Asymmetrische Konzelationssysteme

Inhaltsverzeichnis

1 Einleitung

Ziel dieses Versuches ist, Funktionsweise und charakteristische Eigenschaften asymmetrischer Konzelationssysteme kennenzulernen. Dazu gehören insbesondere ihre Anwendungsgebiete sowie ihre Stärken und Schwächen in punkto Sicherheit.

Zur Motivation: Symmetrische Konzelationssysteme erfordern für jede potentielle Kommunikationsbeziehung in einem Netz einen privaten Schlüssel beider Partner. Die Anzahl t dieser maximal erforderlichen Schlüssel wächst also quadratisch mit der Anzahl k der Netzknoten, denn es gilt:

$$t = \frac{k(k-1)}{2}$$

Die große Menge zu verwaltender Schlüssel reduziert sich drastisch, wenn der Dechiffrier- aus dem Chiffrierschlüssel nicht berechnet werden kann, so daß jeder Netzknoten seinen Chiffrierschlüssel veröffentlichen kann. Obwohl jeder Empfänger nur einen Schlüssel veröffentlicht hat, können ihm anders als mit einem symmetrischen Konzelationssystem, beliebig viele Sender vertraulich Nachrichten zusenden. Dadurch würde pro Netzknoten nur noch ein Schlüssel gebraucht und könnte in einem öffentlichen Register gespeichert werden, so daß gilt:

$$t = k$$

In größeren Netzen ist dies eine gigantische Einsparung. Derartige asymmetrische Kryptosysteme werden nun untersucht.

2 Asymmetrische Konzelationssysteme

Es sei zunächst erläutert, was ein Konzelationssystem ist:

Ein **Konzelationssystem** besteht aus

- einer Menge $\mathcal{M}$ von Klartextnachrichten (messages),
- einer Menge $\mathcal{C}$ von Schlüsseltextnachrichten (ciphertexts),
- einer Menge $\mathcal{R}$ von Zufallszahlen (random), sowie zwei effizient berechenbaren Funktionen
$$k_c : \mathcal{R} \to \mathcal{K}_c \quad und \quad k_d : \mathcal{R} \to \mathcal{K}_d,$$
die auf $r \in \mathcal{R}$ angewandt jeweils einen Chiffrier- bzw. Dechiffrierschlüssel (key) ausgeben. (Dieser kann aus mehreren Parametern bestehen.)
- einer effizient berechenbaren Chiffrierfunktion $c : \mathcal{K}_c \times \mathcal{M} \to \mathcal{C}$ und
- einer effizient berechenbaren Dechiffrierfunktion $d : \mathcal{K}_d \times \mathcal{C} \to \mathcal{M}$.

Die Sicherheit des Systems beruht ausschließlich auf der Geheimhaltung der gewählten Zufallszahl r . Einem Angreifer ohne Kenntnis von r soll es in vertretbaren Zeiträumen nicht möglich sein, $k_d(r)$ zu berechnen.
Für jeden Schlüssel $r \in \mathcal{R}$ und jeden Klartext $m \in \mathcal{M}$ gilt dabei:

$$d\left(k_d(r), c\left(k_c(r), m\right)\right) = m$$

Seit dem grundlegenden Artikel von Diffie und Hellman [DiHe_76] und dem von Simmons [Simm_79] unterscheidet man zwei Arten von Konzelationssystemen, die symmetrischen und die asymmetrischen. Ein Konzelationssystem heißt **symmetrisch**, falls Chiffrierschlüssel $k_c(r)$ und Dechiffrierschlüssel $k_d(r)$ aus dem jeweils anderen effizient berechenbar sind[1], wenn die Zufallszahl r unbekannt ist. Es heißt **asymmetrisch**, falls kein polynomialer Algorithmus öffentlich bekannt ist, der den Dechiffrier- aus dem Chiffrierschlüssel berechnet[2]. (Der umgekehrte Fall ist für Konzelationssysteme uninteressant, weil der Chiffrierschlüssel öffentlich bekannt ist.) Damit gründet die Unterscheidung von symmetrischen und asymmetrischen Konzelationssystemen in der Komplexitätstheorie.

