Kryptographie

Mitschrift von www.kuertz.name

Hinweis: Dies ist **kein offizielles Script**, sondern nur eine private Mitschrift. Die Mitschriften sind teweilse **unvollständig, falsch oder inaktuell**, da sie aus dem Zeitraum 2001–2005 stammen. Falls jemand einen Fehler entdeckt, so freue ich mich dennoch über einen kurzen Hinweis per E-Mail – vielen Dank!

Klaas Ole Kürtz (klaasole@kuertz.net)

Inhaltsverzeichnis

Einführung

Ziel der Vorlesung: Wesentliches Verständnis von `PGP` und `SSH` – Unter welchen Voraussetzungen kann man diese als sicher betrachten? Was heißt überhaupt **sicher**?

Als Protagonisten wollen **Alice** und **Bob** kommunizieren, **Eve** hört immer mit und **Oscar** möchte die Kommunikation manipulieren. Es gibt zwei **Standard-Szenarien:**

1. Alice will Bob eine Nachricht so schicken, daß niemand außer Bob die Nachricht lesen kann: **Vertraulichkeit**. Bob will, daß er sicher sein kann, daß Nachrichten mit Absender „Alice“ auch wirklich von Alice kommen: **Authentizität**. Bob will außerdem sicher sein können, daß die Nachricht unterwegs nicht verändert worden ist: **Integrität**.

2. Alice und Bob haben beide einen Rechner, die gegenseitig miteinander kommunizieren wollen, d.h. daß auch mehrere Nachrichten hin- und hergeschickt werden sollen, aber auch mit vergleichbaren Forderungen nach Vertraulichkeit, Authentizität und Integrität.

Ein anderes Sicherheitsziel könnte **Anonymität** sein (beispielsweise anonymes Surfen).

Kurz zur **Begrifflichkeit:** Meist wird verwendet:

$$\mathsf{Kryptologie} = \mathsf{Kryptographie} + \mathsf{Kryptanalyse}$$

Literatur

1. Stinson: Cryptography – Theory and Practice (gute allgemeine Einführung, nicht unbedingt für Informatiker)
2. Delft, Knebl: Introduction to Cryptography (kommt am stärksten an die Vorlesung ran)
3. Buchmann: Einführung in die Kryptographie (recht mathematisch)
4. Singh: The Code Book (Bettlektüre: Geschichte der Kryptographie)
5. Ruby: Psudorandomness and Cryptographic Applications (recht algorithmisch, aber nur ein Script)
6. Handbook of Applied Cryptography (ein Handbuch eben, ausführliche Literaturhinweise, ...)
7. Mihir Bellare, Goldwasser: Skriptum zu einer Vorlesung am MIT (unter `www-cse.ucsd.edu/users/mihir/` zu finden)
8. Carl Pomerance: Prime Numbers
9. Krypto-Protokoll-Bibliothek: `www.lsv.ens-cachan.fr/spore`

1 Einmalige Verschlüsselung – Kryptosysteme

In diesem Fall kennen sich Alice und Bob vorher, Alice will eine Nachricht von begrenztem, vorher bekanntem Umfang (eine *kurze* Nachricht) über einen unsicheren (abhörbaren) Übertragungsweg (Kanal) an Bob schicken – dabei geht es nur um Vertraulichkeit, nicht um Authentizität oder Integrität. **Ziel** ist es hier,

1. Verfahren zu entwickeln, die sich im Szenario nutzen lassen
2. Sicherheit (Vertraulichkeit) vernünftig zu definieren
3. Beweisen, daß die untersuchten Verfahren sicher sind (oder auch nicht)

Grundlegende Vorgehensweise:

- Alice und Bob einigen sich vorab auf eine Transformation, die aus einer gegebenen Nachricht eine andere erstellt und eine zugehörige Umkehrtransformation erlaubt.
- Die Transformationen werden ausgewählt aus einer Schar (Familie); die Familie ist bekannt, die ausgewählte Funktion nicht und wird durch einen sogenannten **Schlüssel** definiert.

Beispiel: Alice möchte Bob ein Bit schicken. Alice und Bob benutzten entweder die Transformation $x \mapsto x$ (Schlüssel 0) oder $x \mapsto x \oplus 1$ (Schlüssel 1).

Mathematisches Modell: Ein *Kryptosystem (KS)* ist ein Tupel

$$(X, Y, K, \{e_k\}_{k\in K}, \{d_k\}_{k\in K})$$

- X ist eine endliche Menge von *Klartexte*
- Y ist eine endliche Menge von *Chiffretexte*
- K ist eine endliche Menge von *Schlüsseln*
- $\{e_k\}_{k\in K}$ ist eine Familie von *Chiffriertransformationen* $e_k \colon X \to Y$
- $\{d_k\}_{k\in K}$ ist eine Familie von *Dechiffriertransformationen* $d_k \colon Y \to X$
- Forderung: $d_k(e_k(x)) = x$ für alle $x \in X$, $k \in K$

1.1 Kombinatorische Sicherheit

Grundlegende Idee: Ein Kryptosystem ist sicher, wenn das Beobachten eines Chiffretextes zu keiner zusätzlichen Information über den gesendeten Klartext führt.

Mathematisch: Ein Kryptosystem $S = (X, Y, K, E, D)$ ist *kombinatorisch sicher*, falls gilt:

$$\forall\, y \in \bigcup_{k \in K} e_k(X)\ \ \forall\, x \in X\ \ \exists\, k \in K \colon e_k(x) = y$$

Definiere hierzu auch die *aktiven Chiffretexte* $\bar{Y} := \bigcup_{k \in K} e_k(X) \subseteq Y$.

Beispiel: : Benutze $S = (\{0,1\}, \{0,1\}, \{0,1\}, E, D)$ mit $e_k(x) = x \oplus k = d_k(x)$ für alle x, k – offensichtlich ist dieses System kombinatorisch sicher.

1.2 Perfekte (informationstheoretische) Sicherheit

Zusätzliche Annahme: Schlüssel werden entsprechend einer Wahrscheinlichkeitsverteilung ausgewählt, Klartexte ebenso, beide unabhängig voneinander. D.h. es gibt eine Wahrscheinlichkeitsverteilung P_X auf X und eine Verteilung P_K auf K. Bezeichnung: Falls S ein Kryptosystem ist, bezeichne das *Kryptosystem mit zwei Verteilungen (KSVV)* mit (S, P_X, P_K).

Bezeichne $\hat{Y} := \{y \in Y \mid P(y) > 0\}$ als *aktive Chiffretexte*, entsprechend *aktive Klartexte* $\hat{X}$ und *aktive Schlüssel* $\hat{K}$.

Es ergibt sich eine *gemeinsame Verteilung*

$$P((x,y)) = \sum_{\substack{k \in K \\ e_k(x) = y}} P_X(x) \cdot P_K(k)$$

Zudem ergibt sich eine Verteilung für Y:

$$P(y) = \sum_{x \in X} P(x,y) = \sum_{x \in X} \left(P_X(x) \cdot \left(\sum_{\substack{k \in K \\ e_k(x) = y}} P_K(k) \right) \right)$$

Ein Kryptosystem ist S mit Klartext- und Schlüsselverteilung P_X und P_K besitzt *perfekte Sicherheit*, falls für alle x und $y \in \hat{Y}$ gilt:

$$P(x) = P(x|y) = \frac{P(x,y)}{P(y)}$$

Somit sind x und y (stochastisch) unabhängig. BEISPIELE:

1. Definiere ein KSVV durch

		a	b
		$\frac{1}{4}$	$\frac{3}{4}$
K_0	$\frac{1}{3}$	0	1
K_1	$\frac{1}{3}$	1	1
K_2	$\frac{1}{3}$	1	0

   Nun ergibt sich für $x = a$, $y = 0$ zunächst $P(x) = \frac{1}{4}$, weiter ist

$$\begin{aligned} P(x,y) &= \frac{1}{4} \cdot \frac{1}{3} = \frac{1}{12} \\ P(y) &= \frac{1}{4} \cdot \frac{1}{3} + \frac{3}{4} \cdot \frac{1}{3} = \frac{1}{3} \\ P(x) \cdot P(y) &= \frac{1}{4} \cdot \frac{1}{3} = \frac{1}{12} = P(x,y) \end{aligned}$$

   Für $x = a$, $y = 1$ ergibt sich $P(x) = \frac{1}{4}$ und $P(x,y) = \frac{1}{6}$ sowie

$$\begin{aligned} P(y) &= 1 - P(y = 0) = \frac{2}{3} \\ P(x) \cdot P(y) &= \frac{1}{4} \cdot \frac{2}{3} = \frac{1}{6} = P(x,y) \end{aligned}$$

   Für $x = b$, $y = 0$ und $y = 1$ gilt entsprechend $P(x) \cdot P(y) = P(x,y)$.

2. **Vernam'scher one-time-pad**: Für jedes $l \in \mathbb{N}$ gibt es ein Kryptosystem gegeben durch

$$\left(\{0,1\}^l, \{0,1\}^l, \{0,1\}^l, (x,k) \mapsto x \oplus k, (x,k) \mapsto x \oplus k\right)$$

   Untersuche mehrere Varianten:

   (a) Die Verteilung auf den Texten P_X sei beliebig, die Verteilung P_K auf den Schlüsseln sei die Gleichverteilung, d.h. $P(k) = 2^{-l}$. Es

gilt:

$$\begin{aligned} P(x,y) &= \sum_{\substack{k\in K \\ e_k(x)=y}} P_X(x)\cdot P_K(k) = \frac{1}{2^l}\cdot P_X(x)\cdot \sum_{x\oplus k=y} 1 = \frac{1}{2^l}P_X(x) \\ P(y) &= \sum_{x\in X} P(x,y) = \sum_{x\in X}\frac{1}{2^l}P_X(x) = \frac{1}{2^l}\cdot\sum_{x\in X} P_X(x) = \frac{1}{2^l} \\ P_X(x)P(y) &= P_X(x)\frac{1}{2^l} = P(x,y) \end{aligned}$$

Dies führt zum ersten Satz unten.

(b) Sei nun $P(x) = \frac{1}{2^l}$ für alle $x \in X$, aber $P(k) \neq \frac{1}{2^l}$ für ein $k^\star$. Nun ist

$$\begin{aligned} P(y) &= \sum_{x,k\colon e_k(x)=y} P(x)P(k) \\ &= \frac{1}{2^l}\sum_{x,k\colon x\oplus k=y} P(k) \\ &= \frac{1}{2^l}\sum_k \left(P(k)\cdot\left(\sum_{x=y\oplus k} 1\right)\right) \\ &= \frac{1}{2^l}\sum_k P(k) = \frac{1}{2^l} \\ P(x|y) &= \frac{P(x,y)}{P(y)} = \frac{\frac{1}{2^l}P_K(y\oplus k)}{\frac{1}{2^l}} = P_K(y\oplus k) \end{aligned}$$

Es gilt jedoch nicht $P_K(y \oplus k) = P(x) = \frac{1}{2^l}$ für alle x: Wähle $x = 0\ldots0$ und $y = k_0$. Dann gilt $P_K(x \oplus y) \neq \frac{1}{2^l} = P(x)$.

(c) Für $P(x)$ beliebig verteilt und $P(k) \neq \frac{1}{2^l}$ für ein $k^\star$ gilt:

$$\begin{aligned} P(x,y) &= P_X(x)\cdot P_K(y\oplus x) \\ P(y) &= \sum_x P(x,y) = \sum_x P_X(x)P_K(y\oplus x) \end{aligned}$$

Damit nun die Gleichung $P(x) = P(x|y)$ erfüllt ist, müßte für ein konkrete x, y gelten:

$$P_K(y\oplus x) = \sum_k P_X(y\oplus k)P_K(k)$$

Dabei ist jedoch die linke Seite für festes y nach Voraussetzung unterschiedlich, die rechte Seite für ein festes y jedoch fest. Damit

ist keine perfekte Sicherheit gegeben. Dies erlaubt den zweiten Satz unten.

SATZ: Der one-time-pad zusammen mit einer beliebigen Klartextverteilung und der Gleichverteilung als Schlüsselverteilung bietet perfekte Sicherheit.

Hieraus entsteht eine neue Definition:

DEFINITION: Ein *Kryptosystem mit Schlüsselverteilung (KSV)* (S, P_K) ist ein Kryptosystem mit zwei Verteilungen (S, P_X, P_K), wobei P_X beliebig ist.

SATZ: Der one-time-pad zusammen mit einer beliebigen Klartextverteilung und einer von der Gleichverteilung abweichenden Schlüsselverteilung bietet *keine* perfekte Sicherheit.

1.3 Die Sätze von Shannon

Allgemeiner ist der Satz von SHANNON:

SATZ:

a) Für jedes Kryptosystem mit Schlüsselverteilung (S, P_K) und Klartext- und Schlüsselräumen X und K, welches perfekte Sicherheit bietet, gilt: $|X| \leq |K|$.

b) Für den Fall $|X| = |Y| = |K|$ sind die beiden folgenden Aussagen äquivalent für ein KSVV (S, P_X, P_K):

   (A) (S, P_X, P_K) bietet perfekte Sicherheit

   (B) $P_K(k) = \frac{1}{|K|}$ für alle $k \in K$, und für alle x, y existiert k mit $e_k(x) = y$.

   (C) (S, P'_X, P_K) besitzt perfekte Sicherheit für eine andere Klartextverteilung P'_X mit $P'_X(x) > 0$ für alle $x \in X$.

LEMMA: Für jedes sichere KSVV (S, P_X, P_K) mit $P_X(x) > 0$ für alle $x \in X$ gilt:

$$|X| \leq |\{k \in K \mid P_K(k) > 0\}| \leq |K|$$

BEWEIS: durch Widerspruch. Wir setzen $K' = \{k \in K \mid P_K(k) > 0\}$. Angenommen, $|K'| < |X|$. Wir setzen $Y_k = \{e_k(x) \mid x \in X\}$. Dann muß wegen der Dechiffriereigenschaft gelten: $|Y_k| = |X|$.

Für jedes $x \in X$ setzen wir $Y_x = \{e_k(x) \mid k \in K'\}$. Dann gilt $|Y_x| < |X|$, da $|K'| < |X|$ ist. Fallunterscheidung:

- Würde $Y_x = Y_{x'}$ für alle $x, x' \in X$ gelten, so hätten wir

$$|Y_k| \leq \left| \bigcup_{x \in X} Y_x \right| < |X| \ \forall\, k \in K'$$

  Insbesondere auch $|Y_k| < |X|$ für alle $k \in K'$, ein Widerspruch!

- Also gibt es $x, x' \in X$ mit $Y_x \neq Y_{x'}$, also o.B.d.A. ein $y \in Y_x \setminus Y_{x'}$. Also ist $P(x, y) \neq 0 = P(x', y)$. Nach perfekter Sicherheit gilt $P(x') \cdot P(y) = P(x', y) = 0$, andererseits aber $P(y) > 0$, also ist $P(x') = 0$, Widerspruch zur Voraussetzung! ■

LEMMA: Sei (S, P_X, P_K) ein KSVV. Dann bietet (S, P_X, P_K) genau dann perfekte Sicherheit, wenn für alle $y \in Y$ und $x \in \hat{X}$ gilt: $P^x(y) = P(y)$.

BEWEIS:

„⇒“ Dann gilt $P(x, y) = P(x)P(y)$ für alle x, y. Also ist $P(x)P(y) = P^x(y)P(x)$. Für $x \in \hat{X}$ ergibt sich $P^x(y) = P(y)$.

„⇐“ Dann gilt $P(x, y) = P^x(y)P(x) = P(y)P(x)$ für alle $x \in \hat{X}$. Für $x \notin \hat{X}$ gilt aber ohnehin $P(x, y) = 0 = P(x)P(y)$, daraus folgt die Behauptung. ■

FOLGERUNG: Sei (S, P_X, P_K) ein KSVV. Dann sind äquivalent:

- (S, P_X, P_K) ist informationstheoretisch sicher
- (S, P'_X, P_K) ist informationstheoretisch sicher für alle P'_x mit aktiven Schlüsseln $\subseteq \hat{X}$
- (S, P'_X, P_K) ist informationstheoretisch sicher für alle P'_x mit aktiven Schlüsseln $\supseteq \hat{X}$

DEFINITION: Ein Kryptosystem mit Schlüsselverteilung (S, P_K) bietet perfekte Sicherheit, falls (S, P_X, P_K) informationstheoretisch sicher ist für alle P_X (mit $\hat{X} = X$).

SATZ: Sei (S, P_X, P_K) ein KSVV mit $X = \hat{X}$ und $|X| = |K|$. Dann ist (S, P_X, P_Y) informationstheoretisch sicher genau dann, wenn S kombinatorisch sicher ist und $P_K(k) = \frac{1}{|K|}$ für alle $k \in K$

BEWEIS:

„⇒“ a) S ist kombinatorisch sicher. Sei $x \in X$ und $y \in \bar{Y}$. Dann existiert $x' \in X$ und $k \in K$ mit $e_k(x') = y$. Es gilt aber $P(x), P(x') > 0$ wegen $\hat{X} = X$. Aus der Sicherheit folgt dann $P(x|y) = P(x) > 0$, d.h. es existiert k mit $e_k(x) = y$; und ebenso $P(x'|y) = P(x') > 0$.

b) Sei $y_0 \in \bar{Y}$. Dann gibt es zu jedem $x \in X$ mindestens einen Schlüssel k_x mit $e_{k_x}(x) = y_0$ (wegen *a)* oben). Wegen $|X| = |K|$ gibt es dann aber *genau* einen Schlüssel. Für diesen gilt dann: $P(x)P(y_0) = P(x, y_0) = P(x) \cdot P(k_x)$. Nun ergibt sich $P(y_0) = P(k_x)$. Damit ist $P(k_x) = P(y_0) = P(k_{x'})$ für alle $x' \in X$, und wegen $|X| = |K|$ gilt auch $\{k_x \mid x \in X\} = K$, insgesamt also $P_K(k) = \frac{1}{|K|}$.

„⇐“ Es gibt für alle $x \in X, y \in \bar{Y}$ ein $k_{x,y}$ mit $e_{k_{x,y}}(x) = y$. Wegen $|X| = |K|$ ist $k_{x,y}$ eindeutig. Also gilt für $y \in \bar{Y}$:

$$P(y) = \sum_{x \in X} P(X)P(k_{x,y}) = \frac{1}{|K|} \sum_{x \in X} P(x) = \frac{1}{|K|}$$

Andererseits gilt:

$$P^x(y) = \sum_{\substack{k \in K \\ e_k(x) = y}} P(k) = P(k_{x,y}) = \frac{1}{|K|}$$

Also ist $P^x(y) = P(y)$.

■

1.4 Konkrete Kryptosysteme

- **one-time-pad** wie oben
- **Verschiebechiffren:** Sei $n > 0$. Dann ist $(\mathbb{Z}_n, \mathbb{Z}_n, \mathbb{Z}_n, E, D)$ mit $e_k(x) = x + k \bmod n$ und $d_k(y) = y - k \bmod n$. Cäsar hat dieses System angeblich mit $k = 3$ auf dem Alphabet benutzt. Dieses System bietet perfekte Geheimhaltung bei Gleichverteilung auf den Schlüsseln.
- **Affine Chiffren:** Definiere zunächst $\mathbb{Z}_n = \{0, \ldots, n-1\}$ und $\mathbb{Z}_n^* = E(\mathbb{Z}_n) = \{i < n \mid \mathrm{ggT}(i, n) = 1\}$ (d.h. die Elemente von $\mathbb{Z}_n$, die ein multiplikativ inverses besitzen, d.h. Einheit von $\mathbb{Z}_n$ sind). Es gilt $|\mathbb{Z}_n^*| = \varphi(n)$, dabei gilt für Primzahlen $p_0, \ldots, p_r$:

$$\varphi(p_0^{n_0} \cdot \ldots \cdot p_r^{n_r}) = p_0^{n_0 - 1} \cdot (p_0 - 1) \cdot \ldots \cdot p_r^{n_r - 1}(p_r - 1)$$

Beispiel: für $n = 12$ ist $\mathbb{Z}_{12}^* = \{1, 5, 6, 11\}$ und $\varphi(12) = \varphi(2^2 \cdot 3) = 2^{2-1} \cdot (2-1) \cdot 3^{1-1} \cdot (3-1) = 4$. Der **Euklidische Algorithmus** im Beispiel: Rechne in $\mathbb{Z}_{44}$, bestimme das multiplikativ inverse zu 35:

$$\begin{array}{ccccc} 44 & = & \underline{35} & \cdot 1+ & \underline{9} \\ & \swarrow & & \swarrow & \\ \underline{35} & = & \underline{9} & \cdot 3+ & \underline{8} \\ & \swarrow & & \swarrow & \\ \underline{9} & = & \underline{8} & \cdot 1+ & \underline{1} \\ & \swarrow & & \swarrow & \\ \underline{8} & = & \underline{1} & \cdot 8+ & 0 \end{array}$$

$$\begin{array}{rrcl} \Longrightarrow & 1 & = & 9 - 8 \cdot 1 \\ & & = & 9 - (35 - 9 \cdot 3) \\ & & = & 9 - (35 - (44 - 35 \cdot 1) \cdot 3) \\ & & = & -5 \cdot 35 + 4 \cdot 44 \end{array}$$

Das multiplikativ Inverse zu 35 ist also $-5 = 39$. Affine Chiffren sind nun $(\mathbb{Z}_n, \mathbb{Z}_n, \mathbb{Z}_n^* \times \mathbb{Z}_n, E, D)$ mit

$$\begin{array}{rcl} e_{(a,b)}(x) & = & a \cdot x + b \bmod n \\ d_{(a,b)}(y) & = & (y - b) \cdot a^{-1} \bmod n \end{array}$$

Dabei ist a^{-1} das multiplikativ Inverse von a in $\mathbb{Z}_n$.

