Stromchiffren

Verschlüsseln eines Stroms von Daten m_i (Bits/Bytes) mithilfe eines Schlüsselstroms k_i in die Chiffretexte c_i.

Idee: Im One-Time Pad den zufälligen Schlüssel durch eine pseudo-zufällige Folge k_i ersetzen, welche vom Schlüssel k abhängt.

- Aus kurzem Schlüssel k langen Schlüssel $(\dots, k_i, \dots)$ machen.
- Keine perfekte Sicherheit.
- Komplexitätstheoretische Sicherheit, wenn pseudo-zufällige Folge nicht von einer echt zufälligen Folge effizient unterschieden werden kann.

Stromchiffren

Synchrone Stromchiffren: k_i abhängig von k, unabhängig von m_i.
- $\sigma_{i+1} = f(\sigma_i, k)$ (Zustände),
- $k_i = g(\sigma_i, k)$,
- $c_i = \mathcal{E}(k_i, m_i)$, $m_i = \mathcal{D}(k_i, c_i)$.

Selbst-synchrone Stromchiffren: k_i abhängig von k und c_i.
- $\sigma_i = (c_{i-t}, \dots, c_{i-1})$ für ein festes t,
- $k_i = g(\sigma_i, k)$,
- $c_i = \mathcal{E}(k_i, m_i)$, $m_i = \mathcal{D}(k_i, c_i)$.

Meistens $\mathcal{E}(x,y) = \mathcal{D}(x,y) = x \oplus y$.

Wichtiges Beispiel: OFB, CTR, CFB für Blockchiffren.

Stromchiffren

Eigenschaften synchroner Stromchiffren:
- Sender/Empfänger müssen synchronisiert sein.
- Keine Fehlerfortpflanzung.
- Änderung von Chiffretext c_i möglich.

Eigenschaften selbst-synchroner Stromchiffren:
- Nach Übertragungsfehler Selbstsynchronisation, wenn t konsekutive Chiffretexte richtig übertragen werden.
- Beschränkte Fehlerfortpflanzung in maximal t folgende Entschlüsselungen.
- Änderung von Chiffretext c_i leichter erkennbar als bei synchronen Stromchiffren, Einfügen/Ausschneiden von c_i schwerer erkennbar.

In beiden Fällen auf Datenauthentizität und -integrität extra achten.

Schlüsselstromerzeugung

Für synchrone Stromchiffren wird ein Schlüsselstrom benötigt. Darf nicht von einer zufälligen Bitfolge effizient unterscheidbar sein.

$\rightarrow$ Pseudozufallsbitgeneratoren.
- Nimmt kurze Eingabe (seed) und produziert lange, „zufällige“ Ausgabe.
- Statistische Tests von Algorithmen durchführbar.
- Kryptographisch sicher, wenn „alle statistischen Tests“ bestanden werden.

Konstruktionen für Schlüsselstromerzeugung:
- Linear Feedback Shift Register (LFSR, nicht sicher).
- Lineare Kongruenzgeneratoren (nicht sicher).
- Ausgabe von Blockchiffren wie in OFB, CTR (vermutlich sicher).
- ...

Lineare Feedback Shift Register

Zustands- bzw. Schlüsselstromerzeugung durch
Lineare Feedback Shift Register (LFSR):

- Betrachten σ_i aus $\mathbb{F}_2$.
- Startwerte $\sigma_0, \ldots, \sigma_{l-1}$.
- Rekursion: $\sigma_j = a_1\sigma_{j-1} + \cdots + a_l\sigma_{j-l}$.

Maximale Periode $2^l - 1$ für geeignete Wahl der a_i möglich.
($x^l + a_1x^{l-1} + \cdots + a_l$ prim und Nullstelle ist Erzeuger von $(\mathbb{F}_{2^l})^\times$.)

Lineare Komplexität einer Folge $\sigma_0, \sigma_1, \ldots$: Kleinstes l, für das die Folge durch ein LFSR mit geeigneten $a_1, \ldots, a_l$ entsteht.

LFSR's sind schnell in Hardware!

Lineare Feedback Shift Register

Die Ausgabe eines LFSR ist leicht vorherzusagen (die a_i können mit dem Berlekamp-Massey Algorithmus aus wenigen σ_i berechnet werden).

Daher nicht für kryptographische Zwecke geeignet.

Lösung (teilweise): Mehrere LFSR nicht-linear kombinieren.