Bild 1 zeigt die Zusammenhänge beim asymmetrischen Kryptosystem: Jedes Schlüsselpaar (Chiffrier- und Dechiffrierschlüssel) eines asymmetrischen Konzelationssystems teilt die Welt in zwei (allerdings asymmetrische) Klassen: in den einen, der den Dechiffrierschlüssel kennt, und die anderen, die ihn nicht kennen. Er ($k_d(r)$) wird daher **geheimer Schlüssel** genannt. Den Kontakt zwischen beiden Klassen stellt der eine her, der die Zufallszahl r und

[1] Eine Formalisierung von „effizient“ lautet z.B., daß beide Berechnungen von polynomialer Komplexität sind.
[2] Streng genommen heißt es asymmetrisch, falls kein polynomialer Algorithmus bekannt ist, der den Dechiffrierschlüssel mit signifikanter Wahrscheinlichkeit aus dem Chiffrierschlüssel (evtl. probabilistisch) berechnet.

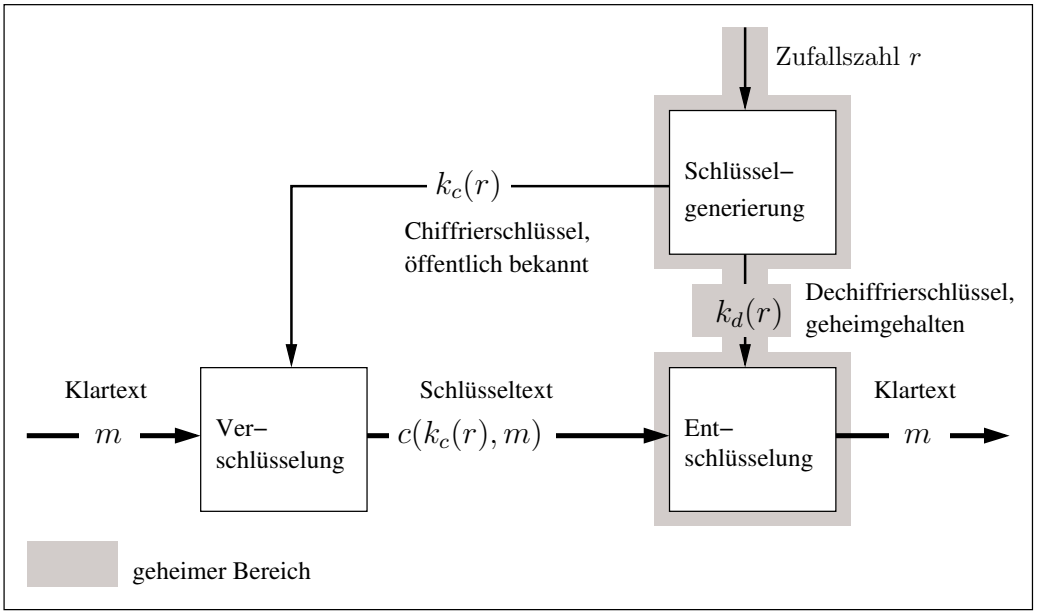

Abbildung 1: Asymmetrisches Konzelationssystem

damit sowohl den Dechiffrier- als auch den Chiffrierschlüssel kennt, indem er diesen Chiffrierschlüssel der anderen Klasse mitteilt d.h. unter seinem Namen veröffentlicht. Dies ($k_c(r)$) ist der **öffentliche Schlüssel**.

Die Sicherheit asymmetrischer Konzelationssysteme ruht allein darauf, daß es „den anderen“ in vernünftigen Zeiträumen unmöglich ist, den geheimen Schlüssel „des einen“ systematisch zu errechnen, selbst wenn sie den zugehörigen öffentlichen Schlüssel kennen.

Für die Chiffrierfunktion eines asymmetrischen Konzelationssystems eignen sich daher nur solche (invertierbaren) Funktionen, die durch Kenntnis des öffentlichen Schlüssels effizient berechenbar sind, deren Inverse aber nur dann effizient berechenbar ist, wenn der geheime Schlüssel bekannt ist. Sie müssen invertierbar sein, weil nur eine Injektion der Klartext- auf die Schlüsseltextmenge eine eindeutige Entschlüsselung aller Schlüsseltexte erlaubt.

Die Informationstheorie lehrt, daß im informationstheoretischen Sinne keine unumkehrbare Injektion auf einer abzählbaren Menge existiert: Eine mögliche Berechnung der Inversen einer Injektion ist stets die erschöpfende Suche in der Urbildmenge, mit anschließendem Auswerten der Funktion und Vergleich mit dem gegebenen Wert (z.B. Schlüsseltext). Da Klartexte Zeichenketten über endlichen Alphabeten sind, ist die Klartextmenge stets abzählbar.