▷ Die bisherigen drei Systeme sind *monoalphabetische* Chiffren, d.h. eine Position im Klartext korrespondiert mit einer Position im Chiffretext. *Polyalphabetisch* bedeutet hingegen, daß der Klartext blockweise verschlüsselt wird.

- **Kryptosystem der polyalphabetischen Substitutionen** über X mit $m > 0$: Für ein Kryptosystem $(X^m, X^m, \mathrm{Perm}(X)^m, E, D)$ mit $\mathrm{Perm}(X)$ als die Menge der Permutationen auf X,

$$e_{(\pi_0,\ldots,\pi_{m-1})}(x_0, \ldots, x_{m-1}) = (\pi_0(x_0), \ldots, \pi_{m-1}(x_{m-1}))$$

- **Vigenère-Chiffre**: Sei $n > 0, m > 0$. Verwende dann $(\mathbb{Z}_n^m, \mathbb{Z}_n^m, \mathbb{Z}_n^m, E, D)$ mit

$$e_{(k_0,\ldots,k_{m-1})}(x_0, \ldots, x_{m-1}) = (x_0 + k_0 \bmod n, \ldots, x_{m-1} + k_{m-1} \bmod n)$$

1.5 Abhörangriffe auf monoalphabetische Chiffren

Betrachte eine buchstabenweise (blockweise) Verschlüsselung einer Folge von Buchstaben $x_0x_1 \dots x_{n-1}$, für die immer derselbe Schlüssel verwendet wird *(ECB-Verschlüsselung, electronic code book mode)*: $e_k(x_0)e_k(x_1) \dots e_k(y_{n-1})$.

Buchstabenweise Verschlüsselung ist aber **schlecht!**

Zu einem Krytosystem $S = (X, Y, K, E, D)$ sei dessen *Blockquadrat* das Kryptosystem $S^{[2]}$ definiert durch $S^{[2]} = (X^2, Y^2, K, E', D')$ mit

$$e'_k(x_1, x_2) = (e_k(x_1), e_k(x_2)) \text{ und } d'_k(x_1, x_2) = (d_k(x_1), d_k(x_2))$$

SATZ: Sei (S, P_K) ein informationstheoretisch sicherers KSV mit $|X| \geq 2$. Dann ist $(S^{[2]}, P_K)$ nicht sicher.

BEWEIS: Seien $x_0 \neq x_1$ zwei Klartexte. Wir betrachten $P^{(x_0,x_0)}$ und $P^{(x_0,x_1)}$. Es gilt $P^{(x_0,x_0)}(y_0, y_1) > 0$ genau dann, wenn $y_0 = y_1$ ist. Andererseits gilt $P^{(x_0,x_1)}(y_0, y_1) > 0$ nur dann, wenn $y_0 \neq y_1$. Damit gilt $P^{(x_0,x_0)} \neq P^{(x_0,x_1)}$. ■

Im folgenden wird angenommen, daß natürliche Sprache buchstabenweise verschlüsselt wird – **Ziel** ist es, konkrete Chiffren berechnen. Die Verfahren für Angriffe sind statistischer Natur: Wir untersuchen Häufigkeiten, d.h. wir nutzen aus, daß die natürliche Sprache gewissen „Regelmäßigkeiten“ unterliegt.

Ausgangspunkt sind daher meist **Häufigkeitstabellen** für Buchstaben, Buchstabenpaare (Digramme) und Buchstabentripel (Trigramme). Als Alphabet wird daher hier zunächst *a, . . . , z* verwendet, häufig auch Zahlen $0, \dots, 25$ – aber keine Sonderzeichen.

1. **Verschiebechiffren**: Verschlüsselung sei $e_k(x) = x + k \bmod 26$. Bestimme nun den häufigsten Buchstaben y im Chiffretext. Vermutung: $e_k(\mathtt{e}) = y$, d.h. $k = y - 4 \bmod 26$

2. **Affine Chiffren**: Verschlüsselung ist $e_{(a,b)}(x) = ax + b \bmod 26$. Durch Häufigkeitsanalyse versucht man, x_0, x_1, y_0, y_1 zu ermitteln mit $e_{(a,b)}(x_0) = y_0$ und $e_{(a,b)}(x_1) = y_1$. Dann löst man das lineare Gleichungssystem

$$\begin{aligned} ax_0 + b &= y_0 \bmod 26 \\ ax_1 + b &= y_1 \bmod 26 \end{aligned}$$

3. **Polyalphabetische Chiffren**: Es wird blockweise derselbe Schlüssel benutzt, System ist $(X^m, X^m, \mathrm{Perm}(X)^m, E, D)$ mit

$$e_{(\pi_0,\ldots,\pi_{m-1})}((x_0,\ldots,x_1)) = (\pi_0(x_0),\ldots,\pi_{m-1}(x_{m-1}))$$

Bemerkung: Wir nehmen (abweichend von unserem bisherigen Prinzip) an, daß m unbekannt ist. Erstes Ziel ist daher, die Periode m zu bestimmen. Dazu gibt es zwei gängige Methoden:

- **Kasiski[1]-Test**: Man analysiert die Häufigkeiten von Trigrammen im verschlüsselten Text und benutzt folgende Annahme: Wenn zwei gleiche Trigramme im Chiffretext vorkommen, dann ist deren Abstand ein Vielfaches der Blocklänge.

  Plausibilitätsbetrachtung: Dazu müssen Trigramme an unterschiedlichen Stellen ($\mathrm{mod} m$) unterschiedlich verschlüsselt werden. Dies ist realistisch, wenn m relativ klein ist, dann von den 26^3 möglichen Trigrammen treten nur wenige tatsächlich auf.

- **Koinzidenzindex**: Wir zählen Buchstabenhäufigkeiten in den einzelnen Positionen modulo verschiedene m'. Wenn wir $m' = m$ wählen, sollte die Verteilung so aussehen, wie bei der natürlichen Sprache. Ansonsten solle sie „flacher“ sein.

- **Fridman**: Man betrachte die Wahrscheinlichkeiten/Häufigkeiten, daß an zwei unterschiedlichen, zufällig gewählten Stellen derselbe Buchstabe auftritt. Diese Wahrscheinlichkeit heißt *Koinzidenzindex*: Sei w ein Wort der Länge $r > 0$ über einer Menge X und $f_x = |\{i < r \mid w(i) = x\}|$ für jedes $x \in X$. Der Koinzidenzindex von w ist gegeben durch

$$I_C(w) = \frac{\sum_{x\in X} \cdot \binom{f_x}{2}}{\binom{r}{2}}$$

  Beispielsweise ist der Koinzidenzindex der natürlichen deutschen Sprache 0.762, für das Wort `AFFE` ist der Koinzidenzindex gegeben durch

$$I_C(\text{„}\texttt{AFFE}\text{“}) = \frac{\binom{1}{2} + \binom{2}{2}\ \binom{1}{2}}{\binom{4}{2}} = \frac{1}{6}$$

  Beobachtungen: Es gilt $I_C(x) = I_C(y)$, wenn y aus x durch monoalphabetische Chiffre entsteht. Für polyalphabetische Chiffren: Wir bestimmen den Koinzidenzindex für die m'-Scheiben $m' = 1, 2, \ldots$.

[1] Kasiski starb 1881

– Falls $m' = m$ ist, so sollte der Koinzidenzindex mit dem der natürlichen Sprache übereinstimmen.
– Falls $m' \neq m$ ist, so sollte der Koinzidenzindex kleiner sein als der der natürlichen Sprache.

2 Frische Verschlüsselung – Blockkryptosysteme

Neues Szenario: Alice möchte Bob in Zukunft **mehrere unterschiedliche** Nachrichten fester, kurzer Länge über einen unsicheren Übertragungsweg zukommen lassen. Eve hat vorübergehend Zugriff auf Alices Verschlüsselungssystem und die Möglichkeit, (wenige) selbstgewählte Nachrichten zu verschlüsseln.

- **Verschiebechiffren**: Eine Anfrage genügt, um den Schlüssel zu bestimmen.
- **Affine Chiffren** sind auch schlecht, hier werden zwei Anfragen benötigt.
- Bei **Vigenère-Chiffren** beöntigt man m Anfragen.
- **Substitutionschiffren** $(X, X, \mathrm{Perm}(X), E, D)$ mit $e_\pi(x) = \pi(x)$: sind in diesem Zusammenhang sicher! Kombinatorische Sicherheit muß hier bedeuten: Wenn Eve sich $x_0, \ldots, x_{n-1}$ hat verschlüsseln lassen und danach y sieht, kann sie nicht mehr sagen, als daß $y \in X \setminus \{x_0, \ldots, x_{n-1}\}$ kommt. Wenn jedoch gilt: (X, X, Π, E, D) mit $\Pi \subsetneq \mathrm{Perm}(X)$, so ist das System nicht kombinatorisch sicher: Sei $\pi \in \mathrm{Perm}(X) \setminus \Pi$. Sei nun $X' \subseteq X$ maximal mit der Eigenschaft, daß es $\pi' \in \Pi$ gibt mit $\pi'|_{X'} = \pi|_{X'}$. Dann gilt: $|X'| \leq |X| - 2$. Sei $X' = \{x_0, \ldots, x_{r-1}\}$ und $x \in X \setminus X'$ sowie $y = \pi(x)$. Sei weiter $\pi' \in \Pi$ mit $\pi'|_{X'} = \pi|_X$. Dann gibt es ein $x' \in X \setminus X'$ (wobei $x' \neq x$) mit $\pi'(x') = y$. Wenn Eve nach Verschlüsselung von $x_0, \ldots, x_{r-1}$ den Chiffretext y sieht, kann sie ausschließen, daß x der zugehörige Klartext ist.

Fazit: Schon kombinatorische Sicherheit ist nur mit riesigen Schlüsselräumen zu erreichen!

2.1 AES – Advanced Encryption Standard

Ab jetzt werden wir *Blockkryptosysteme* verwenden, die von Form

$$(\{0,1\}^k, \{0,1\}^l, \{0,1\}^m, E)$$

sind. AES ist ein konkretes 128-Blockkryptosystem, bei der die Verschlüsselung von 128 Bits in elf Runden erfolgt: Die erste ist einfach, die zweite bis zehnte Runde sind gleich, die elfte Runde sind verkürzt.

Kurzform:

```
AES(x, k)
---------------------------------
x := x XOR Key(k,0);
for i = 1 to 9
    x := Round(x, Key(k, i))
x := ShortRound(x, Key(k, 11))
```

Zu den Einzelteilen:

- Die Funktion `Round(x, r)` ist definiert durch:

  ```
  Round(x, r)
  ---------------------------------
  x := Sub(x);
  x := Perm(x);
  x := Mix(x);
  x := x XOR r;
  ```

- Bei der Variation `ShortRound(x, r)` fehlt der `Mix`-Schritt.

- Gerechnet wird in $GF(2^8)$, die Elemente sind Polynome über $\mathbb{Z}_2$ vom Grad < 8. Addition, Multiplikation etc. wird modulo $x^8+x^4+x^3+x+1$ gerechnet. Genauer: Wir rechnen in $\mathbb{Z}_2[x]/(x^8+x^4+x^3+x+1)$ und identifizieren $b_7 \ldots b_0$ mit $b_7x^7+\ldots+b_1x+b_0$. Dann ist die Addition/Subtraktion bitweises $\oplus$, die Multiplikation ist die Polynommultiplikation und dann Teilen mit Rest. Das Inverse findet man durch den erweiterten Euklidischen Algorithmus.

  Beispiel: 00000101^{-1} ist $(x^2+1)^{-1}$, Euklidischer Algorithmus liefert:

  $$x^8+x^4+x^3+x+1 \;=\; (x^2+1)(x^6+x^4+x)+1$$

  Damit ist $00000101^{-1} = 01010010$.

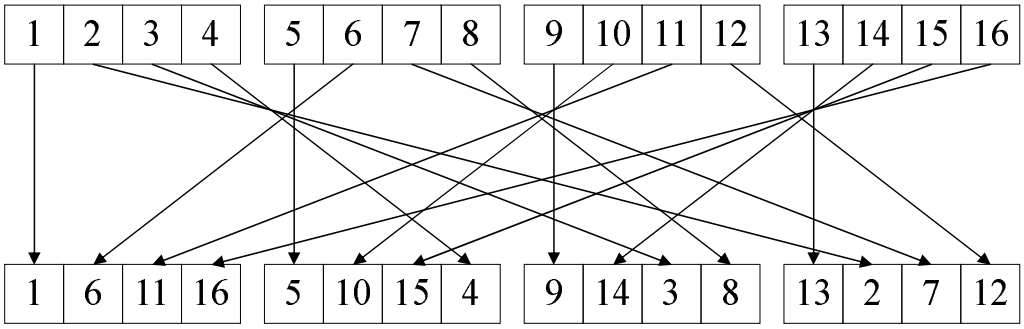
- Sub führt auf jedem Byte x der 16-Byte Eingabe folgende Operation durch:

$$\mathrm{s}(x) = x^{-1} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \oplus 01100011$$

Dabei wird $0^{-1} = 0$ angenommen. BEISPIEL:

$$\begin{aligned} \text{Sub}(00000101) &= 01010010 \cdot [\quad] \oplus 01100011 \\ &= 00001000 \oplus 01100011 = 01101011 \end{aligned}$$

- Die Operation Perm permutiert die 16 Bytes gemäß folgender Zeichnung:



- Der Schritt Mix wendet auch jeden 4-Byte-Block x der Eingabe

$$M : x \mapsto x \cdot \begin{bmatrix} 02 & 01 & 01 & 03 \\ 03 & 02 & 01 & 01 \\ 01 & 03 & 02 & 01 \\ 01 & 01 & 03 & 02 \end{bmatrix}$$

BEISPIEL: 01010100|01110000|11110100|11111111 entspricht $x^6 + x^4 + x^2$, $x^6 + x^5 + x^4$, $x^7 + x^6 + x^5 + x^4 + x^2 + x$

$$\begin{array}{rrcr} & (x^6 + x^4 + x^2) & \cdot & x \\ + & x^6 + x^5 + x^4 & \cdot & (x+1) \\ + & x^7 + x^6 + x^5 + x^4 + x^2 + x & \cdot & 1 \\ + & x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 & \cdot & 1 \\ \hline = & & & x^5 + x^4 + 1 \end{array}$$

Damit ist beginnt das Ergebnis von Mix mit 00110001

- Erzeugung der Rundenschlüssel: Erster Rundenschlüssel $\text{Key}(k, 0) = k$, dann wird $\text{Key}(k, i+1)$ aus $\text{Key}(k, i)$ gemäß Zeichnung gewonnen.

Entschlüsselung:

1. Reihenfolge der Operationen umkehren und Operationen invertieren
2. gleichen Algorithmus mit anderer Runden-Schlüsselerzeugung

2.2 Zufallsgesteuerte Algorithmen

Ein *zufallsgesteuerter Algorithmus* ist ein normaler Algorithmus, der auch die Operation `flip()` benutzen darf – dabei liefert `flip()` zufällig gleichverteilt ein Bit in konstanter Zeit. Erweiterungen: `flip(n)` liefert n voneinander unabhängige zufällige Bits, und für Bitmuster x sei `flip(x)` definiert als `flip(|x|)`.

Zur Notation: Ein normaler Algorithmus wird notiert beispielsweise durch

$$A(x_0 : t_0, \ldots, x_{n-1} : t_{n-1}) : t_n$$

für einen Algorithmus A mit n Parametern $x_0, \ldots, x_{n-1}$ vom Typ $t_0, \ldots, t_{n-1}$; der Typ des Rückgabewertes ist t_n. Der Wert der Funktion für Paramter $a_0, \ldots, a_n$ sei notiert als $A(a_0, \ldots, a_n)$. Für zufallsgesteuerte Algorithmen bezeichnet nun $A\langle\alpha\rangle(a_0, \ldots, a_n)$ den Rückgabewert eines zufälligen Algorithmus, wenn die Zufallsbits aus der unendlichen Bitfolge α entnommen werden.

Beispiele: :

- Gegeben sei:

  $A() : \{0, 1\}$
  ```
  return flip();
  ```

  Dann ist $A\langle 0110\ldots\rangle() = 0$ und $A\langle 1100\ldots\rangle() = 1$.

- Gegeben sei das Programm

  $A()$
  ```
  while flip() == 0
    skip;
  ```

  Dieser hat die Laufzeit $t(A\langle 00011\ldots\rangle()) = 7$, aber $A\langle 0000\ldots\rangle()$ terminiert nicht! Die erwartete Laufzeit ist $\frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 3 + \frac{1}{8}\cdot 5 + \ldots$, insgesamt $\sum_{i>0} 2^{-i}(2i-1) < \infty$, also endlich.

Zu Laufzeitbegriffen:

- *Algorithmen mit polynomieller Laufzeit* terminieren in allen Fällen nach polynomiell vielen Schritten.
- Auch *zufallsgesteuerte Algorithmen mit polynomieller Laufzeit* terminieren in allen Fällen nach polynomiell vielen Schritten.
- *Zufallsgesteuerte Algorithmen mit erwarteter polynomieller Laufzeit* haben einen polinomiell beschreibbaren Erwartungswert der Laufzeit.

2.3 Unterscheider und algorithmische Sicherheit

Der Begriff *sicher* bedeutet hier, daß es nicht möglich ist, in kurzer Zeit mit nicht verschwindender Wahrscheinlichkeit beim Brechen Erfolg zu haben.

Ansatz: Ein BKS ist sicher, wenn es sich nur schwer unterscheiden läßt von Substitutionschiffren. Ein *Unterscheider* ist nun ein Programm U, das eine injektive Chiffrefunktion $f\colon \{0,1\}^l \to \{0,1\}^l$ erhält, diese für beliebig viele Klartexte zum Verschlüsseln benutzen kann und dann entscheidet, ob die gegebene Funktion zum Kryptosystem gehört oder nicht. Man unterscheidet nun

real	**random**
$f = e_k$ für ein zufälliges $k \in K$	zufällige injektive Funktion $f\colon \{0,1\}^l \to \{0,1\}^l$

DEFINITION: Ein (l, L)-*Unterscheider* ist ein zufälliger Algorithmus

$$U(\{0,1\}^l \to \{0,1\}^l)\colon \{\mathit{real}, \mathit{random}\}$$

Die Wahrscheinlichkeit eines *Mißerfolgs* von U ist nun

$$\mathrm{fail}(U) = P\left(\left\{ U(f) = \mathit{real} \mid f \longleftarrow_R \mathrm{Inj}(\{0,1\}^l \to \{0,1\}^l)\right\}\right).$$

Bezüglich eines (l, L, m)-Blockkryptosystems S ist die Wahrscheinlichkeit eines *Erfolgs* von U

$$\mathrm{suc}(U, S) = P(\{ U(e_k(\cdot)) = \mathit{real} \mid k \longleftarrow_R \{0,1\}^m\}).$$

Der *Vorteil* von U ist nun die Differenz

$$\mathrm{adv}(U, S) = \mathrm{suc}(U, S) - \mathrm{fail}(U).$$

BEISPIELE:

1. S sei der one-time-pad der Länge L. Betrachte folgenden Unterscheider:

U = one-time-pad-Distinguisher($f\colon \{0,1\}^l \to \{0,1\}^l$)

```
x_0 = 0^l;
if x_0 == f(f(x_0))
  return real;
else
  return random;
```

Dann ist

$$\begin{aligned} fail(U) &= P\left(\left\{U(f) = real \mid f \longleftarrow_R \mathrm{Inj}(\{0,1\}^l \to \{0,1\}^l)\right\}\right) \\ &= P\left(\left\{U(f) = real \mid f(0^l) = 0^l\right\}\right) \cdot \underbrace{P(f(0^l) = 0^l)}_{=2^{-l}} \\ &\quad + P\left(\left\{U(f) = real \mid f(0^l) \neq 0^l\right\}\right) \cdot \underbrace{P(f(0^l) \neq 0^l)}_{\frac{1}{2^l-1}} \\ &= 2^{1-l} \\ suc(U,S) &= P(\{U(e_k(\cdot)) = real \mid k \longleftarrow_R \{0,1\}^m\}) = 1 \end{aligned}$$

Also ist $adv(U,S) = 1 - 2^{1-l}$ – mit linearer Laufzeit und nur zwei Anfragen an die Funktion f!

2. Sei ein (l,l)-Blockkryptosystem S gegeben und ein zufallsgesteuerter Algorithmus

$$V(f : \{0,1\}^l \to \{0,1\}^l, y\colon \{0,1\}^l) : \{0,1\}$$

Die Wahrscheinlichkeit dafür, daß bei zufälliger Wahl von k $V(e_k(\cdot), e_k(x))$ für ein zufälliges x, das vorher nicht abgefragt wurde, V die Antwort $x(0)$ liefert, ist 90%; der Algorithmus macht zudem nur zehn Abfragen.

$U =$ FirstBit-Distinguisher$(f\colon \{0,1\}^l \to \{0,1\}^l)$

```
x = flip(l);
b = V(f, f(x));
if b == x(0)
  return real;
else
  return random;
```

Nun gilt:

$$\begin{aligned} \mathrm{suc}(U,S) &\geq 0.9 \cdot \frac{2^l - 10}{2^l} \\ \mathrm{fail}(U) &\leq \frac{2^{l-1}}{2^l - 10} \\ \mathrm{adv}(U,S) &\geq 0.9 \cdot \frac{2^l - 10}{2^l} - \frac{2^{l-1}}{2^l - 10} \approx 0.45 \end{aligned}$$

DEFINITION: Seien $q, t, l, L \in \mathbb{N}$. Ein (q, t, l, L)-Unterscheider ist ein (l, L)-Unterscheider, dessen Laufzeit $\leq t$ ist, der maximal q Anfragen an f stellt. Definiere weiter

$$\text{insec}(q, t, S) = \sup\{\text{adv} \mid K \ (q, t, l, L)\text{-Unterscheider}\}$$

Sei $\varepsilon > 0$. Ein System S heißt (q, t, ε)-unsicher, falls $\text{insec}(q, t, S) \geq \varepsilon$, andernfalls sicher.