Brauchen mindestens:

- Große Periode.
- Große lineare Komplexität.
- Gute statistische Eigenschaften.

Sicherheit bei der Verwendung von LFSR's dann häufig trotzdem etwas vage ...

Lineare Kongruenzgeneratoren

Lineare Kongruenzgeneratoren.

- Rechnen in $\mathbb{Z}/(m)$.
- $\sigma_{i+1} = a\sigma_i + b \bmod m$.

Eigenschaften:

- Software-geeignet.
- Viel für nicht kryptographische Anwendungen benutzt.
- Allerdings nicht kryptographisch sicher, da leicht vorhersagbar.

Ausgabe von Blockchiffren

Im OFB, CTR und CFC Mode wird der Schlüsselstrom als Ausgabe eines Blockchiffres definiert.

Die Philosophie hier ist ungefähr:
Verhält sich der Blockchiffre wie eine zufällige Permutation, so ergibt dies einen zufälligen Schlüsselstrom.

RC4

RC = Ron’s Cipher nach Ron Rivest (Mitgründer von RSA).

RC4 sehr schneller Stromchiffre:

- 1987 entwickelt, 7 Jahre geheim,
- 1994 anonym im Internet veröffentlicht,
- kommerziell (Lizenzgebühren).

In RC4 wird im folgenden speziell $m = 256$ verwendet. Klartext und Schlüsselstrom werden geXORed.

Ist weit verbreitet (Oracle SQL, Windows, SSL, IEEE 802.11 WLAN Standard, . . .)
Hat gewisse Sicherheitsschwächen (WEP in IEEE 802.11).

RC4

Alle Berechnungen modulo m:

- $i, j \in \mathbb{Z}/(m)$, $S_0, \ldots, S_{m-1} \in \mathbb{Z}/(m)$.
- Schlüssel k besteht aus $a_i \in \mathbb{Z}/(m)$ für $0 \leq i \leq l-1$.
- Sollte in der Praxis einen IV enthalten. (!)

Initialisierung:

- $S_\nu \leftarrow \nu$ für $0 \leq \nu \leq m-1$.
- $j \leftarrow 0$.
- Für $i = 0, \ldots, m-1$: $j \leftarrow j + S_i + a_{i \bmod l}$, vertausche S_i und S_j.

Neues Schlüsselstromglied berechnen:

- Zum Anfang $i \leftarrow 0,\ j \leftarrow 0$.
- Dann: $i \leftarrow i+1,\ j \leftarrow j + S_i$, S_i und S_j vertauschen, $t \leftarrow S_i + S_j$.
- Ausgabe S_t.

Integrität und Authentizität

Werden nicht durch Verschlüsselung bereitgestellt!

- Integrität Problem bei Verschlüsselung von nicht redundanten Daten (komprimierte Dateien, geheime Schlüssel).

Idee: Geeignete kryptographische Prüfsummen mitschicken.

Davon gibt es zwei Typen:

- Manipulation Detection Codes (MDC), Hashfunktionen ohne Schlüssel. Liefert Fingerprint oder Message Digest.
- Message Authentication Codes (MAC), Hashfunktionen mit Schlüssel.

Hashfunktionen haben vielfache Verwendung nicht nur bei Verschlüsselung (Unterschriften, Pseudozufallszahlen, Wörterbücher ...)

Sicherheitsmerkmale

Seien X, Y Mengen und $f : X \to Y$. Bildwerte $f(x)$ sollen mit einem Algorithmus effizient berechenbar sein.

Aufgaben:

1. (Urbild berechnen) Zu $y \in Y$ ein $x \in X$ mit $f(x) = y$ bestimmen.
2. (Zweites Urbild berechnen) Zu $x \in X$ ein $x' \in X$ mit $x' \neq x$ und $f(x) = f(x')$ bestimmen.
3. (Kollision finden) $x, x' \in X$ mit $x \neq x'$ und $f(x) = f(x')$ bestimmen.

Def: Eine Funktion heißt Einwegfunktion, wenn es keinen effizienten Algorithmus gibt, der Aufgabe 1 mit signifikanter Wahrscheinlichkeit löst.

Def: Eine Funktion heißt (schwach) kollisionsresistent, wenn es keinen effizienten Algorithmus gibt, der Aufgabe 2 (Aufgabe 3) mit signifikanter Wahrscheinlichkeit löst.

