h. 2^{23} verschiedene Werten pro Sekunde.

1. Starten des Rechners
2. Starten des Programms
   (a) A:=ReadTimeStampCounter(),
   (b) B:=KeQuerySystemTime(),
3. Verifizieren eines Logins
   (a) C:=ReadTimeStampCounter(),
4. Erzeugen des Seeds
   (a) D:=ReadTimeStampCounter(),
   (b) SEED:=A||B||C||D,
   (c) Entropie $30 + 23 + 30 + 30 = 113$.

Zur Entropieberechnung:

- Annahme: Angreifer kann Zeiten nur mit Unsicherheit von 1 sec raten
- Die Werte A,B,C,D sind unabhängig voneinander
  (Keine Berechnung eines Wertes bei Kenntniss der drei anderen)