BEISPIEL: Der one-time-pad der Länge 17 ist (2,50,0.5)-unsicher.

Problem: Für kein Kryptosystem konnte bisher eine sinnvolle Sicherheitsaussage bewiesen werden!

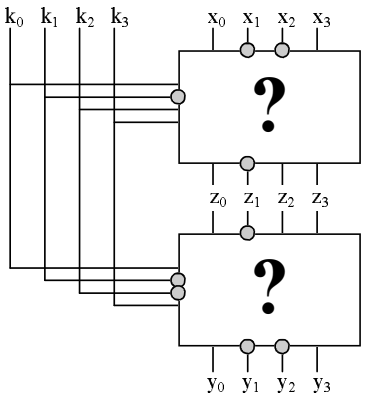
2.4 Lineare Kryptanalyse

Die lineare Kryptanalyse ist eine von vielen Techniken, wobei „linear“ bedeutet: man versucht, Teile des Blockkryptosystems **durch lineare Funktionen anzunähern** und damit den Suchraum zu verkleinern.

Betrachte beispielsweise ein Kryptosystem mit vier Bit Länge, d.h. $X = Y = K = 2^4$. Wenn wir aus der Analyse beispielsweise annehmen können, daß $y_2 = x_1 \oplus x_2 \oplus k_0 \oplus k_1$ für **viele**[2] Eingabebits x_1, x_2 gilt, so löse die Gleichung nach k_1 und k_0 auf und bestimme k_0 und k_1 daraus für viele Eingabemöglichkeiten (x^0, y^0), (x^1, y^1), ... – wobei dies k_0 und k_1 nicht eindeutig bestimmbar sind, aber über Zählen von Häufigkeiten werden Kanidaten gefunden.

Problem: Generell sollte es schwierig sein, für ein gesamtes Kryptosystem lineare Abhängigkeiten zu finden. **Idee:** Wir nutzen den modularen Aufbau!

BEISPIEL:



[2] für alle wird es nicht gelten, dann ist das zu Grunde liegende Kryptosystem zu einfach konstruiert

$$
\begin{aligned}
z_1 &= x_1 \oplus x_2 \oplus k_1 \\
y_1 \oplus y_2 &= z_1 \oplus k_1 \oplus k_2 \\
\Longrightarrow y_1 \oplus y_2 \oplus z_1 &= x_1 \oplus x_2 \oplus z_1 \oplus k_1 \oplus k_1 \oplus k_2 \\
\Longrightarrow y_1 \oplus y_2 &= x_1 \oplus x_2 \oplus k_2
\end{aligned}
$$

Anwendung findet dies bei so genannten *Substitutions-Permutations-Kryptosystemen (SPKS)*: Dies sind Blockkryptosysteme der Form $(\{0,1\}^{lm}, \{0,1\}^{lm}, K, E, D)$, das bestimmt wird durch eine S-Box $S \in \mathrm{Perm}(2^l)$ und eine Bitpermutation $B \in \mathrm{Perm}(\{0,\ldots,ml-1\})$ und einer Schlüsselerzeugungsfunktion $K\colon \{0,1\}^s \times \{0,\ldots,r\} \to \{0,1\}^{lm}$. Als Algorithmus ergibt sich:

$e_k(x)$

$y := x \oplus K(k,0);$
`for` i `=` 1 `to` r
  `for` j `=` 1 `to` $m-1$
    $y[lj, l(j+1)) = S(y[lj, l(j+1)));$
  `for` j `=` 0 `to` $lm-1$ `in parallel`
    $y(j) = y(B(j));$
  $y := y \oplus K(k,i);$

Eine graphische Darstellung findet sich im Anhang B.1.

Gesucht ist eine **lineare Beziehung** zwischen Klartextbits x, Schlüsselbits k und Vorchiffretextbits z (d.h. Bits aus dem Zwischenergebnis vor der verkürzten Runde), etwa der Form

$$z_4 \oplus z_7 = x_3 \oplus x_4 \oplus f(k)$$

Zusätzliche Annahme: Ein Teilwort des letzten Rundenschlüssels hängt nur von wenigen Schlüsselbits ab, etwa $k_{b_0}, \ldots, k_{b_{t-1}} = \bar{k}$. Benutze nun gegebene Klartext-/Chiffretextpaare und bestimme für jede Wahl von $\bar{k}$ und für jedes Paar (x,y) das zugehörige Paar $(x,\bar{z})$. Falls alle Paare die Gleichung erfüllen, so ist $\bar{k}$ ein Kandidat für die entsprechenden Schlüsselbits. Falls nicht, kann diese Kombination ausgeschlossen werden.

Beispiel: Seien im konkreten Beispiel $l = m = r = 4$, seien S und B gegeben durch die folgende Tabelle:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
$B(x)$	0	4	8	C	1	5	9	D	2	6	A	E	3	7	B	F

Schlüssellänge sei $s = 32$, die Rundenschlüssel seien gegeben durch

$K(k,0) = k[0,16)$ $K(k,1) = k[4,20)$ $K(k,2) = k[8,24)$ $K(k,3) = k[12,28)$ $K(k,4) = k[16,32)$

Mit hoher Wahrscheinlichkeit wird nun folgender Ausdruck 1:

$$x_0 \oplus k_0 \oplus x_2 \oplus k_2 \oplus x_3 \oplus k_3 \oplus y_1$$

Weiteres siehe Übung!

Alternative ist beispielsweise die differentielle Kryptanalyse, welche relativ erfolgreich bei DES mit geringer Rundenzahl angewandt werden kann.

3 Uneingeschränkte Verschlüsselung – Kryptoschemen

Das neues Szenario entspricht dem alten ohne die Einschränkung auf kurze Klartexte und daß Klartexte nur einmal verschlüsselt werden, d.h. beliebig lang beliebig häufig verschlüsseln.

3.1 Kryptoschemen

Ein *symmetrisches Kryptoschema (KSc)* der Blocklänge l ist ein Tupel (K, E, D) bestehend aus

- einer endlichen Schlüsselmenge K (meistens $\{0,1\}^m$),
- einem zufallsgesteuerten Chiffrieralgorithmus $E(x : (\{0,1\}^l)^*, k : K) : \{0,1\}^*$ und
- einem Dichiffrieralgorithmus $D(y : \{0,1\}^*, k : K) : (\{0,1\}^l)^*$, wobei
- für alle $x \in (\{0,1\}^l)^*$, $k \in K$, $\alpha \in \{0,1\}^\omega$ gilt: $D(E\langle\alpha\rangle(x,k),k) = x$, und zudem
- die Laufzeit von E und D soll polynomiell beschränkt sein

Die Benutzung ist klar, die Konstruktion von Kryptoschemen ist zweistufig: Zunächst muß eine Blockchiffre gewählt werden, dann muß die Blockchiffre durch Wahl eines so genannten *Modus* in ein Kryptoschema umwandeln. Sei nun ein l Blockkryptosystem wie üblich gegeben, definiere jeweils $x_i := x[li, l(i+1))$, entsprechend für y

- Das zugehörige **ECB-Kryptoschema** ist dann definiert durch folgenden Algorithmus:

$E(x : (\{0,1\}^l)^*, k : K) : (\{0,1\}^l)^*$
$m := |x| / l;$
`for` i `=` 0 `to` $m-1$
  $y_i = e_k(x_i);$
`return` $y_0 \ldots y_{m-1};$

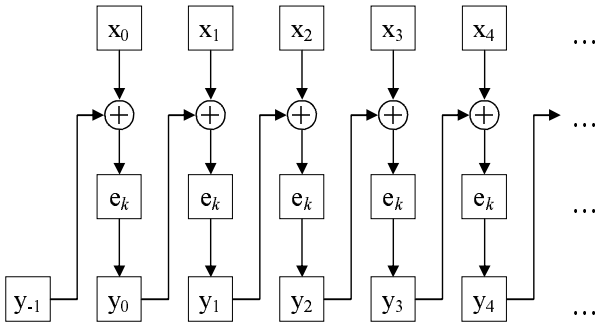
- Das zugehörige **CBC-Kryptoschema** ist definiert durch

$$\begin{array}{l} \underline{E(x : (\{0,1\}^l)^*, k : K) : (\{0,1\}^l)^*} \\ m := |x| \,/l; \\ y_{-1} := \texttt{const}; \\ \texttt{for } i \texttt{ = 0 to } m-1 \\ \quad y_i = e_k(x_i \oplus y_{i-1}); \\ \texttt{return } y_0 \ldots y_{m-1}; \end{array}$$

  Dabei nennt man y_{-1} auch *Initialvektor*. Die zugehörige Entschlüsselung:

$$\begin{array}{l} \underline{D(y : (\{0,1\}^l)^*, k : K) : (\{0,1\}^l)^*} \\ m := |y| \,/l; \\ y_{-1} := \texttt{const}; \\ \texttt{for } i \texttt{ = 0 to } m-1 \\ \quad x_i = d_k(y_i) \oplus y_{i-1}; \\ \texttt{return } x_0 \ldots x_{m-1}; \end{array}$$

  Graphisch:



- Das **R-CBC-Kryptoschema** verschlüsselt mit folgendem zufallsgesteuerten Algorithmus:

$$\begin{array}{l} \underline{E\,\langle\alpha\rangle\,(x : (\{0,1\}^l)^*, k : K) : (\{0,1\}^l)^*} \\ m := |x| \,/l; \\ y_{-1} := \alpha[0,l); \\ \texttt{for } i \texttt{ = 0 to } m-1 \\ \quad y_i = e_k(x_i \oplus y_{i-1}); \\ \texttt{return } y_{-1} y_0 \ldots y_{m-1}; \end{array}$$

  Entschlüsselt wird durch:

$$\begin{array}{l} \underline{D(y : (\{0,1\}^l)^*, k : K) : (\{0,1\}^l)^*} \\ m := |y| \,/l; \\ \texttt{for } i \texttt{ = 0 to } m-2 \\ \quad x_i = d_k(y_{i+1}) \oplus y_i; \\ \texttt{return } x_0 \ldots x_{m-2}; \end{array}$$

- Das Kryptoschema **R-CTR** nutzt die Idee des one-time-pads:

$$\begin{array}{l}\underline{E\,\langle\alpha\rangle\,(x:(\{0,1\}^l)^*, k:K):(\{0,1\}^l)^*} \\ m := |x|\,/l; \\ c := \alpha[0,l); \\ \texttt{for } i \texttt{ = } 0 \texttt{ to } m-1 \\ \quad y_i = x_i \oplus e_k(c+i); \\ \texttt{return } cy_0 \ldots y_{m-1}; \end{array}$$

Der Dechiffrieralgorithmus kommt ohne d_k aus:

$$\begin{array}{l}\underline{D(y:(\{0,1\}^l)^*, k:K):(\{0,1\}^l)^*} \\ m := |y|\,/l; \\ c := y[0,l); \\ \texttt{for } i \texttt{ = } 0 \texttt{ to } m-2 \\ \quad x_i = y_{i+1} \oplus e_k(c+i); \\ \texttt{return } x_0 \ldots x_{m-2}; \end{array}$$

3.2 Unterscheider und algorithmische Sicherheit

Ein l-Unterscheider für Kryptoschemen ist ein zufallsgesteuerter Algorithmus der Form

$$A(g:(\{0,1\}^l)^* \to \{0,1\}^*):\{\mathit{real}, \mathit{random}\}$$

Ist $g(x\colon(\{0,1\}^l)^*):\{0,1\}^*$ ein zufallsgesteuerter Algorithmus, so bezeichnet $R-g$ den Algorithmus

$$\begin{array}{l}\underline{R-g(x:(2^l)^*):2^*} \\ x' = \texttt{flip}(x); \\ \texttt{return } g(x'); \end{array}$$

Ist (K,E,D) ein l-Kryptoschema und A ein l-Unterscheider, so sind Erfolg, Misserfolg und Vorteil definiert durch:

$$\begin{array}{rcl} \mathrm{suc}(A,S) & = & P\,(\{\,A(E(k,\cdot)) = \mathit{real} \mid\ k \longleftarrow_R K\}) \\ \mathrm{fail}(A,S) & = & P\,(\{\,A(R-E(k,\cdot)) = \mathit{real} \mid\ k \longleftarrow_R K\}) \\ \mathrm{adv}(A,S) & = & \mathrm{suc}(A,S) - \mathrm{fail}(A,S) \end{array}$$

Beispiele:

- Unterscheider für ein **ECB-Kryptoschema** S mit Blocklänge l:

  $A :=$ ECB-Distingusher(g)
  $x := 0^l$;
  $y := g(x\,x)$;
  `if` $y[0,l) = y[l,2l)$
    `return` *real*;
  `else`
    `return` *random*;

  In der Realwelt ist ECB-Distinguisher A immer erfolgreich. In der Zufallswelt gibt der Algorithmus genau dann *real* aus, wenn das zufällig erzeugte Argument für g von der Form $x'x'$ ist (zur Erinnerung: $y = g(\texttt{flip}(2l))$), d.h.

  $$\mathrm{suc}(A,S) = 1 \quad \mathrm{fail}(A,S) = \frac{1}{2^l} \quad \mathrm{adv}(A,S) = 1 - \frac{1}{2^l}$$

- Unterscheider für **CBC-Kryptoschema** S

  $A :=$ CBC-Distingusher(g)
  $x := 0^l$;
  $y_0 := g(x)$;
  $y_1 := g(x)$;
  `if` $y_0 == y_1$
    `return` *real*;
  `else`
    `return` *random*;

  Hier gilt wieder:

  $$\mathrm{suc}(A,S) = 1 \quad \mathrm{fail}(A,S) = \frac{1}{2^l} \quad \mathrm{adv}(A,S) = 1 - \frac{1}{2^l}$$

  Dieser Unterscheider hat für jedes deterministische Kryptoschema, das einen Block der Länge l auf einen Block der Länge l abbildet, den Vorteil $1 - 2^{-l}$.

- Wir betrachten nun noch den **ECB-Distinguisher** A von oben für ein **CBC-Kryptoschema**. Dann verhält sich der Unterscheider in der

Zufallswelt wie folgt:

$x = 0^l$;
$y_0 = e_k(\texttt{flip}(l) \oplus y_{-1})$;
$y_1 = e_k(\texttt{flip}(l) \oplus y_0)$;
`if` $y_0 == y_1$
  `return` *real*;
`else`
  `return` *random*;

Also ist $\text{fail}(A, S) = 2^{-l}$. In der realen Welt verhält sich A wie folgt:

$x = 0^l$;
$y_0 = e_k(x \oplus y_{-1})$;
$y_1 = e_k(x \oplus y_0)$;
`if` $y_0 == y_1$
  `return` *real*;
`else`
  `return` *random*;

Dieser Algorithmus verhält sich genau so wie der folgende Unterscheider für das zugehörige Blockkryptosystem S_0:

$B :=$ CBC-Simulator(f)
$x = 0^l$;
$y_0 = f(y_1 \oplus x)$
$y_1 = f(y_0 \oplus x)$
`if` $y_0 == y_1$
  `return` *real*;
`else`
  `return` *random*;

Also ist $\text{suc}(A, S) = \text{suc}(B, S_0) \geq \text{adv}(B, S_0)$, insgesamt

$$\text{adv}(A, S) \geq \text{adv}(B, S_0) - 2^{-l}$$

DEFINITION: Seien n, q, t natürliche Zahlen und $\varepsilon > 0$. Ein (n, q, t, l)-*Unterscheider* ist ein l-Unterscheider, dessen Laufzeit durch t beschränkt ist, der höchstens q-mal den Prozedur-Parameter aufruft und dabei höchstens n Bits anfragt. Zu einem l-Kryptoschema S sei weiter definiert:

$$\text{insec}(n, q, t, S) := \sup\{\text{adv}(A, S) \mid A\ (n, q, t, l)\text{-Unterscheidet}\}$$

Das Kryptoschema heißt (n, q, t, ε)-*unsicher*, falls $\text{insec}(n, q, t, S) \geq \varepsilon$ gilt, sonst (n, q, t, ε)-*sicher*.

BEISPIEL: Jedes ECB-Kryptoschema ist $(2l, 1, \mathcal{O}(l), 1 - 2^{-l})$-unsicher.

Problem: Es gibt keine Sicherheitsbeweise, aber folgenden Hauptsatz:

SATZ: Sei S_0 ein l-Blockkryptosystem und S das zugehörige R-CTR-Kryptoschema. Für jede Wahl von $n, q, t, n < l2^l$ gilt:

$$\text{insec}(n, q, t, S) \leq \text{insec}(\frac{n}{l}, t + \mathcal{O}(n), S_0) + 2\frac{n(q-1)}{l \cdot 2^l}$$

BEWEIS: Sei S_0 ein Kryptoschema, sei weiter $S := \text{R} - \text{CTR} - S_0$. Wir verwenden weiter folgende Bezeichnungen: Die Länge des Blockkryptosystems ist l, ein Unterscheider fragt insgesamt n Bits an und stellt dafür q Anfragen, wobei er bei der i-ten Anfrage m_i Blöcke der Länge l testet. Damit gilt

$$n = q \cdot l \cdot (m_0 + m_1 + \ldots + m_{q-1})$$

Der Beweis erfolgt nun in vier Schritten:

1. Zu jedem l-Unterscheider $A(g)$ für S betrachten wir einen passenden l-Unterscheider $B(f)$ für S_0, der aus A dadurch entsteht, daß jeder Aufruf $g(x)$ ersetzt wird durch

   $r := \texttt{flip}(l);$
   $m := |x| / l;$
   $y := r;$
   `for` $i = 0$ `to` $m - 1$
   $\quad y := y \cdot (f(r + i) \oplus x[il, (i + 1)l));$
   `return` $y;$

   Es reicht zu zeigen:

   $$\text{adv}(A, S) \leq \text{adv}(B, S_0) + 2\frac{n(q-1)}{l2^l}$$

   Dies ist äquivalent zu

   $$\text{suc}(A, S) - \text{fail}(A, S) \leq \text{suc}(B, S_0) - \text{fail}(B, S_0) + 2\frac{n(q-1)}{l2^l}$$

   Dabei können wir den Erfolg suc auf beiden Seiten streichen, da sich A und B in der Realwelt gleich verhalten. Dies ist wiederum äquivalent zu

   $$\text{fail}(B, S_0) \leq \text{fail}(A, S) + 2\frac{n(q-1)}{l2^l}$$

Die Zufallswelt sieht hierbei so aus, daß B zufällige Permutationen erhält. Wir untersuchen auch noch, wie sich B verhält, wenn er mit zufälligen Funktionen (!) (anstelle von zufälligen Permutationen) gespeist wird.

Zur Notation: Ausgabe von A in der Zufallswelt sei X, Ausgabe von B in der normalen Zufallswelt sei Y, Ausgabe von B in der modifizierten Zufallswelt sei Y'. Dann ist obiges äquivalent zu

$$P(Y = real) - P(X = real) \leq 2\frac{n(q-1)}{l2^l}$$

Nun führen wir Y' ein:

$$(P(Y = real) - P(Y' = real)) - (P(Y' = real) - P(X = real)) \leq 2\frac{n(q-1)}{l2^l}$$

Damit bleibt zu zeigen:

$$\begin{aligned} & P(Y = real) - P(Y' = real) \;\leq\; \frac{n(q-1)}{l2^l} \quad \text{(Schritt 1)} \\ \wedge \quad & P(Y' = real) - P(X = real) \;\leq\; \frac{n(q-1)}{l2^l} \quad \text{(Schritt 2 und 3)} \end{aligned}$$

2. Zu zeigen ist

$$P(Y' = real) - P(X = real) \leq \frac{n(q-1)}{l2^l}$$

Wie läuft der B ab? Es erfolgen die folgenden Anfragen:

$$\begin{array}{cccc} f(r_0) \oplus x_0^0 & f(r_0+1) \oplus x_1^0 & \dots & f(r_0+(m_0-1)) \oplus x_{m_0-1}^0 \\ f(r_1) \oplus x_0^1 & f(r_1+1) \oplus x_1^1 & \dots & f(r_1+(m_1-1)) \oplus x_{m_1-1}^1 \\ \vdots & \vdots & \vdots & \vdots \\ f(r_{q'-1}) \oplus x_0^{q'-1} & f(r_{q'-1}+1) \oplus x_1^{q'-1} & \dots & f(r_{q'-1}+(m_{q'-1}-1)) \oplus x_{m_{q'-1}-1}^{q'-1} \end{array}$$

Falls die in der Matrix stehenden Zahlen $r_i + j$ alle verschieden sind, kann man sich die Tabelle als zufällig erzeugt vorstellen. Andererseits: Wenn A in der Zufallswelt läuft, ist die Tabelle aufgrund der zufälligen Wahl der x_i^j zufällig.

Formalisierung: Sei Dstct das Ereignis, daß alle $r_i + j$ unterschiedlich sind. Sei $P(\overline{\text{Dstct}}) = \delta$. Zuerst gilt $P(X = real) = P(Y' = real \,|\, \text{Dstct})$, außerdem

$$P(Y' = real) = P(Y' = real \,|\, \text{Dstct})(1-\delta) + P(Y' = real \,|\, \overline{\text{Dstct}}) \cdot \delta$$

Also gilt:

$$\begin{aligned} & P(Y' = real) - P(X = real) \\ = \quad & (P(Y' = real \,|\, \overline{\text{Dstct}}) + P(Y' = real \,|\, \text{Dstct})) \cdot \delta \leq \delta \end{aligned}$$

Noch zu zeigen ist $\delta \leq \frac{n(q-1)}{l2^l}$.