Es wird vermutet, daß die Komplexitätstheorie ein geeignetes Modell für Funktionen der oben geschilderten Art bietet. Solche Funktionen werden im folgenden Einwegfunktionen[3] mit Geheimnis (*trapdoor-oneway-function*) genannt.

[3]Das bedeutet **nicht**, daß die Funktion nach einmaligem Gebrauch weggeworfen wird.

Einwegfunktion mit Geheimnis (*trapdoor-oneway-function*)

Eine Funktion $f : A \to B$ heißt **Einwegfunktion mit Geheimnis** genau dann, wenn sie und ihre Inverse f^{-1} effizient berechnet werden können. Dabei muß es möglich sein, einen effizienten Algorithmus zur Berechnung von f anzugeben, ohne daß daraus leicht ein effizienter Algorithmus für die Berechnung von f^{-1} gewonnen werden könnte. Das Geheimnis, das einem ermöglicht, beide effizienten Algorithmen zu finden, ist das **Geheimnis von** f.

Ein Kandidat für eine Einwegfunktion mit Geheimnis ist z.B.[4]

1. modulare Potenzierung:

   $$\mathbf{potmod}_{m,n} : \mathbb{Z}_n \to \mathbb{Z}_n \qquad n \in \mathbb{N} \text{ und } n \text{ nicht prim}, m \in \mathbb{Z}^*_{\phi(n)}$$
   $$\mathbf{potmod}_{m,n}(a) := a^m \pmod n$$

   Inverse:

   $$\mathbf{rootmod}_{m,n} : \mathbb{Z}_n \to \mathbb{Z}_n$$
   $$\mathbf{rootmod}_{m,n}(a) := a^{m^{-1}(\bmod \phi(n))} \pmod n \qquad \text{(s. Versuch ZA)}$$

   Geheimnis: Primfaktorzerlegung von n

   Hier ist anzumerken, daß das Geheimnis (Primfaktorzerlegung von n) hinreichend für die effiziente Berechnung von **rootmod** ist. Es ist aber bisher nicht bewiesen, daß sie auch notwendig ist. Das bedeutet, die Berechnung von **rootmod** ist nicht als äquivalent zur Faktorisierung des verwendeten Modulus n bewiesen.

2. modulares Quadrieren:

   $$\mathbf{sqrmod}_n : \mathbf{QR}_n \to \mathbf{QR}_n \qquad n \in \text{BLUM} \qquad \text{(s. Versuch ZA.8)}$$
   $$\mathbf{sqrmod}_n(a) := a^2 \pmod n$$

   Inverse:

   $$\mathbf{sqrtmod}_n : \mathbf{QR}_n \to \mathbf{QR}_n \qquad n \in \text{BLUM}$$
   $$\mathbf{sqrtmod}_n(a) := \mathbf{SQRTMOD}^*(a, p, q) \qquad \text{(s. Versuch ZA.8.1)}$$

   Geheimnis: Primfaktorzerlegung (p, q) von n, mit $p \cdot q = n$

   In diesem Falle ist bewiesen, daß das Geheimnis, die (beiden) Primfaktoren von n, notwendig und hinreichend für die Berechnung der Inversen **sqrtmod** sind [Rabi_78, S.155-166][Bras_88]. Die Inverse **sqrtmod** von **sqrmod** effizient zu berechnen, ist äquivalent zur Faktorisierung des Moduls n (s. Versuch ZA.6.4).

[4] Im folgenden werden Ergebnisse des Versuches „Zahlentheoretische Algorithmen“ benötigt. Im folgenden wird z.B. auf Versuch „Zahlentheoretische Algorithmen“, Kapitel 4 durch „Versuch ZA.4“ verwiesen.

Der Sicherheitsbeweis für jeden dieser Kandidaten ruht demnach auf einer komplexitätstheoretischen Annahme. Es gibt bisher keinen Beweis im strengen Sinne für die angenommene Komplexität der oben genannten Kandidaten, sondern lediglich eine Reduktion auf die Komplexität einer anderen Funktion, über deren Komplexität eine Annahme besteht, der weitgehend zugestimmt wird. In den beiden Beispielen ist dies jeweils die Faktorisierungsannahme (s. Versuch ZA).