Effizient und signifikant

Effiziente Laufzeit und signifikante Erfolgswahrscheinlichkeit eines Algorithmus A kann man quantitativ oder qualitativ angeben.

Quantitativ: Für eine speziell vorgelegte Situation.

Effizient = Laufzeit $\leq 2^{40}$ Bitops, signifikant = Erfolgswahrscheinlichkeit $\geq 2^{-40}$ ...

Qualitativ: Gemessen in der Bitlänge k der Eingabe von A.

Dann effizient = Laufzeit polynomiell in k, signifikant = Erfolgswahrscheinlichkeit nicht vernachlässigbar in k.

Wenn man in der Kryptographie sagt, daß es keinen in k effizienten Angreifer A mit signifikanter Erfolgswahrscheinlichkeit gibt, dann nennt man k häufig Sicherheitsparameter.

Polynomiell und vernachlässigbar

Eine Funktion $g : \mathbb{R}^{\geq 0} \to \mathbb{R}$ heißt polynomiell (in k), wenn es ein Polynom $P \in \mathbb{R}^{\geq 0}[x]$ und $k_0 \in \mathbb{R}^{\geq 0}$ gibt, so daß $|g(k)| \leq P(k)$ für alle $k \geq k_0$.

Eine Funktion $g : \mathbb{R}^{\geq 0} \to \mathbb{R}$ heißt vernachlässigbar (in k), wenn es für jedes Polynom $Q \in \mathbb{R}^{\geq 0}[x]$ ein $k_0 \in \mathbb{R}^{\geq 0}$ gibt, so daß $|g(k)| \leq 1/Q(k)$ für alle $k \geq k_0$.

Zurück zu Einweg- und kollisionsresisten Funktionen

Um von Einweg- und kollisionsresistenten Funktionen im qualitativen Sinn sprechen zu können, betrachtet man eine Familie von Funktionen $f_k : X_k \to Y_k$ und $k \to \infty$.

Es soll einen Algorithmus geben, welcher Bildwerte $f_k(x)$ nach Eingabe von $1^k, x$ effizient berechnet.

Auf der anderen Seite soll es keinen Algorithmus geben, der Urbilder oder Kollisionen mit signifikanter Wahrscheinlichkeit unter Eingabe von k, y bzw. x' effizient berechnen kann.

Hieraus folgen beispielsweise Größenbedingungen an X_k und Y_k in Abhängigkeit von k.

Hashfunktionen

Def: Seien X, Y, Z Mengen mit $\#Z < \#Y < \infty$ und $\#X = \infty$.
Eine Hashfunktion ist eine Funktion $h : X \to Z$.
Eine Kompressionsfunktion ist eine Funktion $h : Y \to Z$.

Meistens $X = \{0,1\}^*$, $Y = \{0,1\}^m$, $Z = \{0,1\}^n$ mit $m > n$.
Sind nicht injektiv!

Wieder Forderung:

Bildwerte von Hash- und Kompressionsfunktionen sollen effizient durch Algorithmen nach Eingabe der Argumente berechnet werden können.

Zwei Anwendungen

1. Paßworte:
   - In der Paßwortdatei sind nur die Hashwerte der Paßwörter gespeichert.
   - Beim Einloggen wird der Hashwert des eingegeben Paßworts berechnet und mit dem gespeicherten verglichen.
   - Paßwortdatei soll nicht unbedingt lesegeschützt sein $\Rightarrow$ Hashfunktion soll Einwegfunktion sein.
   - Aber: Dictionary Angriffe (mögliche Paßworte ausprobieren und Hashwerte vergleichen).
2. Unterschriften:
   - Nicht das Dokument, sondern nur den Hashwert unterschreiben.
   - Angreifer soll kein zweites Dokument mit dem gleichen Hashwert berechnen können $\Rightarrow$ Hashfunktion soll kollisionsresistent sein.

Zufallsorakelmodell

Auch Random Oracle Model (RO). Ist theoretische Herangehensweise.

Hashfunktionen werden idealisiert als zufällige Funktionen modelliert.

- Ermöglicht und vereinfacht Untersuchungen und Sicherheitsbeweise.

Man kann eine zufällige Hashfunktion nicht effizient als ganzes beschreiben. Realisierung/Simulierung durch ein Orakel.