3. Seien $q, m_0, \ldots, m_{q-1}$ und N natürliche Zahlen. Wir wählen $r_0, \ldots, r_{q-1} < N$ zufällig mit gleicher Wahrscheinlichkeit und betrachten das Ereignis

$$\mathrm{Cll}(q, m_0, \ldots, m_{q-1}, N)$$

, daß unter den folgenden Zahlen eine mehrfach vorkommt:

$$\begin{array}{cccc} r_0 & r_0+1 & \ldots & r_0+(m_0-1) \\ r_1 & r_1+1 & \ldots & r_1+(m_1-1) \\ \vdots & \vdots & \vdots & \vdots \\ r_{q'-1} & r_{q'-1}+1 & \ldots & r_{q'-1}+(m_{q'-1}-1) \end{array}$$

Falls $m_0 = m_1 = \ldots = m$, so bezeichne das Ereignis mit (q, m, N), weiter falls $m = 1$ ist schreibe $\mathrm{Cll}(q, N)$.

LEMMA: Es gilt

$$P(\mathrm{Cll}(q, m, \ldots, m_{q-1}, N)) \leq \frac{(q-1)(m_0 + \ldots + m_{q-1})}{N}$$

BEWEIS: Bezeichne mit Dstct_i das Ereignis, daß bis zur der i-ten Zeile alles unterschiedlich ist. Dann ist

$$\begin{aligned} P(\overline{\mathrm{Dstct}}) &= P(\overline{\mathrm{Dstct}_q}) \\ &= P(\overline{\mathrm{Dstct}_{q-1}}) + P(\overline{\mathrm{Dstct}_q} \,|\, \mathrm{Dstct}_{q-1}) \cdot P(\mathrm{Dstct}_{q-1}) \\ &\leq P(\overline{\mathrm{Dstct}_{q-1}}) + P(\overline{\mathrm{Dstct}_q} \,|\, \mathrm{Dstct}_{q-1}) \\ &\ldots \\ &\leq \sum_{i=2}^{q} P(\overline{\mathrm{Dstct}_i} \,|\, \mathrm{Dstct}_{i-1}) \end{aligned}$$

Abschätzung:

$$\begin{aligned} P(\mathrm{Dstct}_i \,|\, \mathrm{Dstct}_{i-1}) &\leq \frac{(m_{i-1} + m_0 - 1) + \ldots + (m_{i-1} + m_{i-2} - 1)}{N} \\ &= \frac{(i-1)m_{i-1} - (i-1) + m_0 + \ldots + m_{i-2}}{N} \\ &\leq \frac{(i-1)m_{i-1} + m_0 + \ldots + m_{i-2}}{N} \end{aligned}$$

Einsetzen liefert:

$$\begin{aligned} P(\overline{\mathrm{Dstct}}) &\leq \sum_{i=2}^{q} \frac{(i-1)m_{i-1} + m_0 + \ldots + m_{i-2}}{N} \\ &\leq \frac{(q-1)(m_0 + \ldots + m_{q-1})}{N} \end{aligned}$$

4. Zu zeigen ist:

$$P(Y = \mathit{real}) - P(Y' = \mathit{real}) \leq \frac{n(q-1)}{l2^l}$$

Wir untersuchen den Unterschied zwischen normaler Zufallswelt (beliebige permutation) und modifizierter Zufallswelt (beliebige Funktion). Vorstellung: In der modifizierten Zufallswelt werden die Funktionswerte ausgewürftelt, wenn sie benötigt werden. D.h., wir befinden uns in ähnlicher Situation wie eben. Wir wissen: $P(Y = \mathit{real}) = P(Y' = \mathit{real} \,|\, \mathrm{Dstct})$, und

$$P(Y' = \mathit{real}) = P(Y' = \mathit{real} \,|\, \mathrm{Dstct})(1-\delta) + P(Y' = \mathit{real} \,|\, \overline{\mathrm{Dstct}}) \cdot \delta$$

analog zu oben. Das Lemma aus dem dritten Schritt liefert die gewünschte Abschätzung (mit $m = 1$). ■

3.3 Eine untere Schranke und ein Beispiel

Ziel: wir wollen einsehen, daß die obere Schranke aus dem Hauptsatz recht gut ist. Dazu konstruieren wir einen konkreten Unterscheider. Wiederholung des R-CTR-Schemas:

$$\begin{array}{l} \underline{E\,\langle\alpha\rangle\,(x : (2^{lm}), k : K) : (2^{lm})} \\ m := |x|\,/l; \\ y_{-1} := \alpha[0, l); \\ \texttt{for } i \texttt{ = } 0 \texttt{ to } m-1 \\ \quad y_i = x_i \oplus e_k(y_{-1} + i); \\ \texttt{return } y_{-1}y_0 \ldots y_{m-1}; \end{array}$$

Idee: Wir stellen q Anfragen und lassen jeweils 0^{lm} verschlüsseln, dann wird folgende Matrix M zurückgeliefert:

$$M = \begin{pmatrix} r_0 & e_k(r_0) & e_k(r_0+1) & \ldots & e_k(r_0+m-1) \\ r_1 & e_k(r_1) & e_k(r_1+1) & \ldots & e_k(r_1+m-1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ r_{q-1} & e_k(r_{q-1}) & e_k(r_{q-1}+1) & \ldots & e_k(r_{q-1}+m-1) \end{pmatrix}$$

Dann gilt in der realen Welt: Wenn $|r_i - r_j| < m$, etwa $r_j - r_i = m$, dann gilt

$$e_k(r_i + (r_j - r_i)) \oplus x^i_{r_j - r_i} = e_k(r_j) x^j_0$$

Das heißt: in obiger Matrix M ist $M_{i,r_j-r_i+1} = M_{j,1}$. Der Unterscheider arbeitet nun folgendermaßen:

1. Stelle q Anfragen mit $x = 0^{lm}$.
2. Überprüfe, ob r_i und r_j der obigen Art existieren.
3. Wenn ja, dann prüfe, ob $M_{i,r_j-r_i+1} = M_{j,1}$ gilt – wenn ja, gibt *real* zurück, ansonsten *random*.

R-CRT-Distinguisher(g)

```
for i = 0 to q − 1
  y_1 = g(0^{lm});
find i and j so that
  (i ≠ j) ∧ (0 ≤ y_j[0, l) − y_i[0, l) < m)
if no such values exist
  return (flip == 0) ? real : random;
else
  m' = y_j[0, l) − y_i[0, l)
  if y_j[l, 2l) == y_i[l(m' + 1), l(m' + 2))
    return real;
  else
    return random;
```

SATZ: Ist S ein Blockkryptosystem der Blocklänge $l \geq 5$ und bezeichnet A den obigen Unterscheider, so gilt $adv(A, S) =\geq 0.3\frac{n(q-1)}{l2^l}$; und falls zudem $q \leq 2^{\frac{l+1}{2}}$, so ist

$$\text{insec}(n, q, t, S) \geq 0.3\frac{n(q-1)}{l2^l}$$

BEWEIS: Egal, in welcher Welt wir uns befinden, die zugrunde liegenden Chiffren werden auf die Matrix angewandt, wobei die r_i zufällig erzeugt werden:

$$M = \begin{pmatrix} r_0 & r_0+1 & \dots r_0+m-1 \\ r_1 & r_1+1 & \dots r_1+m-1 \\ \vdots & \vdots & \vdots \\ r_{q-1} & r_{q-1}+1 & \dots r_{q-1}+m-1 \end{pmatrix}$$

Sei Dstct das Ereignis, daß alle Werte unterschiedlich sind. X bezeichne das Ereignis von A in der Realwelt, Y in der Zufallswelt. Wenn Dstct eintritt, liefert A mit gleicher Wahrscheinlichkeit *real* oder *random*. Also: gilt:

$$\begin{aligned} \text{adv}(A,S) &= P(\overline{\text{Dstct}})(P(X = \mathit{real}\,|\,\overline{\text{Dstct}}) - P(Y = \mathit{real}\,|\,\overline{\text{Dstct}})) \\ &\quad + P(\text{Dstct})(\underbrace{P(X = \mathit{real}\,|\,\text{Dstct}) - P(Y = \mathit{real}\,|\,\overline{\text{Dstct}})}_{=0}) \\ &\geq P(\overline{\text{Dstct}})(1 - 2^l) \end{aligned}$$

Wir müssen noch eine untere Schranke für $\overline{\text{Dstct}}$finden. Das Ereignis $\overline{\text{Dstct}}$tritt genau dann ein, wenn zwei der r_i eine Differenz $< m$ besitzen, also insbesondere schon dann (aber auch in anderen Fällen), wenn zwei der r_i im selben Intervall der Form $[km, (k+1)m)$ liegen. Weiter gilt: $P(\overline{\text{Dstct}}) \geq \text{Cll}(q, \frac{2^l}{m})$, da entsprechend q Zahlen r_i aus 2^l gezogen werden und jeweils in eines der $\frac{2^l}{m}$ Intervalle der Länge m eingeteilt wurden.

Die Behauptung ergibt sich also aus dem folgenden Lemma unter Berücksichtigung der Ungleichung $0.63(1 - 2^{-5}) \geq 0.6$.

LEMMA: (Geburtstagsphänomen) Für positive Zahlen q und N sei $\varepsilon(q, N) = \frac{q(q-1)}{2N}$. Dann gilt:

$$1 - e^{-\varepsilon(q,N)} \leq \text{Cll}(q, N) \leq \varepsilon(q, N)$$

und für $q \leq \sqrt{2, N}$ gilt:

$$0.63 \cdot \varepsilon(q, N) \leq 1 - e^{-\varepsilon(q,N)}$$

BEWEIS: Sei C_i das Ereignis, daß die i-te ausgewürfelte Zahl mit einer der vorherigen übereinstimmt. Dann gilt $P(C_i) \leq \frac{i-1}{N}$, also:

$$\text{Cll}(q, N) \leq P(C_2 \cup \ldots \cup C_q) \leq \sum_{i=2}^{q} \frac{i-1}{N} = \varepsilon(q, N)$$

Sei D_i nun das Ereignis, daß wir noch keine gleichen Zahlen gefunden haben, nachdem schon i ausgewählt wurden. Dann gilt:

$$\begin{aligned}
1 - \text{Cll}(q, N) &\leq P(D_q) \\
&= P(D_q \mid D_{q-1}) P(D_{q-1}) \\
&= P(D_q \mid D_{q-1}) P(D_{q-1} \mid D_{q-2}) P(D_{q-2}) \\
&= \prod_{i=1}^{q-1} P(D_{i+1} \mid D_i) = \prod_{i=1}^{q-1} (1 - \frac{1}{N}) \\
&\leq \prod_{i=1}^{q-1} e^{-\frac{i}{N}} = e^{-\varepsilon(q,N)}
\end{aligned}$$

Die spezielle Schranke folgt unter Benutzung von $(1 - \frac{1}{e})x \leq 1 - e^{-x}$ für alle $x \in [0, 1]$. ■

Abschlußüberlegung: Wir betrachten AES im R-CTR-Modus und nehmen

an, daß der beste Angreifer auf AES der ist, der einfach q Anfragen stellt und dann testet, ob die gewonnenen Klartext-/Chiffretextpaare zu s zufällig gewählten Schlüsseln passen. Wir wollen $\mathrm{adv}(A, \mathrm{AES})$ nach oben abschätzen. Es ist

$$\mathrm{insec}(q, s, \mathrm{AES}) \leq \frac{s}{2^{128}}$$

Dies gilt nicht formal, aber durch das große Verhältnis zwischen der Anzahl möglicher Permutationen $(2^{128})!$ und den möglichen Schlüsseln 2^{128} läßt sich damit gut rechnen.

4 Asymmetrsiche Verschlüsselung

Szenario: Alice möchte einem bislang unbekannten Kommunikationspartner Bob eine Nachricht über einen unsicheren (abhörbaren) Kanal schicken.

Idee: Trennung zwischen Schlüssel k_b zum Verschlüsseln (wird veröffentlicht) und Schlüssel $\hat{k}_b$ zum Entschlüsseln (bleibt beheim) – Verfahren: Alice verschickt $y = e_{k_b}(x)$ und Bob ermittelt $x = d_{\hat{k}_b}(y)$. Wichtig ist, daß es schwierig sein muß, $\hat{k}$ aus k zu berechnen. Erstes Beispiel für ein solches System ist RSA.

DEFINITION: Ein asymmetrisches Kryptoschema ist ein Tupel (E, G, D) bestehend aus

- einem zufallsgesteuerten Chiffrieralgorithmus $E(x : 2^*, k : 2^*) : 2^*$
- einem zufallsgesteuerten Schlüssel(paar)erzeugungsalgorithmus $G(p : \{0\}^*) : 2^* \times 2^*$ (wobei p der *Sicherheitsparameter* ist)
- einem Dechiffrieralgorithmus $D(y : 2^*, k : 2^*) : 2^*$

Es muß $D(E\langle\alpha\rangle(x, k), \hat{k}) = x$ für alle Paare $(k, \hat{k})$ gelten, die G erzeugt. Die Laufzeiten von E, G und D sollen polynomiell beschränkt sein.

Intuitiv: $E(k, \cdot)$ ist schwer zu invertieren ohne Kenntnis von $\hat{k}$ – insbesondere darf $\hat{k}$ nicht leicht aus k herleitbar sein.

4.1 Einwegfunktionen mit Hintertüren

DEFINITION: Eine *invertierbare Funktionsfamilie* ist ein Tupel $(\mathcal{F}, t, \mathcal{G})$ bestehend aus Familien $\mathcal{F} = \{f_i\}_{i\in I}$, $\mathcal{G} = \{g_j\}_{j\in J}$ von partiellen Funktionen $2^* \dashrightarrow 2^*$ mit endlichem Definitionsbereich, wobei die Indexmengen $I, J \subseteq 2^*$ sind und die Abbildung $t\colon I \to J$ so beschaffen ist, daß $g_{t(i)}(f_i(x)) = x$ für alle $x \in \mathrm{Def}(f_i)$.

BEISPIEL: Die RSA-Familie:

- Sei $I = J$ bestehend aus Paaren (n, e) mit $n = p \cdot q$, $p \neq q$ Primzahlen und e invertierbar in $\mathbb{Z}_{(p-1)(q-1)}$, d.h. $e \in \mathbb{Z}_m^*$ mit $m = (p-1)(q-1)$.
- Sei $\mathcal{F} = \mathcal{G}$ wobei $f_{(n,e)}\colon \mathbb{Z}_n \to \mathbb{Z}_n$ mit $x = x^e \bmod n$
- Sei $t\colon (n, e) \to (n, d)$ definiert durch $e \cdot d \bmod m = 1$.

Zu zeigen ist hier noch: $(x^e \bmod n)^d \bmod n = x$ für alle $x \in \mathbb{Z}_n$. Fallunterscheidung:

- $x \in \mathbb{Z}_n^*$. Dann gilt: $x^m \bmod n = 1$. Nun gilt:

$$(x^e \bmod n)^d \bmod n = (x^{e^d}) \bmod n = x^{ed} \bmod n$$

  Da d invers zu e ist, ist $ed \bmod m = 1$, also existiert k mit $ed = km + 1$, somit ist

$$x^{ed} \bmod n = x^{km+1} \bmod n = x \bmod n$$

- $x \in \mathbb{Z}_n \setminus \mathbb{Z}_n^* \longrightarrow$ in der Übung

Zusätzliche **algorithmische** Aspekte:

DEFINITION: Eine *Einwegfamilie mit Hintertüren* besteht aus einer invertierbaren Funktionsfamilie (FF) $(\mathcal{F}, t, \mathcal{H})$ und den folgenden Algorithmen:

- ein Polynomzeit-Algorithmus, der die f_is berechnet,
- ein Polynomzeit-Algorithmus, der die h_js berechnet,
- ein Polynomzeit-Algorithmus $G(p : \{0\}^*) : I \times J$, der die Elemente von $\{(i, t(i)) \mid i \in I\}$ erzeugt,
- ein zufallsgesteuerter Algorithmus $D(i \colon I) : 2^*$, der zu jedem $i \in I$ die Elemente des Definitionsbereichs von f_i liefert,

Wichtigste Bedingung ist die **Einweg-Eigenschaft**: Für jeden Polynomzeit-Algorithmus $A(i : I, y : 2^*) : 2^*$ gilt

$$P(\{f_i(A(i, f_i(x))) = f_i(x) \mid (i, j) \in G(\{0\}^n) \wedge x \in D(i)\}) \in \mathcal{O}(n^{-c}) \; \forall \, c > 0$$

4.2 Die RSA-Familie

Der Name RSA stammt von den Erfindern RIVEST, SHAMIR, ADLEMAN. Wir untersuchen nun die Eigenschaften einer Einwegfamilie mit Hintertüren:

- f_i bzw. h_j sind in Polynomzeit berechenbar: Z.z. ist: $x^e \bmod n$ läßt sich berechnen polynomiell in $\log n$.

  Idee: Exponentation durch iteriertes Quadrieren.

- Erzeugung der Paare $((n, e), (n, d))$ – dabei entstehen zwei Probleme: Man muß große Primzahlen finden und Elemente von $\mathbb{Z}_m^*$ finden. **Ideen:** Primzahlen finden wir durch raten und testen, ebenso die Elemente von

$\mathbb{Z}_m^*$.

LEMMA: Sei $m > 0$. Dann gilt:

$$\frac{\varphi(m)}{m} \geq \frac{1}{\log m}$$

BEWEIS: Es gilt:[3]

$$\begin{aligned} \frac{\prod_i p_i^{\alpha_i - 1}(p_i - 1)}{\prod_i p_i^{\alpha_i}} &= \prod_i \frac{p_i - 1}{p_i} \geq \prod_{i<r} \frac{i+1}{i+2} \\ &\geq \frac{1}{\log m} = \frac{1}{r+1} \geq \frac{1}{\log m + 1} \end{aligned}$$

Das Produkt von Quotienten aus Primzahlen, z.B. $\frac{1}{2} \cdot \frac{2}{3} \cdot \frac{6}{7} \cdot \frac{12}{13} \cdot \ldots$, schätzen wir dabei einfach ab durch $\frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} \cdot \frac{4}{5} \cdot \ldots$. ■

Also: Die Wahrscheinlichkeit, daß bei $k = \log^2 m$ Versuchen kein Element aus $\mathbb{Z}_m^*$ gefunden wird, ist

$$\left(1 - \frac{1}{\log m}\right)^{\log^2 m} = \left(\frac{1}{e}\right)^{\log m}$$

SATZ: Zu jeder natürlich Zahl n sei $\pi(n)$ gegeben durch $\pi(n) = \{p \leq n \mid p \in \mathbb{P}\}$. Dann gilt für alle $n >$ und $c = 0.92129 \cdot \log_2 e$ und $C = 1.105548 \cdot \log_2 e$:

$$c \cdot \frac{n}{\log n} \leq \pi(n) \leq C \cdot \frac{n}{\log n}$$

Folgerung als Übungsaufgabe: Es gibt genug Primzahlen einer vorgegebenen Länge. In der Praxis werden probabilistische Algorithmen zum Primzahltest verwendet, es gilt jedoch sogar:

SATZ [AGRAWAL 2002]: Man kann in Polynomzeit überprüfen, ob eine Zahl eine Primzahl ist.

Annahme: Die RSA-Familie besitzt die Einweg-Eigenschaft.

[3] wobei die Abschätzungen so ungeau sind, daß man das +1 am Ende noch wegbekommen könnte...

Diese **Annahme** ist verwandt mit der **Annahme**, daß Faktorisieren schwierig ist.

Bemerkungen:

1. Wenn man $\varphi(n)$ berechnen kann, dann kann man auch p und q berechnen.
2. Wenn man aus (n, e) d berechnen kann, dann kann man auch p und q berechnen
3. Offene Frage: Falls RSA keine Einweg-Funktion ist, dann ist Faktorisierung einfach.

4.3 Angriffe auf RSA

Ziel: Wir wollen uns klarmachen, daß man RSA nicht „einfach so“ zum Verschlüsseln von Blöcken benutzen darf, d.h., indem man $x < n$ durch $x^e \bmod n$ verschlüsselt.

Definition: Ist $a \in \mathbb{N}_{>1}$ und $n \in \mathbb{Z}$, dann ist n ein *quadratischer Rest modulo* a, wenn es ein $x \neq 0$ gibt, so daß $x^2 \bmod a = n \bmod a$ gilt.

Lemma: Sei p eine ungerade Primzahl und n eine ganze Zahl.

1. Für die Zahl $j = n^{\frac{p-1}{2}} \bmod p$ gilt: $j \in \{-1, 0, 1\}$.
2. Die Zahl n ist ein quadratischer Rest modulo p genau dann, wenn $j = 1$ ist.
3. Die Zahl n ist Vielfaches von p genau dann, wenn $j = 0$ ist.

Im Beweis benutzt man, daß $n^{p-1} = 1 \bmod p$ ist für $0 < n < p$.

DEFINITION: Für n ganze Zahl und p Primzahl setzen wir das *Jacobi- oder Legendre-Symbol*[4]:

$$J_p(n) = \begin{cases} 1 & \text{falls} \quad n \text{ quadratischer Rest} \\ 0 & \text{falls} \quad n = 0 \bmod p \\ -1 & \text{sonst} \end{cases}$$

Dabei kann J_p in Polynomzeit berechnet werden. Für eine ganze Zahl $a = \prod p_i^{\alpha_i}$ setzen wir

$$J_a(n) = \prod (J_{p_i}(n))^{\alpha_i}$$

Dann gilt nicht mehr, daß $J_a(n) = 1$ ist genau dann, wenn n quadratischer Rest ist. Zudem ist unklar, ob $J_a(n)$ polynomiell berechnet werden kann. Dazu:

LEMMA: Für $m, n \in \mathbb{Z}$ und $a > 0$ gilt:

- $J_a(1) = 1$
- $J_a(-1) = (-1)^{\frac{a-1}{2}}$ für a ungerade
- $J_a(2) = (-1)^{\frac{a^2-1}{8}}$ für a ungerade
- $J_a(mn) = J_a(m)J_a(n)$
- $J_a(n) = J_a(n \bmod a)$
- $J_a(n) = (-1)^{\frac{n-1}{2}\frac{a-1}{2}} J_n(a)$ für n, a ungerade und teilerfremd

BEISPIEL:

$$J_{487}(293) = J_{293}(487) = J_{293}(194) = J_{293}(2) \cdot J_{293}(97) = \ldots$$

Folgerung: Iterierte Anwendung liefert Polynomzeitalgorithmus!