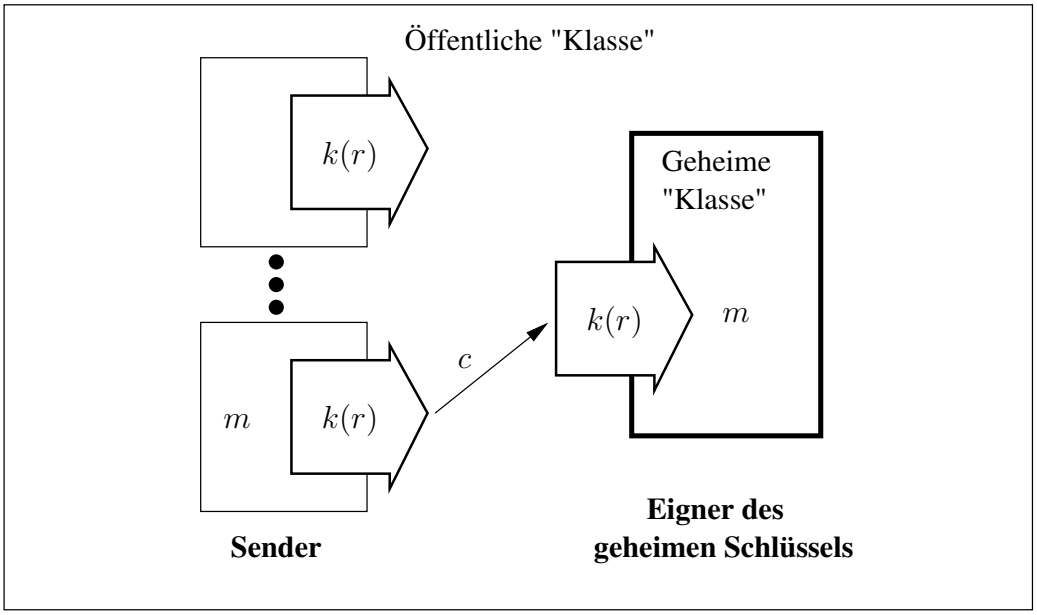

Abbildung 2: Die „Welt“ aus der Sicht eines asymmetrischen Konzelationssystems

Trifft sie zu, so trennt die beiden eingangs beschriebenen Klassen eine komplexitätstheoretische Grenze (angedeutet durch die fette Kapsel in Bild 2). Der eine (Inhaber des geheimen Schlüssels) hätte einen zeitlichen Informationsvorsprung, den die anderen (die den geheimen Schlüssel nicht kennen) in vernünftigen Zeiträumen beweisbar nicht einholen könnten (Bild 2).

3 Angriffe und Angreifermodelle

Man unterscheidet grundsätzlich zwei Arten von Angriffen auf Konzelationssysteme (und auf Signatursysteme), je nachdem, welche Art von Kontakt der Angreifer A zum Angegriffenen B aufnimmt.

Beschränkt sich der Angriff darauf, Informationen, Nachrichten, Schlüssel etc. zu empfangen bzw. abzuhören, so spricht man von einem **passiven Angriff**. Die Kommunikation ist allein von B (wenn auch ungewollt) zum Angreifer A gerichtet.

Sendet der Angreifer darüber hinaus auch selbst Nachrichten an B, um ihn zu stören, zu täuschen oder seine Reaktion auszuwerten, so spricht man von einem **aktiven Angriff**.

Der Angreifer nutzt sowohl die Möglichkeit der Kommunikation von A zu B als auch von B zu A. Dabei unterscheidet man noch, ob der Angreifer alle Nachrichten, die er an B schicken wird, wählt, bevor er die erste von ihnen abschickt, oder ob er sie in Abhängigkeit der empfangenen Antworten (interaktiv) wählt. Letztere bezeichnet man als **adaptive aktive Angriffe**. Aktive Angriffe sind potentiell komplexer und gefährlicher als passive Angriffe, adaptive Angriffe sind gefährlicher als nichtadaptive Angriffe.

Sowohl für passive Angriffe (basierend auf Turingmaschinen [GoMi_84]) als auch für aktive Angriffe [BDPR_98] auf asymmetrische Konzelationssysteme existiert ein formales **Angreifermodell**. In [CrSh1_98] wurde erstmals ein praktikables asymmetrisches Konzelationssystem vorgestellt, das auch gegen aktive Angriffe beweisbar sicher ist.

Eine grobe Klassifizierung von aktiven und passiven Angriffen auf Konzelationssysteme zeigt Bild 3.