Hashsimulation durch Orakel

Bei einer Anfrage nach dem Hashwert von x geht das Orakel wie folgt vor: Das Orakel überpüft, ob $h(x)$ schon einmal erfragt und berechnet wurde, wenn

- ja, dann wird dieser Wert zurückgegeben.
- nein, dann wird ein zufälliger Wert zurückgegeben und als $h(x)$ gespeichert.

Insofern wird wirklich eine zufällige Funktion $h: X \to Z$ definiert.

Eine Orakelanfrage zählt in der Laufzeit eines Algorithmus als ein Schritt (konstante Zeit). Die Anzahl der Orakelanfragen wird üblicherweise angegeben und sollte polynomiell sein.

Angriffe im Zufallsorakelmodell

Sei $h: X \to Z$ mit $\#Z = n$.

Angriff auf Einweg-Eigenschaft von h:

- Angreifer berechnet s Hashwerte durch Anfragen an das Orakel.
- Mit Wahrscheinlichkeit $(1 - 1/n)^s$ ist Zielwert y nicht darunter.
- Mit Wahrscheinlichkeit $(1 - (1 - 1/n)^s) + (1 - 1/n)^s/n$ kann er das richtige Urbild x raten.

Angriff auf die schwache Kollisonsresistenz von h:

- Ähnlich wie bei der Einweg-Eigenschaft.

Aufgrund des Zufallsorakelmodells gibt es auch keine Strategien, die eine bessere Erfolgswahrscheinlichkeit haben würden.

Angriffe im Zufallsorakelmodell

Schreibe $n = 2^k$.

- Der Beitrag $(1-1/n)^s/n$ ist vernachlässigbar in k, unabhängig von s.
- Für einen in k polynomiellen Algorithmus muß s polynomiell sein. Wegen $(1-1/n)^s \geq 1-s/n$ ist $1-(1-1/n)^s \leq s/n$. Dies ist ebenfalls vernachlässigbar in k.
- Damit ist die Erfolgswahrscheinlichkeit vernachlässigbar, wenn s nur polynomiell ist.
- Umgekehrt sind ungefähr 2^k Orakelanfragen erforderlich, um eine konstante Erfolgswahrscheinlichkeit zu haben.

Folgerung: Eine Hashfunktion im Zufallsorakelmodell ist eine Einwegfunktion und schwach kollisionsresistent.

Geburtstagsparadoxon

Thm: Wählen wir aus $n \geq 2^{16}$ Elementen $k \geq 1.18\sqrt{n}$ zufällig mit Zurücklegen aus, so haben wir mit Wahrscheinlichkeit $> 1/2$ mindestens ein Element mindestens zweimal ausgewählt.

Bew: Sei X_i das Ereignis, daß das i-te gewählte Element nicht mit einem der vorherigen übereinstimmt. Dann gilt $\Pr(X_i \mid X_1,\ldots,X_{i-1}) = 1-(i-1)/n$ und $\Pr(X_1,\ldots,X_k) = \prod_{i=1}^k(1-(i-1)/n)$. Aus $1+x \leq e^x$ folgt $\prod_{i=1}^k(1-(i-1)/n) \leq \prod_{i=1}^k e^{-(i-1)/n} \leq e^{-k(k-1)/(2n)}$. Für das angegebene n und k gilt $e^{-k(k-1)/(2n)} < 1/2$. Somit erhalten wir eine Doppelauswahl mit Wahrscheinlichkeit $> 1/2$. $\Box$

Auf der anderen Seite ergibt sich aus $\prod_{i=1}^k(1-(i-1)/n)$ $\geq (1-k/n)^k \geq 1-k^2/n$ die obere Schranke k^2/n für die Wahrscheinlichkeit einer Doppelauswahl.

Angriffe im Zufallsorakelmodell

Angriff auf die Kollisionsresistenz von h:

- Angreifer berechnet $s \geq 1.18\sqrt{n}$ Hashwerte durch Anfragen an das Orakel.
- Befindet sich unter den Hashwerten eine Kollision, so wird diese ausgegeben.

Nach dem Geburtstagsparadoxon ist die Wahrscheinlichkeit hierfür $> 1/2$. Speicherbedarf ca. $\sqrt{n}$.

Zum Speichern der Hashwerte eine Tabelle anlegen und nach den ersten $\log_2(\sqrt{n})$ Bits der Hashwerte indizieren.

Beim Suchen direkt im entsprechenden Tabellenfeld nachschauen, ob bereits Hashwert definiert bzw. das zugehörige x eingetragen wurde.