LEMMA: $J_n(x^e \bmod n) = J_n(x)$ für $n, e, \ldots$ wie bei RSA.

BEWEIS: Es gilt (wegen e ungerade):

$$J_n(x^e \bmod n) = J_n(x^e) = (J_n(x))^e = J_n(x)$$

Das bedeutet: Würde man RSA „einfach so“ zum Verschlüsseln benutzen, würde sich die Eigenschaft „x hat Jacobi-Symbol 1“ auf den Chiffretext übertragen und umgekehrt! Außerdem ist J schnell berechenbar und nicht-trivial, denn viele Element haben $J = 1$ und viele $J = -1$. ■

4.4 Algorithmische Sicherheit

Ein *Erzeuger-Unterscheider-Paar* ist ein Paar (M, A) bestehend aus einem Klartext-Paar-Erzeuger $M(: \{0\}^*) : 2^* \times 2^*$ und einem Unterscheider

$$A(p : \{0\}^*, k : 2^*, x_0 : 2^*, x_1 : 2^*, y : 2^*) : 2$$

wobei beide zufallsgesteuert sein dürfen und in erwarteter Polynomzeit arbeiten müssen.

Vorstellung: M erzeugt Klartext-Klartext-Paare und A versucht, diese zu unterscheiden.

Wir betrachten zu M und A und einem vorgegebenem asymmetrischem Kryptoschema S:

$B(M, A, p)$

$(k, \hat{k}) = G(p);$
$(x_0, x) = M(p);$
$b = flip;$
$y = E(k, x_b);$
$b' = A(p, k, x_0, x_1, y);$
`if` $b == b'$
  `return` 1;
`else`
  `return` 0;

Der Vorteil von (M, A) bezüglich S für den Sicherheitsparameter p ist gegeben durch

$$\text{adv}(p, M, A, S) = P(B(M, A, p) = 1) - \frac{1}{2}$$

Beispiele:

- Wenn A ignorant wäre und einfach b' auswürfeln würde, dann wäre

$$\text{adv}(p, M, A, S) = 0$$

- Wenn S ein deterministisches Kryptoschema wäre (d.h. deterministische Verschlüsselung), würde das folgende Erzeuger-Unterscheider-Paar einen

großen Vorteil haben:

$M(p)$
$return(0, 1);$

$A(p, k, x_0, x_1, y)$
$y_0 = E(k, 0);$
`if` $y == y_0$
  `return` 1;
`else`
  `return` 0;

Hier gilt dann $\text{adv}(p, M, A, S) = \frac{1}{2}$.

- RSA-Erzeuger-Unterscheider-Paar auf der Grundlage des Jacobi-Symbols.

Sei S ein asymmetrisches Kryptoschema und (M, A) wie oben. (M, A) *hat keinen Erfolg gegen* S, falls $\text{adv}(p, M, A, S) \in \mathcal{O}(p^{-c})$ für alle $c > 0$. Das asymmetrische Kryptoschema heißt *sicher*, wenn es kein Erzeuger-Unterscheider-Paar gibt, das gegen S erfolgreich ist.

4.5 Schwierige Prädikate

Idee für sichere Verschlüsselung: bitweise! Aber wie?

Antwort: Wir suchen eine schwierige Klartexteigenschaft und verschlüsseln 1 durch den Funktionswert eines Klartextes, der die Eigenschaft hat, und 0 durch den Funktionswert eines Klartextes, der die Eigenschaft nicht hat.

Was heißt „schwierig“? Ein *schwieriges Prädikat* einer Einwegfamilie mit Hintertüren $(\mathcal{F}, t, \mathcal{H})$ besteht aus einer Familie von Funktionen $\mathcal{P} = \{p_i\}_{i \in I}$ mit den folgenden Eigenschaften:

1. Für jedes $i \in I$ haben f_i und p_i denselben Definitionsbereich.
2. Für jedes $i \in I$ hat p_i den Wertebereich $\{0, 1\}$.
3. Es gibt einen Polynomzeitalgorithmus, $P(i, x) : 2$, der $\mathcal{P}$ berechnet.
4. Für jeden zufallsgesteuerten Polynomzeitalgorithmus $B(i, y) : 2$ gilt:

$$P(B(i, f_i(x)) = p_i(x) \,|\, i = G(0^n) \text{ und } x = D(i)) - \frac{1}{2} \in \mathcal{O}(n^{-c}) \; \forall \; c > 0$$

Beispiel: Das LSB-Prädikat: $p_i(x) = x \bmod 2$.

SATZ: Das LSB-Prädikat ist unter der RSA-Annahme ein schwieriges Prädikat für RSA.

Allgemeiner:

SATZ: Zu einer Einwegfamilie mit Hintertüren $(\mathcal{F}, t, \mathcal{H})$ betrachten wir $(\mathcal{F}', t, \mathcal{H}')$ definiert wie folgt: Für jedes x im Definitionsbereich von f_i und jedes $r \in 2^*$ mit $|x| = |r|$ ist xr im Definitionsbereich von f'_i und es gilt: $f'_i(xr) = f_i(x)r$. Für jedes y im Definitionsbereich von h_j und jedes $r \in 2^*$ mit $|y| = |r|$ ist yr im Definitionsbereich von h'_j und es gilt $h'_j(yr) = h_j(y)r$. Dann ist $(\mathcal{F}', t, \mathcal{H}')$ eine Einwegfamilie mit Hintertüren und $\mathcal{P}$ mit $|x| = l$

$$p_i(xr) = x_0 r_0 \oplus \ldots \oplus x_{l-1} r_{l-1}$$

4.6 Bitweise Verschlüsselung

Gegeben sei eine Einwegfamilie mit Hintertüren $(\mathcal{F}, t, \mathcal{H})$ und ein schwieriges Prädikat $\mathcal{P}$. Dann ist

$E(x, k)$

$y := \varepsilon$;
$n := |x|$;
`for` $i = 0$ `to` $n - 1$
  $b := x_i$;
  `repeat`
    $x' := D(k)$;
    $b' := \mathcal{P}(x', k)$;
  `until` $b == b'$;
$y := y \# f_k(x')$;
`return` y;

SATZ: Das obige asymmetrische Kryptoschema ist sicher.

BEWEIS: (Skizze, Spezialfall) zu zeigen: Aus einem erfolgreichen Erzeuger-Unterscheider-Paar (M, A) läßt sich ein erfolgreicher Angreifer B auf $\mathcal{P}$ konstruieren. Stark vereinfachende Annahme: M liefert nur ein Paar (x_0, x_1) und x_0 und x_1 unterscheiden sich nur in einem Bit, etwa $x_0 = u0v$ und $x_1 = u1v$ mit $|u| = r$. Dann kann A gut unterscheiden zwischen Wörtern der

Form

$$\begin{array}{ll} & y_0\#y_1\#\ldots\#y_{r-1}\#w\#y_{r+1}\#\ldots\#y_{l-1} \\ \text{und} & y_0\#y_1\#\ldots\#y_{r-1}\#w'\#y_{r+1}\#\ldots\#y_{l-1} \end{array}$$

mit $y_i, y_i' \in f_k(X_i)$ und $w \in f_k(\{x \mid p(k,x) = 0\})$ sowie $w' \in f_k(\{x \mid p(k,x) = 1\})$, wobei $X_i = \{x \mid p_k(x) = x_0(i)\}$. B sieht deshalb wie folgt aus:

$B(p,k,y)$

$(x_0, x_1) := M(p,k);$
$z_0\#\ldots\#z_{l-1} := E(x_0,k);$
$\alpha := z_0\#\ldots\#z_{r-1}\#y\#z_{r+1}\#\ldots\#z_{l-1};$
`return` $A(p,k,x_0,x_1,\alpha);$

Bemerkungen: ■

- Das beschriebene, sichere Verfahren ist viel zu aufwendig: Ein Bit wird durch 1024/2048 Bits verschlüsselt!
- Die erste Alternative nutzt ein anderes, viel komplexeres Verfahren, das die „Chiffretextexplosion“ vermeidet.
- Eine weitere Alternative (OAEP): die Benutzung von Hashfunktionen bringt eine Effizienzsteigerung, hier ist jedoch eine zusätzliche Annahme über die Hashfunktion nötig!
- Die dritte Alternative ist ein hybrides Verfahren: Ein symmetrischer Schlüssel wird asymmetrisch verschlüsselt (sicher), dann wird der Rest symmetrisch verschlüsselt.
- Praxisrelevante Alternative: Ein weiteres hybrides Verfahren mit „unsicherer“ Benutzung von RSA (oder anderer Einwegverfahren)

5 Nachrichtenauthentifizierung, Datenintegrität

Szenario: Alice möchte Bob eine Nachricht schicken, an der Leitung lauscht Oscar, er kann den Verkehr aber auch manipulieren! Bob möchte in der Lage sein, nachzuprüfen, ob eine Nachricht, die ihn erreicht, von Alice kommt. In diesem Kapitel teilen sich beide einen geheimen Schlüssel.

Idee: Alice verschickt eine Nachricht x zusammen mit einer von dem Schlüssel und der Nachricht abhängigen *Markierung* (oder *message tag, message digest*) $m := T(k, x)$. Bob überprüft dann, ob $m == T(k, x)$ gilt.

Beispiel: Verschlüsseln tut's nicht! Annahme: Wir benutzen AES mit CTR, Alice schickt Bob eine Überweisung, die aus 128 Bits bestehen soll und bei der die letzten zehn Bits den Betrag angeben sollen. Wenn Oscar dann (...`020`, $y_{-1}y$) sehen würde, könnte er daraus (...`200`, $y_{-1}(y \oplus 0^{118}00011011100)$) machen!

Definition: Ein *Nachrichtenauthentifizierungsschema (NAS, message authentication code MAC)* S ist ein Tripel (M, K, T) bestehend aus einer endlichen Menge von Markierungen M, einer endlichen Menge K von Schlüssel und einem deterministischen polynomiellen Markierungsalgorithmus $T(k : K, x : 2^*) : M$. Ein Tupel (x, m) heißt *Nachrichten-Markierungspaar (NMP)*; es heißt *gültig bezüglich* k, falls $m = T(k, x)$ ist.

5.1 Algorithmische Sicherheit und Fälscher

Vorbemerkung: Wiederholungsangriffe sind (vorerst) erlaubt, der Angreifer kann sich zudem konkrete gültige NMPs beschaffen. Der Angreifer ist gut, wenn er zusätzliche gültige NMPs erzeugen kann.

DEFINITION: Ein (n, q, t)-Fälscher A ist ein randomisierter Algorithmus, dessen Laufzeit durch t beschränkt ist und der höchstens q mal den Parameter T aufruft und dabei höchstens n Bits übergibt.

Sei $S = (M, K, T)$ ein MAC. Der Erfolg von A gegen S wird dann durch das folgende Experiment $E(A, S)$ beschrieben:

```
flip k ∈ K;
(x, m) = A(T(·, k));
if T(x, k) == m ∧ A hat den Parameter
     nicht mit dem Argument x aufgerufen
  return gültig;
else
  return ungültig;
```

Der *Erfolg* ist nun

$$\mathrm{suc}(A, S) = P(E(A, S) = \textit{gültig})$$

Die *Unsicherheit*:

$$\mathrm{insec}(n, q, t, S) = \max\{\mathrm{suc}(A, S) \mid A \ (n, q, t)\text{-Fälscher}\}$$

Für $\varepsilon > 0$ heißt S (n, q, t, ε)-*sicher*, falls $\mathrm{insec}(n, q, t, S) < \varepsilon$, sonst (n, q, t, ε)-*unsicher*.

BEMERKUNG: Sicherheitsbegriff auch deshalb stark, weil der Angreifer sich die Nachricht aussuchen kann, zu der er eine Markierung bestimmen möchte. Sprechweise: *existentielle Fälschung.*

BEISPIEL:

- Sei S_0 ein Blockkryptosystem der Länge l. Dann wählen wir

$$\begin{aligned} S &= (\{0,1\}^l, \{0,1\}^l, T) \\ T(k, x_0 x_1 \ldots x_{m-1}) &= e_k(x_0) \oplus \ldots \oplus e_k(x_{m-1}) \end{aligned}$$

  Betrachte dann einfach den Angriff

$$\frac{A(T)}{\texttt{return } (0^{2l}, 0^l)}$$

  Dieser Angreifer hat $\mathrm{suc}(A, S) = 1$!

- Verbesserung: Durchnumerieren der Blöcke; wir spendieren r Bits für eine Nummer. Das heißt, $x = x_0 x_1 \ldots x_{m-1}$ mit $|x_i| = n - r$ und $m \leq 2^r$

erhält die Markierung

$$e_k(\langle 0\rangle_r\, x_0) \oplus e_k(\langle 1\rangle_r\, x_1) \oplus \ldots \oplus e_k(\langle m-1\rangle_r\, x_{m-1})$$

wobei $\langle i\rangle_r$ die Zahl i in Binärdarstellung der Länge r sein soll. Der alte Angriff funktioniert nicht mehr, aber ein neuer!

5.2 CBC-MAC

Idee ist, den letzten Block der verschlüsselten Nachricht als Marke zu benutzen: Sei $S_0 = (\{0,1\}^l, K, \{0,1\}^l, e, d)$ ein l-Blockkryptosystem. Dann ist S_0-CBC-MAC der l-MAC $(\{0,1\}^l, K, T)$ mit $T(x,k) = y_{n-1}$, falls $x = x_0 \ldots x_{n-1}$ mit $x \in \{0,1\}^l$, $y_{-1} = 0^l$ und $y_i = e_k(y_{i-1} \oplus x_i)$ für alle $i < n$.

Fakt: Der S_0-CBC-MAC ist sicher, sofern er nur auf Nachrichten fester Länge angewandt wird. Genauer: Bei Anwendung auf Nachrichten der Länge $l \cdot n$ ergibt sich:

SATZ: Seien $l, n, q, t \geq 1$, so daß $q \cdot n \leq 2^{\frac{l+1}{2}}$. Sei S_0 ein l-Blockkryptosystem. Dann gilt:

$$\text{insec}(lnq, q, t, S_0\text{-CBC-MAC}(n)) \leq \text{insec}(q', t', S_0) + \frac{2q^2n^2+1}{2^l}$$

mit $q' = nq$ und $t' = t + \mathcal{O}(nlq)$. Für $1 \leq q \leq 2^{\frac{l+1}{2}}$ und $n \geq 2$ gilt:

$$\text{insec}(nql, q, t, S_0\text{-CBC-MAC}(n)) \geq 0.3 \cdot \frac{(q-1)(q-2)}{2^l}$$

Zum Beweis der unteren Schranke konstruieren wir einen Fälscher: Lege

$a_i \in \{0,1\}^l$ für $i < q$ fest, paarweise verschieden.

$A(T(\cdot,k))$

```
for i = 0 to q − 1
    r_i := flip(l);
    x_i := a_i · r_i · 0^{l(n−2)};
    m_i := T(x_i, k);
find i and j so that
    (i ≠ j) ∧ (m_i = m_j)
if no such values exist
    return;
else   a' ∈ ({0,1}^l \ {0^l});
    x'_i = a_i · (r_i ⊕ a') · 0^{l(n−2)};
    x'_j = a_j · (r_j ⊕ a') · 0^{l(n−2)};
    m'_i = T(x'_i, k);
    return (x'_j, m'_i);
```

Zu beachten:

$$T(a \cdot r \cdot 0^{l(n-2)}, k) = e_k(e_k(e_k(\ldots(e_k(a) \oplus r)\ldots)))$$

Also gilt $T(a \cdot r \cdot 0^{l(n-2)}) = T(a' \cdot r' \cdot 0^{l(n-2)})$ genau dann, wenn $e_k(a) \oplus r = e_k(a') \oplus r'$ ist. Bei der Anwendung auf x'_i und x'_j gilt

$$e_k(a_i) \oplus r_i \oplus a = e_k(a_j) \oplus r_j \oplus a$$

da gilt: $e_k(a_i) \oplus r_i = e_k(a_j) \oplus r_j$. Damit liefert der Fäscher im `else`-Fall ein gültiges NMP. Außerdem sorgt die Addition von $a \neq 0^l$ dafür, daß der Fälscher ein NMP (x'_j, m'_i) zurückgibt, von dem x'_j noch nicht abgefragt wurde. Insgesamt erhält man, daß die Erfolgswahrscheinlichkeit gleich der Kollisionswahrscheinlichkeit ist. ■

Bemerkung: Egal welches BKS dem CBC-MAC zugrunde liegt, ist er unsicher, wenn die Nachrichtenlänge variieren darf – siehe Übung.

5.3 Kryptogrpahische Hashfunktionen

Definition: Eine l-Hashfunktion ist eine Funktion $h\colon \{0,1\}^* \to \{0,1\}^l$, die die Eigenschaft hat, daß es „praktische unmöglich“ ist, eine Kollision zu finden, d.h. (x, x') mit $x \neq x'$ und $h(x) = h(x')$.

Bemerkungen:

- Dies ist **keine formale Definition**, in späteren formalen Analysen werden wir idealisierte Hashfunktionen benutzen: Jeder Wert wird zufällig gewählt, mit der Annahme, daß ein bereits abgefragter Wert erneut den gleichen Hashwert erhält.
- Notwendige Voraussetzung: l ist genügend groß, damit ein Geburtstagsangriff nicht zum Erfolg führt. Übliche Werte für l: 128, 160
- Heute verwendete Hashfunktionen: MD5, SHA-1, RIPE MD-160, ...

Grundlegende Idee zur Konstruktion von Hashfunktionen: Ausgehend von einer so genannten Kompressionsfunktion

$$f\colon \{0,1\}^b \times \{0,1\}^l \to \{0,1\}^l$$

Häufig wird auch $f\colon 2^m \to 2^l$ mit $m > l$ angegeben, wobei obiges dann $m = b + l$ entsprechen würde. Sei x gegeben, die Berechnung des Hashcodes h in einer (b, l)-Hashfunktion:

1. $x' = x \langle |x| \rangle$
2. Verlängern von x' zu $\bar{x} = x_0 \dots x_{n-1}$ auf ein Vielfaches von b *(Padding)*
3. $h = h_{n-1}$ mit $h_i = f(x_i, h_{i-1})$ für $i < n$, wobei h_{-1} fest gewählt

Eine konkrete iterierte $(512, 160)$-Hashfunktion ist SHA-1 (als Nachfolger von SHA), die Strings der Länge $\leq 2^{64} - 1$ hasht.

Gegeben sei $x \in \{0,1\}^{\leq 2^{64}-1}$. Hilfsmittel zur Berechnung:

- Hilfsoperationen auf Wörtern (d.h. 32 Bit): bitweises AND: $\wedge$, bitweises OR: $\vee$, bitweises XOR: $\oplus$, bitweises NOT: $\neg$, Addition modulo 2^{32}: $+$ und Linksrotation um s Positionen: rot^s
- Padding:

  $$\text{SHA-1}{-}\mathrm{PAD}(x) = x \cdot 1 \cdot 0^d \cdot l \text{ mit } d = 447 - |x| \mod 512, l = \langle |x| \rangle_{64}$$

  Die Länge des Ergebnisses modulo 512 ist

  $$|x| + 1 + 447 - |x| + 64 \bmod 512 = 448 + 64 = 0$$

- Zusätzliche Funktionen $f_0, \dots, f_{79}$ mit

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & \text{falls } 0 \leq t < 20 \\ B \oplus C \oplus D & \text{falls } 20 \leq t < 40, 60 \leq t < 80 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{falls } 40 \leq t < 60 \end{cases}$$

- Zusätzliche Konstanten $K_0, \ldots, K_{79}$ mit

$$K_t = \begin{cases} 5A827999 & \text{falls} \quad 0 \leq t < 20 \\ 6ED9EBA1 & \text{falls} \quad 20 \leq t < 40 \\ 8F1BBCDC & \text{falls} \quad 40 \leq t < 60 \\ CA62C1D6 & \text{falls} \quad 60 \leq t < 80 \end{cases}$$

Ablauf des Verfahrens:

5.4 Schlüsselbasierte Hashfunktionen

Ziel ist die Konstruktion von MACs, Problem jedoch: die Hashfunktionen haben keine Schlüssel. Erste **Idee:** Man könnte den Initialisierungsvektor variieren!

DEFINITION: Zu einer (b, l)-Kompressionsfunktion f betrachten wir die Familie $\{F_{k,f}\}_{k \in \{0,1\}^l}$ von iterierten (b, l)-Hashfunktionen. Eine Familie $\{F_k\}_{k \in \{0,1\}^l}$ von l-Hashfunktionen heißt *Familie von schlüsselbasierten Hashfunktionen*, falls sie wie oben beschrieben durch eine (b, l)-Kompressionsfunktion induziert wird.

Sicherheitsbegriff für schlüsselbasierte Hashfunktionen:

DEFINITION: Für n, q, t ist ein (n, q, t)-*Angreifer* A ein zufallsgesteuerter Algorithmus mit Prozedur-Parameter, dessen Laufzeit durch t beschränkt ist und der höchstens q mal den Prozedurparameter aufruft und dabei insgesamt höchstens n Bits abfragt. Die Ausgabe von A ist ein Tupel (m, m').