	Bezeichnung des Angriffs	bekannte Information (außer $k_c(r)$ bei asymmetrischen Konzelationssystemen)	gesuchte Information
passive Angriffe	Schlüsseltext-Angriff (ciphertext-only-attack)	$c_1, \ldots, c_k$	$m_1, \ldots, m_k$
	Klartext-Schlüsseltext-Angriff (known-plaintext-attack)	$(m_1, c_1), \ldots,$ $(m_{k-1}, c_{k-1}), c_k$	m_k
aktive Angriffe	gewählter Schlüsseltext-Klartext-Angriff (chosen-ciphertext-attack)	$(m_1, \mathbf{c_1}), \ldots,$ $(m_{k-1}, \mathbf{c_{k-1}}), c_k$	m_k
	gewählter Klartext-Schlüsseltext-Angriff (chosen-plaintext-attack)	$(\mathbf{m_1}, c_1), \ldots,$ $(\mathbf{m_{k-1}}, c_{k-1}), c_k$	m_k

Abbildung 3: Angriffsarten auf symmetrische und asymmetrische Konzelationssysteme. (Dabei bezeichne (m_i, c_i) jeweils ein Klartext-Schlüsseltextpaar. Die vom Angreifer wählbaren Parameter sind durch Fettdruck hervorgehoben. Die grau hinterlegten Angriffe sind nur bei symmetrischen Konzelationssystemen sinnvoll.)

Bei asymmetrischen Systemen ist die Unterscheidung zwischen Schlüsseltext-Angriff und Schlüsseltext-Klartext-Angriff nicht sinnvoll, da sich der Angreifer **ohne** Interaktion mit dem Angegriffenen zu vom Angreifer gewählten Klartexten m_i die zugehörigen Schlüsseltexte c_i erzeugen kann. Es wäre also bei asymmetrischen Konzelationssystemen vollkommen unreali-

stisch anzunehmen, daß der Angreifer keine Klartext-Schlüsseltext-Paare hätte[5].

Im symmetrischen Fall kann man bei aktiven Angriffen unterscheiden, ob der Angreifer Klartexte wählt und Schlüsseltexte erhält oder umgekehrt oder gar beides. Im asymmetrischen Fall ist ein aktiver Angriff nur sinnvoll, wenn der Angreifer die Schlüsseltexte wählt und die Klartexte erhält. Das umgekehrte kann der mit Hilfe des öffentlichen Schlüssels selbst.

4 Ein deterministisches asymmetrisches Konzelationssystem: RSA

Als Beispiel für ein asymmetrisches Konzelationssystem sei das **RSA-Verfahren**, nach seinen Erfindern Rivest, Shamir, Adleman [RSA_78] benannt, vorgestellt.

- Aus der Zufallszahl r eines RSA-Konzelationssystems werden zunächst zwei (zufällige) Primzahlen p und q berechnet.
- Der öffentliche Schlüssel besteht aus zwei Komponenten $k_c(r) = (e, n)$ wobei $n = p \cdot q$, und e ein fest vereinbarter, öffentlicher Exponent ist.
- Der geheime Schlüssel besteht aus 5 Komponenten, um die Dechiffrierfunktion möglichst effizient berechnen zu können: $k_d(r) = (dp, dq, p, q, invpmodq)$ wobei gilt:

$$\begin{aligned} dp &= e^{-1} \pmod{\phi(p)} = e^{-1} \pmod{p-1} \\ dq &= e^{-1} \pmod{\phi(q)} = e^{-1} \pmod{q-1} \\ invpmodq &= p^{-1} \pmod{q} \end{aligned}$$

- Der Chiffrieralgorithmus ist

$$\mathbf{C}_{k_c(r)}(m) = \mathbf{C}_{e,n}(m) := \mathbf{potmod}_{e,n}(m) = m^e (\mathrm{mod}\, n) = c$$

- und der Dechiffrieralgorithmus[6] entsprechend

$$\mathbf{D}_{k_d(r)}(c) = \mathbf{D}_{dp,dq,p,q,invpmodq}(c) := \mathbf{rootmod}_{e,n}(c) = c^{e^{-1}} (\mathrm{mod}\, n) = c^d (\mathrm{mod}\, n) = m$$