Sei $F := \{F_k\}_{k \in \{0,1\}^l}$ wie oben. Der *Erfolg* von A gegen F wird durch das folgende Experiment bestimmt:

$E(A, F)$

$k = \texttt{flip}(l)$;
$(m, m') = A(F_k(\cdot))$;
`if` $m \neq m' \wedge F_k(m) = F_k(m')$
  `return` *Kollision*;
`else`
  `return` *keine Kollision*;

Dann ist der *Erfolg*:

$$\text{suc}(A, F) = P(E(A, F) = \textit{Kollision})$$

Es sei f eine (b,l)-Kompressionsfunktion und $F_k = F_{k,f}$ die zugehörige induzierte Familie, Dann ist f-NMAC der MAC $(\{0,1\}^l, \{0,1\}^l \times \{0,1\}^l, T)$ mit $T(x,k) = F_{k1}(F_{k2}(x))$ für $x \in \{0,1\}$ und $(k_1,k_2) \in \{0,1\}^l \times \{0,1\}^l$.

SATZ: Es sei f eine (b,l)-Kompressionsfunktion und es sei F wie oben. Weiter sei S_f der durch f indizierte (b,l)-MAC und S der durch f induzierte NMAC. Dann gilt für alle $n,q,t \geq 0$:

$$\text{insec}(n,t,q,S) \leq \text{insec}(n,q+1,t,F) + \text{insec}(bq,q,t,S_f)$$

5.5 HMAC

Problem bei NMAC ist, daß der Initialisierungsvektor jeweils verändert werden müßte (kompliziert bei Hardware!). Deshalb: Integration des Schlüssels in die Nachricht. Es sei F eine iterierte (b,l)-Hashfunktion, bei der b und l durch 8 teilbar sind. Dann ist der f-HMAC der MAC $(\{0,1\}^l, \{0,1\}^l, T)$ mit

$$T(x,k) = F(\bar{k} \oplus \mathit{opad} \cdot F((\bar{k} \oplus \mathit{ipad}) \cdot x))$$

wobei $\bar{k} = k0^{b-l}$, $\{0,1\}^l = 363636363636\ldots$ und $\mathit{ipad} = 5C5C5C5C\ldots$.

HAMC im Vergleich zu NMAC:

Sei f die zugrundeliegende (b,l)-Kompressionsfunktion und T_N die Markierungsfunktion von f-NMAC und T_H entsprechend vom F-HMAC. Mit $k_1 = f(\bar{k} \oplus \mathit{opad})$ und $k_2 = f(\bar{k} \oplus \mathit{ipad})$ gilt dann:

$$T_H(x,k) = T_N(x(k_1,k_2))$$

In der Praxis benutzt man SHA-1 oder MD-5 als Grundlage.

6 Digitale Signatur

Neues Szenario: Alice möchte Bob eine Nachricht schicken; Bob möchte prüfen, ob die Nachricht wirklich von Alice kommt. Alice hat dazu einen privaten Schlüssel $\hat{k}_A$, und es liegt öffentlich ein Schlüssel k_A bereit[5]. Alice schickt (x, s) für eine Nachricht x mit Signatur $s = S(x, \hat{k}_A)$ für einen Signaturalgorithmus S. Bob verwendet zur Verifikation einen Algorithmus $V(x, s, k_A)\colon \{\textit{gültig}, \textit{ungültig}\}$. Die Digitale Signatur ist also die **asymmetrische Variante eines MAC**.

Vorteil der digitalen Signatur ist eine stärkere Bindung der Unterschrift an die Nachricht und eine bessere Überprüfbarkeit der digitalen Signatur; jedoch ist die Abänderung von x einfacher, als Text oberhalb einer realen Unterschrift einzufügen.

6.1 Signierschemen

DEFINITION: Ein Signierschema ist ein Tupel (G, S, V, D) bestehend aus

- einem zufallsgesteuerten Schlüsselerzeugungsalgorithmus

$$G(p : \{0\}^*)\colon \{0,1\}^* \times \{0,1\}^*$$

- einem zufallsgesteuerten Signieralgorithmus

$$S(x : \{0,1\}^*, \hat{k} : \{0,1\}^*)\colon \{0,1\}^*$$

- einem deterministischen Verifikationsalgorithmus

$$V(x : \{0,1\}^*, s : \{0,1\}^*, k : \{0,1\}^*)\colon \{\textit{gültig}, \textit{ungültig}\}$$

- einem zufallsgesteuerten Bereichsalgorithmus

$$D(k : \{0,1\}^*)\colon \{0,1\}^*$$

[5] Im **Gesetz zur digitalen Signatur** (SigG): Der private Schlüssel heißt *Signaturschlüssel*, der öffentliche Schlüssel *Signaturprüfschlüssel*. *Zertifikate* sind Daten, die einen öffentlichen Schlüssel an die Identität einer Person binden. Zudem spricht man von *sicheren Signatur-Erstellungs-Einheiten*.

Dabei müssen noch folgende Anforderungen erfüllt werden:

- Die pessimalen Laufzeiten von S, V und D seien polynomiell beschränkt, die erwartete Laufzeit von G sei polynomiell beschränkt.
- Bei Eingabe des privaten Schlüssels $\hat{k}$ liefert D die Nachrichten, die signiert werden können.
- Für $l \geq 0$ liefert $G(\{0\}^l)$ Paare $(k, \hat{k})$ mit einem öffentlichen Schlüssel k und einem privaten Schlüssel $\hat{k}$.
- Für $(k, \hat{k}) = G(\{0\}^l)$ und $x = D(\hat{k})$ ist $S(x, \hat{k})$ die Signatur von x bezüglich $\hat{k}$.
- Für $(k, \hat{k}) = G(\{0\}^l)$ und $x = D(\hat{k})$ ist $V(x, s, k) = \textit{gültig}$, falls es eine unendliche Zufallsfolge γ gibt mit $S\langle\gamma\rangle(x, \hat{k}) = s$ und $V(x, s, k) = \textit{ungültig}$ sonst.
- Wir nennen (x, s) ein *Nachrichten-Signaur-Paar (NSP)*. Es heißt gültig bezüglich k, falls $V(x, s, k) = \textit{gültig}$.

Die Sicherheitsdefinition ist analog zur Definition bei MACs, einziger Unterschied: der Fälscher kennt k.

DEFINITION: Ein (n, q)-*Fälscher* ist ein zufallsgesteuerter Polynomzeitalgorithmus, der als Eingabe den mit dem privaten Schlüssel ausgestatteten Signieralgorithmus, den öffentlichen Schlüssel und den Sicherheitsparameter erhält und der höchstens q mal den Signieralgorithmus anfragt und dabei höchstens n Bits abfragt, d.h.

$$A(S(\cdot, \hat{k}), k, \{0\}^l)$$

Zur Definition der Sicherheit:

Es sei $\bar{S} = (G, S, V, D)$ ein Signierschema und A ein Fälscher wie oben. Dann wird der Erfolgt von A auf S durch das folgende Experiment $E(\bar{S}, A, \{0\}^l)$ bestimmt:

$E(\bar{S}, A, \{0\}^l)$

$(k, \hat{k}) = G(\{0\}^l);$
$(x, s) = A(S(\cdot, \hat{k}), k, \{0\}^l);$
`if` $V(x, s, k) =$ *gültig*
$\wedge$ A hat nicht $S(x, \hat{k})$ angefragt
`return` *Erfolg*;
`else`
`return` *Misserfolg*;

Erfolg des Fälschers:

Der Erfolg $\mathrm{suc}(\bar{S}, A, \{0\}^l)$ von A auf $\bar{S}$ bezüglich $\{0\}^l$ ist

$$P(E(\bar{S}, A, \{0\}^l) = \textit{Erfolg})$$

Wir nennen $\bar{S}$ (n, q)-*sicher*, falls für jeden (n, q)-Fälscher A gilt: $\mathrm{suc}(\bar{S}, A, \{0\}^l) \in \mathcal{O}(l^{-c})$ für alle $c > 0$ und ansonsten (n, q)-*unsicher*.

6.2 Einfaches Signieren mit Einwegfunktionen

Idee: Die „schon vorhandene“ Einwegfunktion mit Hintertür, die zum Verschlüsseln benutzt wird, wird auch zum Signieren benutzt; eine Einwegfamilie war ein Tupel $(\mathcal{F}, t, \mathcal{H})$. Dann sind zusätzliche Annahmen nötig:

- f_i und $h_{t(i)}$ haben denselben Definitionsbereich
- $f_i(h_{t(i)}(x)) = x$ (sonst war gefordert: $h_{t(i)}(f_i(x)) = x$, z.B. RSA besitzt aber auch die neue Eigenschaft)

Zugehöriges Signierschema: $S(x, t(i)) = g_j(x)$, und $V(x, s, i) =$ *gültig* genau dann, wenn $f_i(s) = x$.

Mögliche Sicherheit des Verfahrens beruht auf folgender Überlegung: Man kann zeigen,daß für jeden zufallsgesteuerten Polynomzeitalgorithmus $A(i, x)$ und für alle $c > 0$ gilt:

$$P(A(i, x) = h_{t(i)}(x) \,|\, i \in G(\{0\}^l), x \in D(i)) \in \mathcal{O}(l^{-c})$$

Problem: Die ist eine probabilistische Betrachtun, damit aber ein konkretes Signierschema unsicher ist, braucht man lediglich in der Lage zu sein, **ein** einzelnes NSP zu erzeugen!

$$\frac{A(S(\cdot, t(i)), i, \{0\}^l)}{\begin{array}{l} s = \texttt{flip}(l); \\ \texttt{return } (f_i(s), s); \end{array}}$$

Nach der Verschlüsselungseigenschaft $h_{t(i)}(f_i(x)) = x$ gilt $h_{t(i)}(f_i(s)) = s$, also gilt $V(f_i(s), s, i) = \textit{gültig}$! Formal: der Erfolg ist $\text{suc}(A, S, \{0\}^l) = 1$!

Bemerkungen:

- Der Fälscher produziert eine so genannte *existentielle Fälschung*.
- Für bestimmte Familien von Einwegfunktionen sind auch universelle Fälschungen möglich, d.h. der Fälscher kann zu einer vorgegebenen Nachricht eine gültige Signatur erzeugen. Dies gilt beispielsweise für RSA.[6]
- Mit einigem Aufwand lassen sich jedoch sichere Signierschemen aus Einwegfunktionen konstruieren, Vorgehensweise (entfernt) ähnlich zu der Vorgehensweise bei den asymmetrischen Verschlüsselungsverfahren. Insbesondere überlegt man sich zuerst, wie man ein Bit signieren kann. Die so gewonnenen Verfahren sind in der Regel sehr ineffizient.

6.3 Signieren mit Hashfunktionen

In der Praxis üblich sind **hybride Verfahren** (analog zur Vorgehensweise bei der asymmetrischen Verschlüsselung), bei denen nur ein Hashwert der Nachricht asymmetrisch signiert wird.

Beispiel: Signierschema aus PKCS#1 (Public Key Cryptography Standards, RSA Corporation): Wir benutzen RSA mit $|n| = 1024$ und SHA-1 (160 Bit), zunächst wird gehasht:

$$\text{PKCS} - \text{hash}(x) = 00001^{1024-160-12} \cdot 0^8 SHA - 1(x)$$

Signiert wird dann statt x nur $\text{PCKS} - \text{hash}(x)$ wie im letzten Paragraphen.

Vorteile: Das Verfahren ist effizient, es können beliebige Nachrichten signiert werden (keine Bereichsbeschränkung). Vor allem schlagen aber die oben beschriebenen beiden Angriffe und auch andere Verfahren hier fehl.

[6] siehe Übung, Hinweis: Multiplikativität

Nachteile: Geburtstagsangriffe auf die Hashfunktion übertragen sich auf das Signierschema, notwendige Voraussetzung ist also eine gute Hashfunktion. Zudem nutzt die verwendete Hashfunktion bzw. das Signierschema nur einen extrem kleinen Teil der verwendeten Einwegfunktion aus, genauer: Nur ein 2^{-864}-tel aller Möglichkeiten kommen vor. Damit lassen sich übliche Annahmen über RSA (z.B. unsere von oben) wohl kaum verwenden.

Ziel ist nun ein sicheres Signierschema dieser Art. zu entwickeln. Allgemeine Definition eines *Hash-Signierschemas (hash-and-decryp-scheme)*:

DEFINITION: Es sei $\bar{S} = (G, S, V, F)$ ein Signierschema, K die Menge der in $\bar{S}$ verwendeten öffentlichen Schlüssel und $H(x, k : K)$ ein deterministischer Hash-Algorithmus, so daß für alle $l > 0$ und $(k, \hat{k}) \in G(\{0\}^l)$ der Bildbereich von $H(\cdot, K)$ eine Teilmenge ist vom Definitionsbereich $S(\cdot, \hat{k})$ ist. Dann ist $(G, S_{\text{hash}}, V_{\text{hash}})$ das durch $\bar{S}$ und H induzierte *Hash-Signierschema* gegeben durch

$$\begin{aligned} S_{\text{hash}}(x, \hat{k}) &= S(H(x, k), \hat{k}) \\ V_{\text{hash}}(x, s, k) &= V(H(x, k), s, k) \end{aligned}$$

Im Beispiel oben ignoriert die Hashfunktion das k. Der Fälscher hat jetzt auch Zugriff auf H und (wie immer) auf k. Zusätzlicher Parameter in der Sicherheitsdefinition: Ein $(n, q_{\text{sig}}, q_{\text{hash}}, l, t)$-*Fälscher* enthält q_{sig} und q_{hash}, die die Zugriffe auf S und auf H angeben, wobei die Zugriffe auf H innerhalb von S mitzählen. t ist die Laufzeit des Fälschers, l der Sicherheitsparameter.

Sichere Variante: sichere Hashfunktion und eine Hashfunktion, die den gesamten Definitionsbereich der Einwegfunktion ausschöpft *(full domain hash, FDH)*.

Wir zeigen, daß $\bar{S} = (S_{\mathsf{RSA}}, H_{\mathsf{RSA}})$ sicher ist. Dabei nehmen wir an, daß $H(\cdot, (n, e))$ eine Hashfunktion ist, die $\mathbb{Z}_n$ ausschöpft. Idealisierte Annahme: $H(\cdot, (n, e))$ verhält sich wie ein zufälliges Orakel, d.h. $H(\cdot, (n, e))$ liefert Zufallswerte mit der Einschränkung, daß bei zwei Abfragen mit gleichen Parametern derselbe Wert zurückgegeben wird. Ansatz ist nun, die Sicherheit in Bezug auf die Sicherheit von RSA als Einwegfunktion zu bewerten.

SATZ: Für $\mu, q_{\text{sig}}, q_{\text{hash}}, l, t \geq 0$ mit $q_{\text{hash}} > q_{\text{sig}} + 1$ und $\bar{S}$ wie oben gilt:

$$\text{insec}(\mu, q_{\text{sig}}, q_{\text{hash}}, \{0\}^l, t) \leq q_{\text{hash}} \, \text{insec}(\{0\}^l, t', \mathsf{RSA})$$

mit $t' = t + q_{\text{hash}} \cdot \mathcal{O}(l^3)$.

BEWEIS: Wenn das Signierverfahren unsicher ist, dann ist auch RSA invertierbar! Aufgabe ist also, aus einem Fälscher einen Invertierer zu konstruieren.

Grundlegende Idee: Der Invertierer simuliert den Fälscher und versucht, an der richtigen Stelle y als Hashwert vorzugeben. Annahme dazu: Falls der Fälscher erfolgreich ist, hat er die Hashfunktion an der entsprechenden Stelle abgefragt.

Invertierer$(y, (n, e))$

$j := -1$;
$i :=$ `flip`(q_{hash});
simuliere Fälscher, dabei:
  falls der Fälscher eine Anfrage der Form $H(x, (n, e))$ stellt
    benutze $H - \text{Sim}(x)$ anstelle von $H(x, (n, e))$;
  falls der Fälscher eine Anfrage der Form $S(x, (n, d))$ stellt
    benutze $S - \text{Sim}(x)$ anstelle von $S(x, (n, d))$;
$(m, s) :=$ vom Fälscher produziertes NSP;
`return` s;

Zur Simulation der Hashfunktion $H - \text{Sim}(x)$:

$H - \text{Sim}(x)$

$A_{\text{msg}}[++j] := x$;
finde kleinstes r mit $A_{\text{msg}}[r] == x$;
`if` $r < j$
  $A_h[j] := A_h[r]$;
  $A_S[j] := A_S[r]$;
`else`
  `if` $j == i$
    $A_h[j] := y$;
    $A_S[j] := \perp$;
  `else`
    $A_S[j] :=$ `flip`$(\mathbb{Z}_n)$;
    $A_h[j] := (A_S[j])^e \bmod n$;

Zur Simulation der Signaturfunktion $S - \text{Sim}(x)$:

$S - \text{Sim}(x)$

$H - \text{Sim}(x)$;
`return` $A_S[j]$;

BEMERKUNG: Problem ist, daß die Unsicherheit um den Faktor q_{hash} größer wird (i wird geraten). Durch geschickte Konstruktion kann das behoben werden. ■

7 Protokolle

Vorbemerkung: Bislang eigentlich nur „Sicherung“ von Dokumenten (Vorstellung von Kommunikation zwischen Alice und Bob war dabei allerdings hilfreich):

- **Vertraulichkeit**: Wir bearbeiten einen Text so, daß er nur von einer bestimmten Person gelesen werden kann.
- **Integrität/Authentizität**: Wir fügen zu einem Text eine Marke bzw. Signatur hinzu, so daß man anhand der Marke/Signatur feststellen kann, ob der Text verändert wurde, und ob er vom richtigen stammt.

In diesem Kapitel geht es um **vertrauliche und authentische Kommunikation**, z.B. im Internet.

Randbedingungen: Die Kommunikationspartner sind nicht durch konkrete Leitungen miteinander verbunden, sondern schicken Nachrichten paketweise durch das Netzwerk. Wir gehen sogar davon aus, daß das Netzwerk vollständig durch unseren Angreifer kontrolliert wird – oder: Das Netzwerk ist der Angreifer!

Weitere Annahmen: Die Beteiligten sind *idealisierte Rechner* (Turingmaschinen), wir arbeiten auf der Bitebene: Zeitliche Aspekte (z.B. timing attacks) werden fast völlig außer Acht gelassen; physikalische Effekte (z.B. Wärmeabstrahlung) vollständig.

Ziel ist sichere Kommunikation in diesem Modell im bisherigen Sinne: ein Angreifer kann mit vertretbarem Aufwand nur mit sehr geringer Wahrscheinlichkeit erfolgreich sein; **genauer**: Sicherheit aufgrund der bekannten schwachen Annahmen (z.B. Sicherheit von AES als Blockchiffre, Sicherheit von RSA als Einwegfunktion).

Erster Ansatz zum Protokollbegriff: Satz von miteinander kommunizierenden Programmen, wobei ein Programm wie im üblichen Sinne verstanden wird erweitert um folgende Komponenten:

- **Anweisung zum Senden**: `!x` bedeutet, daß die Nachricht x ins Netzwerk (zum Angreifer) geschickt wird. `!B:x` bedeutet, daß x an B gehen soll.
- **Empfangen**: `y := ?` bedeutet, daß y die nächste Nachricht zugewiesen wird; `y := ?B` ist die nächste an B adressierte Nachricht.

Kommunikationsparameter sind Variablen für Namen bzw. Identitäten der Kommunikationspartner; wobei ein *Kommunikationspartner* vorstellbar ist

als eine Instanz eines Programms bzw. ein Programmlauf – dann ist ein Kommunikationsparameter beispielsweise eine Prozess-ID.

Ein Programm ist *korrekt*, wenn eine Ausführung der Programme mit zueinander passenden Kommunikationsparametern dazu führt, daß alle Programme terminieren und das Gewünschte ausgeben (unter der Voraussetzung, daß der Angreifer Nachrichten nicht anfasst).

7.1 Beispiele für angreifbare Schlüsselaustauschprotokolle

Grundsätzliche Vorgehensweise bei vielen Protokollen: Zu Beginn einer *Sitzung* (Protokolllauf) – vor dem eigentlichen Datenaustausch – einigen sich die Kommunikationspartner auf einen neuen symmetrischen *Sitzungsschlüssel*, der genau diese Sitzung absichern soll. Deshalb studieren wir zuerst *Schlüsselaustauschprotokolle*.

Gründe für die Benutzung von Schlüsselaustauschprotokollen

- Symmetrische Verschlüsselungsverfahren sind wesentlich effizienter.
- Die symmetrische Verschlüsselung „schont“ die asymmetrischen Schlüssel: Weniger Nachrichten bedeuten weniger Angriffsfläche!
- Wird ein Sitzungsschlüssel kompromittiert, ist nur diese Sitzung betroffen.

Zur **Korrektheit und Sicherheit** von Schlüsselaustauschprotokollen zwischen A und B:

- Am Ende des Protokolllaufs haben sich A und B auf einen Schlüssel geeinigt, was sie dadurch signalisieren, daß sie den Schlüssel beide ausgeben.
- Keiner außer den beiden kenn diesen Schlüssel (mit hoher Wahrscheinlichkeit).
- Sollte ein Angreifer ins Spiel kommen, dürfen die Programme vorzeitig abbrechen.

Beispiele: Einfache Protokolle:

1. Einfachste Idee:

$P_0(A,B)$	$P_1(B)$
$k := \texttt{flip}(l)$; $y := e_{k_B}(Ak)$; $!B\colon y$; `return` k, B;	$y = ?B$; $(A,k) := d_{\hat{k}_B}(y)$; `return` A, k;

   Problem: Jeder andere außer Alice könnte die Nachricht geschickt und sich als Alice ausgegeben haben!

   Kurznotation:

   - für das Protokoll: $A \to B\colon e_{k_B}(Ak)$
   - für das Problem: $I(A) \to B\colon e_{k_B}(Ak)$

2. Folgendes Protokoll:

$$A \to B\colon e_{k_B}(Ak)S_{\hat{k}_A}(Ak)$$

   Probleme:

   (a) *Wiederholungsangriff (replay attack):* Ein Angreifer kann später dieselbe Nachricht noch einmal schicken und sich so als A ausgeben:

$$A \to B\colon e_{k_B}(Ak)S_{\hat{k}_A}(Ak); \quad I(A) \to B\colon e_{k_B}(Ak)S_{\hat{k}_A}(Ak)$$

   (b) Ein Angreifer kann eine Nachricht, die an ihn geschickt wurde, weiterschicken unter As Absender:

$$A \to I\colon e_{k_I}(Ak)S_{\hat{k}_A}(Ak); \quad I(A) \to B\colon e_{k_B}(Ak)S_{\hat{k}_A}(Ak)$$

7.2 Das Protokoll von Needham und Schroeder

Vereinfachte Variante eines Protokolls von Needham und Schroeder aus dem Jahr 1978, hier als Schlüsselaustauschprotokoll vorgestellt.

$$\begin{array}{lll} (1) & A \to B\colon & e_{k_B}(kA) \\ (2) & B \to A\colon & e_{k_A}(kN) \\ (3) & A \to B\colon & e_{k_B}(N) \end{array}$$

Bemerkung: Allgemeine Techniken gegen Wiederholungsangriffe: Zum einen kann man einen Zeitstempel mitschicken (wenig benutzt), oder man benutzt eine Zufallszahl N (Abkürzung für *Nonce*, number used once): Man

schickt einen zufälligen Bitstring und erwartet, daß dieser wieder zurückgeschickt wird. Gleichzeitig werden Nonces zur Authentifizierung benutzt.

Im NEEDHAM-SCHROEDER-Protokoll werden zwei Nonces benutzt zur gegenseitigen Authentifizierung; gleichzeitig wird ein Nonce als Sitzungsschlüssel benutzt.

Angriff:

$$\begin{array}{lllll}
(1) & A \to I\colon & e_{k_I}(kA) & & \\
(1') & & & I(A) \to B\colon & e_{k_B}(kA) \\
(2') & & & B \to I(A)\colon & e_{k_A}(kN) \\
(2) & I \to A\colon & e_{k_A}(kN) & & \\
(3) & A \to I\colon & e_{k_I}(N) & & \\
(3') & & & I(A) \to B\colon & e_{k_B}(N)
\end{array}$$

Beide Protokollläufe sind ordnungsgemäß, aber Bob kann annehmen, daß k ein Sitzungsschlüssel ist, der nur Alice und ihm bekannt ist und daß er mit Alice kommuniziert.

Eine von LOWE vorgeschlagene Modifikation (NSL-Protokoll) fügt in der zweiten Nachricht Bobs Absender ein und verhindert damit den Angriff:

$$\begin{array}{lll}
(1) & A \to B\colon & e_{k_B}(kA) \\
(2) & B \to A\colon & e_{k_A}(kNB) \\
(3) & A \to B\colon & e_{k_B}(N)
\end{array}$$

NSL ist in einem sehr starken Sinn sicher (analog zu Sicherheitsbeweisen aus vorherigen Kapiteln)!

Wir betrachten weitere Varianten:

- Eine mögliche Sparversion von NSL:

$$\begin{array}{lll}
(1) & A \to B\colon & e_{k_B}(kA) \\
(2) & B \to A\colon & e_{k_A}((k \oplus B)N) \\
(3) & A \to B\colon & e_{k_B}(N)
\end{array}$$

- Angriff:

$$\begin{array}{lllll}
(1) & A \to I\colon & e_{k_I}(kA) & & \\
(1') & & & I(A) \to B\colon & e_{k_B}((k \oplus B \oplus I)A) \\
(2') & & & B \to I(A)\colon & e_{k_A}(\underbrace{((k \oplus B \oplus I) \oplus B)}_{k \oplus I} N) \\
(2) & I \to A\colon & e_{k_A}((k \oplus I)N) & & \\
(3) & A \to I\colon & e_{k_I}(N) & & \\
(3') & & & I(A) \to B\colon & e_{k_B}(N)
\end{array}$$

- Noch eine letzte Variante von NSL:

$$\begin{array}{lll} (1) & A \to B: & e_{k_B}(AN) \\ (2) & B \to A: & e_{k_A}(NkB) \\ (3) & A \to B: & e_{k_B}(k) \end{array}$$

- Und ein möglicher Angriff:

$$\begin{array}{lllll} (1) & I(A) \to B: & e_{k_I}(AI) & & \\ (2) & B \to I: & e_{k_A}(IkB) & & \\ (1') & & & I \to A: & e_{k_A}(IkB) \\ (2') & & & A \to I: & e_{k_I}(kBk'A) \\ (3) & I(A) \to B: & e_{k_I}(k) & & \end{array}$$

  Ian hat sich nun Bob gegenüber als Alice ausgegeben! Aber: Hier entstehen sogenannte *Typfehler*: Im Schritt (1) wird anstelle eines Nonces ein Name geschickt; im Schritt $(1')$ statt eines Nonces eine Konkatenation aus einem Schlüssel und einem Namen geschickt.

Bemerkung: Zur allgemeinen Vorgehensweise bei der Sicherheitsanalyse:

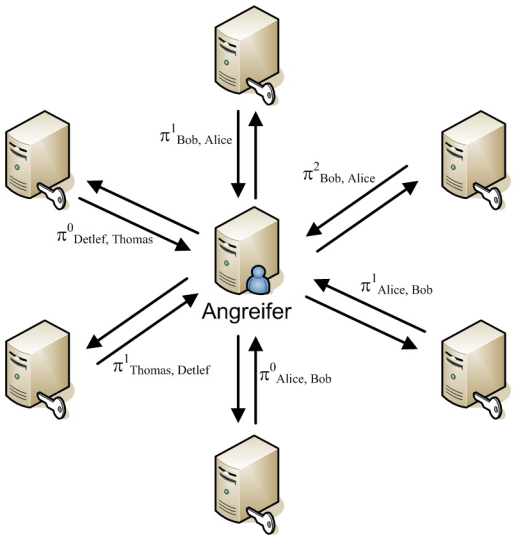
- Erste Möglichkeit ist die kryptographische/komplexitätstheoretische Analyse (wie bisher); dies ist *wahnsinnig* kompliziert!
- Zweite Möglichkeit ist die formale Ebene – unter der Annahme, daß Nachrichten Terme sind und Verschlüsselung etc. perfekt funktionieren:
  1. Vollautomatische Analysen sind praktisch, aber nur eingeschränkt einsetzbar – siehe Vorlesung nächstes Semester!
  2. Halbautomatische Analysen werden beispielsweise unter Benutzung von Theorembeweisern durchgeführt.
- Ideal wäre eine Verknüpfung, die sagt: Wenn im idealen System nichts schief geht, dann geschieht auch im realen (komplexitäts-theoretisch) Sinn nicht.

Unter `www.lsv.ens-cachan.fr/spore` findet man ein Verzeichnis von interessanten kryptographischen Protokollen!

7.3 Ein Sicherheitsbegriff für Authentifizierungs- und Schlüsselaustauschprotokolle

Quelle: Bellare, Rogaway: Entity Authentication and Key Distribution (Crypto '93)

Wir stellen uns den Angreifer nun stärker vor, er steuert alle Protokolle/Programme; von jedem Protokoll gibt es mehrere Instanzen:



Dabei werden die einzelnen Verbindungen/Protokollläufe mit $\pi^s_{i,j}$ gekennzeichnet, wobei i und j für die Kommunikationspartner stehen und s für die Nummer des Protkolllaufs, etwa $\pi^0_{\text{Alice,Bob}}$.

DEFINITION: Protokolle werden durch Programme beschrieben – Eingabeparameter dieser Programme:

- $\{0\}^k$ als Sicherheitsparameter
- i ist der Name (die Identität) des Senders; Annahme dabei: Es gibt eine vorbestimmte Menge $I \subseteq \{0,1\}^k$ von Identitäten.
- j ist die Identität des gewünschten Empfängers $\in I$
- a ist die geheime Information des Senders $\{0,1\}^*$
- κ ist die bisher abgelaufene Kommunikation ($c$onversation) $\{0,1\}^*$
- r ist der Zufallsbitstring des Senders

Die Ausgabe ist dann

$$\pi(\{0\}^k, i, j, a, \kappa, r) = (m, \delta, \alpha)$$

- m ist die nächste zu sendende Nachricht $\in \{0,1\}^* \cup \{\star\}$ (bei $\star$ wird keine Nachricht gesendet)
- δ ist Entscheidung $\in \{A, R, \star\}$ für *Accepted* (A), *Rejected* (B) und *unentschieden* ($\star$)
- α ist eine private, nicht-öffentliche Ausgabe $\in \{0,1\}^* \cup \{\star\}$ (z.B. Vermutung über ausgetauschten Schlüssel; $\star$ heißt wiederum keine Ausgabe)

R ist die Anzahl der Runden eines Protokolls (d.h. die Anzahl der geschickten Nachrichten; polynomiell im Sicherheitsparameter – hier meistens fest); zusätzlicher Bestandteil eines Protokolls ist ein LL-Schlüsselgenerator:

Ein LL-Schlüsselgenerator *(longlived bzw. langlebig)* ist ein Algorithmus $G(1^k, \iota, r_G)$ mit $\iota \in I \cup \{E\}$, wobei E der Angreifer ist und r_G ein Zufallsbitstring.

Der Angreifer ist ein probabilistischer Algorithmus

$$E(\{0\}^k, a_E, r_E, \{\pi^s_{i,j}\}_{i,j \in I; s \in \mathbb{N}})$$

Dabei steht $\pi^s_{i,j}$ für die s-te Kopie des Protokolls mit Sender/Empfänger i, j und $s \in \mathbb{N}$ (s erinnert an Sessions).

Die Interaktion zwischen E und den $\pi^s_{i,j}$ findet jetzt über Nachrichten der Form (i, j, s, x) statt mit folgender Beschreibung eines gemeinsamen Laufes von E und den $\pi^s_{i,j}$:

- Wähle $r_G \in \{0,1\}^\omega$ und setze $a_i = G(\{0\}^k, i, r_G)$ für alle $i \in I$ und $a_E = G(\{0\}^k, E, r_G)$.
- Wähle Zufallsstrings r_E, $r^s_{ij} \in \{0,1\}^\omega$.
- Setze $\kappa^s_{i,j} = \varepsilon$ für alle $i, j \in I$.
- Lasse E laufen mit Eingabeparametern $\{0\}^k$, a_E, r_E und

$$\left\{ \pi^s_{i,j}(\{0\}^k, i, j, a_i, \square, r^s_{ij}) \;\middle|\; i, j \in I; s \in \mathbb{N} \right\}$$

wobei gilt: Wenn E die Nachricht (i, j, s, x) erzeugt, wird $\kappa_{i,j}^s x$ in $\square$ von $\pi_{i,j}^s$ eingesetzt, $\pi_{i,j}^s$ berechnet eine Ausgabe (m, δ, α); E erhält dann (m, δ) und $\kappa_{i,j}^s$ wird durch $\kappa_{i,j}^s x$ ersetzt.

Als Zeitraster für die Kommunikationsprotokolle verwenden wir $\mathbb{N}$. Wir suchen nun zueinander passende Nachrichten (matching conversations). Sei E ein Angreifer. Der Nachrichtenverlauf eines Protokollinstanz $\pi_{i,j}^s$ wir beschrieben durch eine Folge

$$K = (\tau_1, \alpha_1, \beta_1)(\tau_2, \alpha_2, \beta_2) \dots (\tau_m, \alpha_m, \beta_m)$$

mit der Bedeutung, daß zum Zeitpunkt τ_p der Protokolllauf $\pi_{i,j}^s$ die Eingabe α_p erhielt und Ausgabe β_p lieferte. Dabei ist $\tau_p < \tau_{p+1}$. Ist $\alpha_p = \varepsilon$, so bezeichnen wir dieses Tupel als Initiatorinstanz; andernfalls als Antwortinstanz.

DEFINITION: *Matching Conversations* für den Fall $R = 2\varrho - 1$:

Seien $\pi_{i,j}^s$ und $\pi_{j,i}^t$ zwei Protokolläuft mit Nachrichtenverlauf K und K'.

- K' ist eine *matching conversation* zu K, falls $\tau_0 < \tau_1 < \ldots < \tau_{R-1}$ und $\alpha_1, \beta_1, \ldots, \alpha_\varrho, \beta_\varrho$ so existieren, daß K beginnt mit

  $$(\tau_0, \varepsilon, \alpha_1)(\tau_2, \beta_1, \alpha_2) \dots (\tau_{2\varrho-4}, \beta_{\varrho-2}, \alpha_{\varrho-1})(\tau_{2\varrho-2}, \beta_{\varrho-1}, \alpha_\varrho)$$

  und K beginnt mit

  $$(\tau_1, \alpha_1, \beta_1)(\tau_3, \alpha_2, \beta_2) \dots (\tau_{2\varrho-3}, \alpha_{\varrho-1}, \beta_{\varrho-1})$$

- K ist eine *matching conversation* zu K', falls $\tau_0 < \tau_1 < \ldots < \tau_{R-1}$ und $\alpha_1, \beta_1, \ldots, \alpha_\varrho, \beta_\varrho$ so existieren, daß K' beginnt mit

  $$(\tau_1, \alpha_1, \beta_1)(\tau_3, \alpha_2, \beta_2) \dots (\tau_{2\varrho-1}, \alpha_\varrho, \star)$$

  und K beginnt mit

  $$(\tau_0, \varepsilon, \alpha_1)(\tau_2, \beta_1, \alpha_2) \dots (\tau_{2\varrho-4}, \beta_{\varrho-2}, \alpha_{\varrho-1})(\tau_{2\varrho-2}, \beta_{\varrho-1}, \alpha_\varrho)$$

DEFINITION: $\text{NoMatching}^E(k)$ ist das Ereignis, daß es i, j, s gibt, so daß $\pi_{i,j}^s$ akzeptiert, es aber kein $\pi_{j,i}^t$ gibt, das eine matching conversation mit $\Pi_{i,j}^s$ geführt hat.

Nun definiere einen Sicherheitsbegriff:

Sichere gegenseitige Authentifizierung:

1. (MC $\Rightarrow$ acc) Wenn $\pi_{i,j}^s$ und $\pi_{j,i}^t$ eine matching conversation geführt haben, dann akzeptieren beide.
2. (acc $\Rightarrow$ MC) Die Wahrscheinlichkiet von $\text{NoMatching}^E(k)$ ist in $\mathcal{O}(k^{-c})$ für alle $c > 0$.

BEISPIEL: Sei a ein gemeinsamer Schlüssel von A und B. Betrachte folgenden Austausch:

$$\begin{aligned} A \to B\colon &\quad x := \{A, 10\} \\ B \to A\colon &\quad y := \{B, 10, 20\}_a \\ A \to B\colon &\quad z := \{A, 20\}_a \end{aligned}$$

Dann akzeptieren $\pi_{A,B}^1$, $\pi_{A,B}^2$ und $\pi_{B,A}^1$. Damit aber eine matching conversation vorliegt, müßte im gestrichelten Bereich eine weitere Instanz $\pi_{B,A}^2$ vorhanden sein, die ein x erhält und die mit einem y antwortet.

DEFINITION: Sei $\text{MultipleMatch}^E(k)$ das Ereignis, daß ein $\pi_{i,j}^s$ akzeptiert und es $\pi_{j,i}^t$ und $\pi_{j,i}^{t'}$ gibt mit $t \neq t'$, die eine matching conversation mit $\pi_{i,j}^s$ geführt haben.

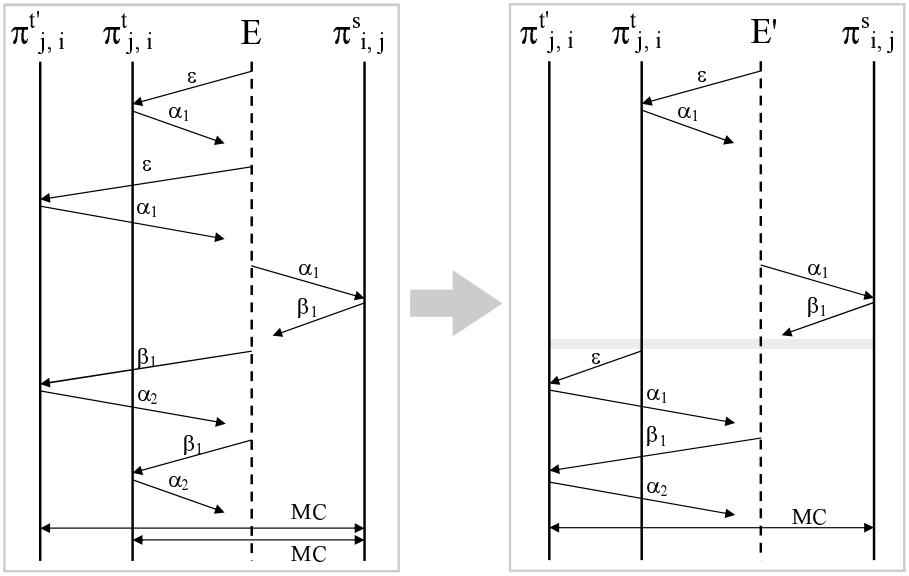
SATZ: Ist ein Authentifizierungsprotokoll sicher, so gilt

$$P(\text{MultipleMatch}^E(k)) \in \mathcal{O}(k^{-c}) \; \forall \; c > 0$$

BEWEIS: Wir trennen $\text{MultipleMatch}^E(k)$ in das Ereignis, daß ein Antworter und zwei Initiatoren vorliegen ($\text{MM}_1^E(k)$) und daß ein Initiator und zwei Antworter vorliegen ($\text{MM}_2^E(k)$). Dann ist

$$P(\text{MultipleMatch}^E(k)) \leq P(\text{MM}_1^E(k)) + P(\text{MM}_2^E(k))$$

Erster Fall: Angenommen, $P(\text{MM}_1^E(k))$ sei nicht vernachlässigbar klein. Dann muß folgende Kommunikation vorliegen (links):



Wir konstruieren aus E (links) einen Angreifer E' (rechts), der einen Verstoß gegen „acc ⇒ MC“ aufdeckt. Idee: Liegt ein MultipleMatch vor, so verschiebe die Kommunikation mit einem der beiden Protokolläufe einfach nach hinten und repliziere die vorher aufgezeichneten Nachrichten.

Problem hierbei: i und j sind dem Angreifer unbekannt, deshalb rät er i und j (Faktor $|I|^2$). Die entsprechenden Protokolläufe kann er dann durchprobieren. Dann ist

$$P(\text{NoMatch}) \geq P(\text{MultipleMatch}) \cdot |I|^{-2}$$

Wir betrachten nun das Protokoll $\mathsf{MAP1}^a$, von dem wir zeigen, daß es sicher ist: ■

$$\begin{aligned} A \to B\colon &\quad R_A \\ B \to A\colon &\quad [B\, A\, R_A\, R_B]_a \\ A \to B\colon &\quad [A\, R_B]_a \end{aligned}$$

Dabei sind R_A, R_B zufällig mit Länge k und $A, B \in \{0,1\}^k$. Weiter ist $[x]_a := x f_a(x)$ mit einer Funktion $f_a\colon \{0,1\}^{\leq 4k} \to \{0,1\}^k$, wobei $\{f_a\}_{a\in\{0,1\}^k}$ eine Familie von Funktionen ist, die ein Unterscheider „nicht“ von der Familie aller Funktionen $\{0,1\}^{4k} \to \{0,1\}^k$ unterscheiden kann.

SATZ: Das Protokoll MAP1 bietet sichere Authentifizierung.

BEWEISSKIZZE: Grundsätzliche Beweisstrategie: Wir nehmen an, es gibt einen erfolgreichen Angreifer E auf MAP1 und zeigen, daß es dann einen erfolgreichen Unterscheider U für $\{f_a\}$ gibt. Erster Schritt: Wir untersuchen, wie sich MAP1 verhält, wenn wir es in einer idealen *random world* laufen

lassen, d.h. wenn wir annehmen, daß anstelle der Funktion f_a eine zufällige Funktion $g\colon \{0,1\}^{4k} \to \{0,1\}^k$ gewählt werden. ■

NOTATION: $\mathsf{MAP1}^g$ sei

$$\begin{array}{ll} A \to B\colon & R_A \\ B \to A\colon & [B\,A\,R_A\,R_B]_g \\ A \to B\colon & [A\,R_B]_g \end{array}$$

wobei $[x]_g = xg(x)$. Sei im folgenden E ein Angreifer und sei $T_E(k)$ eine polynomielle obere Schranke für die Anfragen, die E an die einzelnen Protokolle stellt.

LEMMA: Die Wahrscheinlichkeit, daß E erfolgreich ist bei $\mathsf{MAP1}^g$ – der Zufallsvariante von $\mathsf{MAP1}$ – ist $\leq T_E(k)^2 \cdot 2^{-k}$.

Hier heißt *erfolgreich*, daß akzeptiert wird, ohne daß eine matching conversation vorliegt. Mit anderen Worten: Die Zufallsvariante ist sicher.

BEWEIS:

1. Seien A, B, s fest. Die Wahrscheinlichkeit, daß $\pi^s_{A,B}$ akzeptiert ohne MC, vorausgesetzt $\pi^s_{A,B}$ ist ein Initiator, ist $\leq T_E(k) \cdot 2^{-k}$. Wir unterscheiden zwei Fälle:

   (a) $[B\,A\,R_A\,R_B]_g$ wurde von keinem Prozess vor/bei τ_0 gesendet.

   (b) $[B\,A\,R_A\,R_B]_g$ wurde vor/bei τ_0 gesendet.

   Nun in den einzelnen Fällen:

   (a) Wir zeigen: In diesem Fall hat E nur mit Wahrscheinlichkeit 2^{-k} Erfolg. Zu einem erfolgreichen Lauf betrachten wir einen Lauf zu g' mit

   $$g(x) = \begin{cases} g'(x) & \text{falls} \quad x \neq B\,A\,R_A\,R_B \\ y \neq g'(x) & \text{falls} \quad x \neq B\,A\,R_A\,R_B \end{cases}$$

   Da sich bis zum Zeitpunkt τ_2 die Nachricht $[B\,A\,R_A\,R_B]_g$ nicht verschickt wurde, wird bis zum Zeitpunkt τ_2 alles gleich ablaufen. Insbesondere wird $\Pi^s_{A,B}$ die Nachricht $[B\,A\,R_A\,R_B]_g$ erhalten, aber eine Nachricht der Form $[B\,A\,R_A\,R_B]_{g'}$ erwartet und nicht akzeptieren!