  Der Dechiffrieralgorithmus kann effizienter berechnet werden, wie der folgenden Umformung zu entnehmen ist. Aufgrund der Kongruenzen

$$\begin{aligned} & m \equiv c^{e^{-1}} \pmod{n} && \text{(s. Versuch ZA.5)} \\ \Longrightarrow\quad & m \equiv c^{e^{-1}} \pmod{p} \quad \text{und} \quad m \equiv c^{e^{-1}} \pmod{q} \\ \Longrightarrow\quad & m \equiv c^{e^{-1}(\bmod p-1)} \pmod{p} \quad \text{und} \quad m \equiv c^{e^{-1}(\bmod q-1)} \pmod{q} \\ \Longleftrightarrow\quad & m \equiv c^{dp} \pmod{p} \quad \text{und} \quad m \equiv c^{dq} \pmod{q} \end{aligned}$$

[5] Es gibt noch einen kleinen, subtilen Unterschied: Im symmetrischen Fall wählt der Angegriffene die Klartexte, im asymmetrischen Fall der Angreifer. Letzteres ist dabei der stärkere Angriff.

[6] Bei Verwendung dieses weniger effizienten Algorithmus wird als geheimer Schlüssel nur eine Komponente $e^{-1} \pmod{\phi(n)} = d$ benötigt.

und des chinesischen Restsatzes

$$m = c^{dp} + p \cdot \Big((c^{dq} - c^{dp}) \cdot p^{-1} \pmod q \Big) \pmod n$$

ergibt sich schließlich für den Dechiffrieralgorithmus

$$\mathbf{D}_{dp,dq,p,q,invpmodq}(c) = c^{dp} + p \cdot \Big((c^{dq} - c^{dp}) \cdot invpmodq \pmod q \Big) \pmod n$$

Näheres zur effizienten Berechnung der RSA-Chiffrier- und Dechiffrierfunktionen findet man bei [Merr_83, Zim1_86].

Damit beruht RSA auf der Einwegfunktion **potmod** (1. Beispiel). Gleichzeitig ist damit weiterhin die Frage offen, ob die Dechiffrierfunktion, evtl. ohne p und q zu kennen, effizient berechnet werden kann. Denn mit **potmod** ist auch RSA bisher nicht als äquivalent zur Faktorisierung nachgewiesen.

Um einen Angriff durch Faktorisierung des Moduls n zu verhindern, verdient die Wahl der Primfaktoren p und q größere Aufmerksamkeit. Werden sie zu klein oder ungeeignet gewählt, so ist ihr Produkt leicht zu faktorisieren. Viele Autoren nennen Primzahlen p, q, deren Produkt $n = p \cdot q$ „schwer“ zu faktorisieren ist, **sichere Primzahlen** (safe primes). Welche Primzahlen sicher sind oder sein können, ist solange von den augenblicklich besten Faktorisierungsalgorithmen abhängig, wie keine grundlegenderen Ergebnisse über die Komplexität bestimmter Operationen wie z.B. Faktorisierung vorliegen. Das Produkt zweier Primzahlen muß derzeit mindestens 700 Bit lang sein, wobei dann die Primzahlen p und q etwa gleich lang sein sollten und sowohl $p-1$ als auch $q-1$ wenigstens einen großen Primfaktor enthalten sollten. Mit verbesserten Faktorisierungsalgorithmen wächst diese erforderliche Länge (s. Versuch ZA).

In der Literatur finden sich viele Hinweise darauf, wie die jeweils als sicher definierten Primzahlen erzeugt werden können. Es gibt prinzipiell zwei Methoden:

1. Direkte Konstruktion sicherer Primzahlen

2. Auswahl „sicherer Zahlen“ und anschließendes Aussieben der Primzahlen durch einen Primzahltest. Dabei sei eine sichere Zahl eine, die alle Eigenschaften einer sicheren Primzahl aufweist, aber evtl. nicht prim ist.

Insbesondere ist beim zweiten Verfahren zu unterscheiden, ob die Ergebnisse des verwendeten Primzahltests mit einem (wenn auch beliebig kleinen) Fehler behaftet sind, oder nicht. Sind sie fehlerbehaftet, so nennt man die erzeugten Zahlen **Pseudoprimzahlen** (vgl. Versuch ZA.7). Da alle heute bekannten effizienten Primzahltests probabilistisch sind, und zusammengesetze Zahlen nicht mit Sicherheit als solche erkennen, sind die mit der zweiten Methode gewonnenen Zahlen sämtlich Pseudoprimzahlen. Denn Zahlen in der Größenordnung von 350 Bit oder länger mit Sicherheit als Primzahl nachzuweisen, ist bisher nicht effizient durchführbar. Zu Erzeugung sicherer Primzahlen siehe [Knut_81, Denn_82, Maur_89, OSI1_85].