   (b) Wenn $[B\,A\,R_A\,R_B]_g$ gesendet wurde, muß dies durch ein $\pi^t_{B,A}$ geschehen sein, das R_A empfangen hat. Da R_A zum Zeitpunkt τ_0 zufällig gewählt wird, ist die Wahrscheinlichkeit, daß es vorher schon einmal auftrat, sogar $\leq (T_E(k) - 1) \cdot 2^{-k}$.

2. Analoge Situation für Antworter: Etwas schwieriger, da die Nachricht $[A\, R_B]_g$ auch von einem $\pi^u_{A,C}$ kommen kann.

Zum Schluß zum Unterscheider: Dieser simuliert einfach E und die $\pi^s_{i,j}$ mit der gegebenen Funktion g bzw. f_a und sagt *real*, wenn E erfolgreich ist. Das Lemma garantiert dann, daß der Unterscheider gut ist. ■

7.4 Authentifizierter Schlüsselaustausch

Ziel ist die Erweiterung von MAP1 zu AKEP1 und AKEP2. Zusätzlich zu bisherigem:

- Wir haben eine Familie $\{S_k\}_{k\in\mathbb{N}}$ von Schlüsselverteilungen mit $S_k \subseteq \{0,1\}^{\sigma(k)}$.

  Die **private Ausgabe** α eines Teilnehmers soll der Sitzungsschlüssel sein (genauer: wenn akzeptiert wird, ist $\alpha \in S_k$, sonst ist $\alpha = \star$).

- Der Angreifer darf sich Schlüssel „besorgen“, d.h. er darf Nachrichten der Form (i, j, s, reveal) schicken. Dann wird $\alpha^s_{i,j}$ an E zurückgegeben.

- Zum Schluß befragt der Angreifer E ein frisches (s.u.) $\pi^s_{i,j}$ durch Senden der Anfrage (i, j, s, test). Er erhält mit je gleicher Wahrscheinlichkeit entweder $\alpha^s_{i,j}$ oder einen zufälligen Schlüssel und muß dann *real* oder *random* antworten. Das Ereignis, daß er richtig rät, sei $\text{GoodGuess}^E(k)$. Der Vorteil des Angreifers ist dann

$$\text{adv}^E(k) = \max\{0, P(\text{GoodGuess}^E(k)) - \frac{1}{2}\}$$

DEFINITIONEN:

- $\pi^s_{i,j}$ ist zu Beginn *ungeöffnet*. $\pi^s_{i,j}$ bleibt ungeöffnet, bis E eine Anfrage (i, j, s, reveal) stellt, dann ist $\pi^s_{i,j}$ *geöffnet*.
- $\pi^s_{i,j}$ heißt *frisch*, falls
  - $\pi^s_{i,j}$ akzeptiert hat,
  - $\pi^s_{i,j}$ ist ungeöffnet, und
  - $\pi^s_{i,j}$ hat keine matching conversation mit einem geöffneten $\pi^s_{j,i}$.

Ein „harmloser“ Angreifer:

Ein *transparenter Angreifer* verändert keine Nachrichten, sondern liefert diese korrekt aus. Er darf dann aber Protokolläufe anstoßen und auch test-Nachrichten versenden.

Zum Sicherheitsbegriff:

DEFINITION: Ein *sicheres authentifiziertes Schlüsselaustauschprotkoll*

- beinhaltet ein sicheres Authentifizierungsprotokoll;
- für den transparenten Angreifer gilt: für vom ihm gewählte $\pi_{i,j}^s$ und $\pi_{j,i}^t$ soll gelten,
  1. daß beide akzeptieren,
  2. daß $\alpha_{i,j}^s = \alpha_{j,i}^t$, und
  3. daß $\alpha_{i,j}^s$ (als Zufallsvariable betrachtet) wie S_k verteilt ist;
- für jeden beliebigen Angreifer E ist $\left|\mathrm{adv}^E(k)\right| \in \mathcal{O}(n^{-c})$ für alle $c > 0$.

BEISPIELE:

1. AKEP1: Vorausgesetzt wird, daß A und B zwei langlebige Schlüssel a_1 und a_2 besitzen. a_1 wird für das Signieren verwendet, a_2 für das symmetrische Verschlüsseln: $\{a\}_{a_2} = (r, f'_{a_2}(r) \oplus \alpha)$. Dabei ist

   $$f_{a_1} \colon \{0,1\}^{\leq 5k+\sigma(k)} \to \{0,1\}^k \quad \text{und} \quad f'_{a_2} \colon \{0,1\}^k \to \{0,1\}^{\sigma(k)}$$

   Das Protokoll ist nun wie folgt aufgebaut:

   $$\begin{array}{ll} A \to B\colon & R_A \\ B \to A\colon & [B\, A\, R_A\, R_B\, \{a\}_{a_2}]_{a_1} \\ A \to B\colon & [A\, R_B]_{a_1} \end{array}$$

   Es gilt der folgende Satz:

   SATZ: AKEP1 ist sicher im obigen Sinn, sofern $\{f_{a_1}\}$ und $\{f'_{a_2}\}$ sicher bzw. pseudozufällig sind.

2. AKEP2

   $$\begin{array}{ll} A \to B\colon & R_A \\ B \to A\colon & [B\, A\, R_A\, R_B]_{a_1} \\ A \to B\colon & [A\, R_B]_{a_1} \end{array}$$

Der ausgetauschte Schlüssel ist $\alpha = g(f'_{a_2}(R_B))$ mit einer deterministischen, polynomiellen Funktion g, die dafür sorgt, daß die Verteilung von Bild f anpaßt an die geforderte Verteilung von S_k.

BEMERKUNGEN: Es gibt praktikable und als sicher bewiesene Protokolle – das gleiche gilt auch für den asymmetrischen Fall! Es gibt zudem eine ganze Reihe von Protokollen, die für spezielle Anwendungen entwickelt werden.

8 Sichere Kanäle

Ultimatives Ziel ist, daß Alice und Bob miteinander sprechen (Nachrichtenaustausch) und zwar so, daß

- Vertraulichkeit gewahrt ist,
- Nachrichtenintegrität garantiert wird und
- Authentizität (von Nachrichten und Personen) gewährt wird.

In der Praxis benutzte Protokolle (z.B. ssh) sind sehr komplex – alle Techniken, die wir in der Vorlesung entwickelt haben, werden benutzt!

Ein geeigneter Sicherheitsbegriff wäre entsprechend auch sehr komplex[7], aber es gibt Untersuchungen zu Teilaspekten.

Beispiel: ssh bietet sicheres Einloggen mit drei wichtigen Komponenten:

- transport layer protocol
  - + server authentication
  - + Vertraulichkeit
  - + Integrität
- user authentication
- connection protocol: unterschiedliche logische Kanäle werden durch einen tatsächlichen Kanal realisiert (multiplexing)

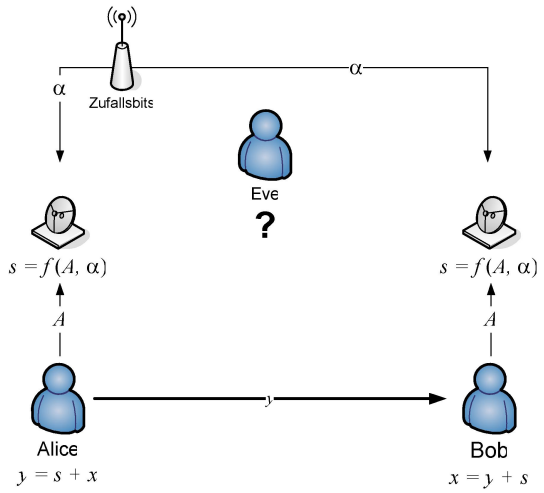
[7] Es gibt nur eine Arbeit, die das in Angriff nimmt!

9 Ausblick & Verschiedenes

9.1 Hyper Encryption – Everlasting Security – Bounded Storage Model

Grundsätzliche Funktionsweise (Maurer, Michael Rabin, Yan Zong Ding): symmetrisches Verfahren, bei dem Alice und Bob sich einen langlebigen geheimen Schlüssel teilen. Die Sitzungsschlüssel (one-time-pad) werden von Alice und Bob aus dem langlebigen Schlüssel und einer öffentlichen Quelle von Zufallsbits erzeugt; die eigentliche Verschlüsselung erfolgt dann wie beim one-time-pad.

Sicherheitsgarantie: Wenn Eve zum Zeitpunkt des Nachrichtenaustauschs nur wenig Speicherplatz zur Verfügung steht, kann sie später nur mit geringer Wahrscheinlichkeit eine Aussage zum Klartext machen, selbst dann, wenn sie den langlebigen Schlüssel bekommt.



Formalisierung: Sei B die Anzahl Bits an Speicherplatz, die zum Zeitpunkt der Verschlüsselung zur Verfügung stehen. Seien $\alpha_0, \alpha_1, \ldots$ Zufallsstrings der Länge n, die zum Verschlüsseln der Nachrichten $x_0, x_1, \ldots$ der Länge m benutzt werden. Dabei ist $\frac{B}{n} =: \eta < 1$. Der Sicherheitsparameter ist k; der Schlüssel A ist eine $(k \times m)$-Matrix mit Einträgen aus $\{0, \ldots, n-1\}$, d.h. Bitstrings der Länge $\log n$. Die Matrix enthält pro Klartextbit eine Spalte und entsprechend k Zeilen. Der Schlüssel s_l für das one-time-pad zur Verschlüsselung von x_l ergibt sich dann bitweise als

$$s_l(j) = \alpha_l(A_{0,j}) \oplus \alpha_l(A_{1,j}) \oplus \alpha_l(A_{2,j}) \oplus \ldots \oplus \alpha_l(A_{k-1,j})$$

Die eigentliche Verschlüsselung ist dann

$$y_l = x_l \oplus s_l$$

Sicherheitsaussage (adaptiver Angreifer): Es werden zu Beginn zwei Klartexte $x_0 \neq x_1$ gewählt. Der Angreifer funktioniert über l Runden und speichert sich in jeder Runde Informationen:

$$\begin{aligned} b_0 &= g_0(\alpha_0) \\ b_1 &= g_1(\alpha_1, b_0, s_0) \\ b_2 &= g_2(\alpha_2, b_1, s_0, s_1) \\ \dots &\quad \dots \\ b_{l-1} &= g_{l-1}(b_{l-2}, \alpha_{l-1}, s_0, s_1, \dots, s_{l-2}) \\ \beta &= h(b_{l-1}, A, x_0, x_1, y_p) \end{aligned}$$

Am Ende erhält der Angreifer also eine Nachricht y_p mit $p \in \{0, 1\}$ und muß sagen, ob diese $x_0 \oplus s_{l-1}$ oder $x_1 \oplus s_{l-1}$ war.

Für alle $m, k < 0.001 \cdot n$, für Nachrichten $x_0, x_1 \in \{0,1\}^m$, für alle g_i, h mit $\frac{B}{n} = \eta = \frac{1}{6}$ sowie $|g_i(\dots)| \leq B$ definiere:

$$\text{adv} = |P(h(b_{l-1}, A, x_0, x_1, y_0) = 0) - P(h(b_{l-1}, A, x_0, x_1, y_1) = 0)|$$

Dann ist

$$\text{adv} < m \cdot (3 \cdot l \cdot 2^{-\frac{k}{3}} + l \cdot 2^{-0.01 \cdot n})$$

Konkrete Zahlenwerte: Wir wählen:

n	$=$	2^{50} Bit	$=$	128 Terabyte
m	$=$	64 Bit	$=$	8 Byte
k	$=$			200
Übertragungsrate für α	$=$			50 Gigabyte $\cdot\, s^{-1}$
Übertragungszeit	$=$	$\frac{128\text{TB}}{50\text{GB}}\, s$	$=$	44 min
Schlüsselgröße	$=$	$64 \cdot 200 \cdot 50$ Bit	$\approx$	80 Kilobyte

Ausweg aus dem Problem der Schlüsselgröße: Aus dem initalen Schlüssel wird durch Schlüsselverstärkung der eigentliche Schlüssel hergestellt; dann dauert jedoch auch die Erzeugung viel länger.

9.2 Zero-Knowledge-Authentifizierungsprotokolle (Identification Schemes)

Ziel: Alice möchte Bob von ihrer Identität überzeugen!

Ansatz: Alice veröffentlicht ein schwieriges Rätsel und behauptet, die Lösung zu kennen. Wenn Alice Bob dann überzeugt, daß sie die Lösung kennt, ist Bob davon überzeugt, daß es sich um Alice handelt – **Problem**: Bob sollte

möglichst nichts von der Lösung erfahren, denn sonst könnte er sich für Alice ausgeben.

BEISPIEL: Drei-Färbung einer Landkarte! Analyse:

- Wenn Alice r mal richtig geantwortet hat, bei falschen Karte mit n Ländern und m Grenzen:

$$\left(1-\frac{1}{m}\right)^r = \left(\left(1-\frac{1}{m}\right)^m\right)^{\frac{r}{m}} \leq \left(\frac{1}{\sqrt[m]{2}}\right)^r$$

  D.h., die Erfolgswahrscheinlichkeit für eine „falsche Alice“ fällt exponentiell mit der Anzahl der Runden! Konkreter Wert für r, der gut genug ist: $200 \cdot m$.

- Bob erfährt nichts: Was ein Beobachter von außen sieht, könnte man ohne Kenntnis der Karte leicht produzieren: Für das Länderpaar einer Runde bräuchte man nur zwei verschiedene Farben zufällig zu erzeugen

-

Praktische Umsetzung: zwei Probleme:

- Wie kommt Alice an eine Karte?
- Wie legt sich Alice fest auf eine Färbung? Antwort: Sie legt sich bitweise fest und benutzt ein *Bit-Commitment-Schema*!

9.3 Ein Bit-Commitment-Schema

1. Alice wählt
   (a) zwei große unterschiedliche Primzahlen p und q; sei $n = p \cdot q$
   (b) eine Zahl v, die kein quadratischer Rest modulo n ist und Jacobi-Symbol 1 hat[8]: $J_n(v) = 1$
2. Festlegung: Um sich auf b festzulegen, wählt Alice $r \in \mathbb{Z}_n^*$ zufällig und setzt $c = r^2 v^b$ und schickt n, v und c zu Bob. Falls also $b = 0$ ist, wird r^2 geschickt; falls $b = 1$ ist, so $r^2 v$.
3. Aufdecken: Alice schickt p, q, r, b und Bob überprüft alles.

[8]Das heißt: Da die Zahl Jacobi-Symbol 1 hat, muß $J_p(n) = J_q(n)$ gelten und $J_p(n) = \pm 1$ sein. Der Fall $J_p(n) = J_q(n) = 1$ fällt jedoch weg, da die Zahl dann quadratischer Rest modulo p und q und damit (chinesischer Restsatz) auch modulo n ist. Also ist die Zahl weder quadratischer Rest modulo p noch modulo q.

Idee dabei: Falls Bob ohne Hilfe von Alice aufdecken könnte, so könnte er auch quadratische Reste modulo n erkennen. Das ist aber (hoffentlich) schwierig.

9.4 Ein praktisches Zero-Knowledge-Protokoll

Die vereinfachte Version von FIAT-SHAMIR:

- Alice wählt als „Geheimnis“ (x, n) mit $n = pq$ und $x = y^2$ mit $y \in \mathbb{Z}_n^*$.
- Alice wählt $r \in \mathbb{Z}_n^*$ und schickt $a = r^2$ zu Bob.
- Challenge von Bob: $e \in \{0, 1\}$
- Alice berechnet $b = ry^e$, d.h. r, falls $e = 0$ und ry, falls $e = 1$.
- Bob akzeptiert, falls $b^2 = ax^e$ ist.

Das allgemeine FIAT-SHAMIR-Schema:

- Alice wählt als „Geheimnis“ $(x_0, x_1, \ldots, x_{t-1}, n)$ mit $n = pq$ und $x_0 = y_0^2$, $x_1 = y_1^2$ usw. mit $y_i \in \mathbb{Z}_n^*$.
- Alice wählt $r \in \mathbb{Z}_n^*$ und schickt $a = r^2$ zu Bob.
- Challenge von Bob: $(e_0, e_1, \ldots, e_{t-1}) \in \{0, 1\}^t$
- Alice berechnet $b = ry_0^{e_0} y_1^{e_1} \cdot \ldots \cdot y_{t-1}^{e_{t-1}}$
- Bob akzeptiert, falls $b^2 = ax_0^{e_0} \cdot \ldots \cdot x_{t-1}^{e_{t-1}}$ ist.

9.5 Elektronische Wahlen

Ein Beispiel-Verfahren – grundsätzliche Ideen:

- Die Abgabe der Stimmen erfolgt in Form von signierten, verschlüsselten Nachrichten.
- Das Auszählen des Ergebnisses geschieht durch mehrere vertrauenswürdige Institutionen.
- Nur das Gesamtergebnis wird entschlüsselt.

Vorüberlegung: Wie verteilt man ein „Geheimnis“ an n Personen, so daß jeweils $\geq t$ Personen gemeinsam das Ergebnis rekonstruieren können, $< t$ Personen aber nie?

Benutze ein **Secret Sharing Scheme:**: Das Geheimnis ist die Konstante eines Polynoms f vom Grad $(t-1)$ in einem Körper mit $> n$ Elementen. Jede der n Personen erhält als „Anteil“ am Geheimnis ein Paar $(x, f(x))$. Jede Wahl von t Paaren bestimmt das Polynom eindeutig.

Für jede Wahl $\{(x_0, y_0), \ldots, (x_k, y_k)\}$ mit $k < t-1$ und je zwei Konstanten c und c' gilt: Die Anzahl der Polynome $(t-1)$-ten Grades mit $f(x_i) = y_i$ und $f(0) = c$ ist genauso groß wie die Anzahl der Polynome mit $f(0) = c'$.

Ablauf des Verfahrens:

- **Voraussetzung**: $A_0, \ldots, A_{n-1}$ seien die vertrauenswürdigen Instanzen, $V_0, \ldots, V_{m-1}$ seien die Wähler.
- **Initialisierung**:
  - wähle eine große Primzahl p
  - wähle eine große Primzahl q mit $q \mid p-1$
  - wähle $g \in \mathbb{Z}_p^*$ mit Ordnung q
  - sei G die von g erzeugte Untergruppe von Z_p^*, also $G := \{g, g^2, g^3, \ldots\}$
  - geheimer Schlüssel ist $s \in \{1, \ldots, q-1\}$
  - öffentlicher Schlüssel ist $h := g^s \bmod p$
  - jede Instanz A_i erhält einen Teil (j, s_j) von s nach einem SecretSharingScheme (hier verwenden wir obige Polynom-Interpolation)
  - die Werte $h_j = g^{s_j}$ werden veröffentlicht

  Die Nachrichten sind dann Elemente von G. Die Verschlüsselung erfolgt nach dem ElGamal-Schema: $m \in G$ durch $(g^\alpha, h^\alpha m)$ mit α zufällig. Entschlüsselt wird eine Nachricht (c, c') dann durch $m = c' \cdot c^{-s}$.
- **Stimmabgabe**:
  - Jeder Wähler V_i entscheidet sich für $v_i \in \{-1, 1\}$.
  - Er wandelt dies in g^{v_i} um und verschlüsselt dies zu $(c_i, c_i') = (g^{\alpha_i}, h^{\alpha_i} g^{v_i})$.
  - Er signiert und veröffentlicht seine Stimme.
- **Auszählen**:

- Sei $(c, c') = (\prod_{i=0}^{m} c_i, \prod_{i=0}^{m} c'_i)$. Dies ist dann die Verschlüsselung von $g^{\sum v_i}$, also g^d mit d als Differenz zwischen *ja*- und *nein*-Stimmen.
- Sei J die Menge der (mitspielenden) vertrauenswürdigen Instanzen.
- Jedes A_j veröffentlicht $w_j = c^{s_j}$; daraus kann man c^s berechnen:

$$c^s = c^{\sum_{j \in J} \lambda_{j,J} s_j} = \prod_{j \in J} w_j^{\lambda_{j,J}} \text{ mit } \lambda_{j,J} = \prod_{l \in J \setminus \{j\}} \frac{l}{l - j}$$

- Man erhält dann c^s und somit kann man g^d entschlüsseln. Durch Ausprobieren (relativ kleine Anzahl Wähler bzw. einigermaßen bekanntes Ergebnis) kann man dann d bestimmen!

- **Überprüfung** problematischer Stellen:
  - V_i darf nur $v_i \in \{1, -1\}$ wählen!
  - A_j sollte wirklich c^{s_j} veröffentlicht haben!

  Für beide Probleme werden gesonderte Subprotokolle benutzt.

A Häufigkeiten von Buchstaben etc.

e	17,40%	d	5,08%	o	2,51%	v	0,67%
n	9,78%	h	4,76%	b	1,89%	j	0,27%
i	7,55%	u	4,35%	w	1,89%	y	0,04%
s	7,27%	l	3,44%	f	1,66%	x	0,03%
r	7,00%	c	3,06%	k	1,21%	q	0,02%
a	6,51%	g	3,01%	z	1,12%		
t	6,15%	m	2,53%	p	0,79%		

ein	1,22%	der	0,86%
ich	1,11%	che	0,75%
nde	0,89%	end	0,75%
die	0,87%	gen	0,71%
und	0,87%	sch	0,66%

Koinzidenzindex für deutsche Sprache: $\kappa = 0.0762$

B Visualisierungen von Kryptosystemen

B.1 Beispielsystem für die lineare Kryptanalyse