5 Passiver Angriff auf einen RSA-Schlüsseltext

Dieser Versuch soll am Beispiel RSA eine gravierende Schwäche aller deterministischen asymmetrischen Konzelationssysteme vermitteln. Grundsätzlich ist die Verschlüsselung $\mathbf{C}_{k_c(r)}(m)$ deterministischer Konzelationssysteme von genau zwei Parametern abhängig, dem öffentlichen Schlüssel $k_c(r)$ und dem Klartext m. Das hat zur Folge, daß derselbe Klartext stets auf denselben Schlüsseltext abgebildet wird. Da ein Angreifer anders als bei symmetrischen Konzelationssystemen den Chiffrierschlüssel kennt, ermöglicht der Determinismus prinzipiell einen Schlüsseltext-Angriff durch Raten oder systematisches Suchen des Klartextes. Der Angriff ist umso erfolgversprechender, je kürzer der zugehörige Klartext war. Besonders riskante Situationen entstehen, wenn der Angreifer die ungefähre Länge des Klartextes kennt, und seine Ausstattung mit Rechnerressourcen ausreicht, um den entsprechenden Klartextraum erschöpfend zu durchsuchen.

Wurde ein längerer Klartext im Codebuchverfahren (ECB, siehe [Pfit_98]) verschlüsselt, so liegt qualitativ dieselbe Situation vor, denn der Angriff braucht nur sukzessive auf jeden Schlüsseltextblock angewandt zu werden. Es gibt in der Praxis durchaus relevante Fälle, in denen ein Angreifer mit Recht davon ausgehen kann, den Schlüsseltext eines Klartextes z.B. der Länge 2 bis 4 Zeichen eines bekannten Alphabetes abgehört zu haben. Dies gilt z.B. für die meisten Einträge im Betragsfeld eines Eurocheques. Ähnlich ermöglichen viele Einträge in vorgedruckte Formulare einen derartigen passiven Angriff.

5.1 Angriffsszenario

Im folgenden Versuch ist der RSA-Schlüsseltext eines ausgefüllten Eurocheque-Formulares zu brechen. Gegeben ist

- der öffentliche Schlüssel des RSA-Systems (inkl. Kenntnis des Chiffrierverfahrens),
- der (unbemerkt) abgehörte Schlüsseltext,
- die Betriebsart ECB und
- die verwendete Blocklänge.

Die Aufgabe besteht darin, einen Teil des Schlüsseltext-Angriffs in möglichst effizienter Weise zu programmieren, wobei der abgehörte Schlüsseltext als vorbelegte Variable `targetPkt` im Programm zur Verfügung steht und die Kenntnis aller anderen Parameter bereits durch ein initialisiertes RSA-Konzelationssystem bereitgestellt wird. Dies ist sicher nicht mehr Information, als ein Angreifer realistischerweise bekommen kann. Ziel ist, den kompletten Klartext zu ermitteln.

Die Länge des verwendeten RSA-Moduls beträgt `RSA_CIPHER_LEN` [Bit], die Länge zu verschlüsselnder Codeblöcke: `RSA_PLAIN_LEN` [Bit]. (Um gegen heute verfügbare Rechnerleistungen Sicherheit zu gewähren, ist `RSA_CIPHER_LEN` $\geq$ 1024 Bit erforderlich. Im Versuch wird jedoch aus programmspezifischen Gründen mit einem kleineren Wert gearbeitet.)

Suchraumbeschränkungen

Um einen passiven Angriff auf den gegebenen Schlüsseltext zu ermöglichen, ist der Suchraum durch einige weitere Informationen (erheblich) einzuschränken:

1. Die Voreinstellung beträgt für diesen Versuch `RSA_CIPHER_LEN` = 256 Bit.
2. Der gegebene Schlüsseltext besteht aus insgesamt `blocks` Blöcken.
3. Die Eintragungen in den Formularfeldern des Eurocheques erfolgt in den Schlüsseltextblöcken $shl_1 \| \ldots \| shl_{\texttt{blocks}}$, wobei in den Blöcken folgende Einträge stehen:

Schlüsseltextblöcke $shl_1 \ldots shl_6$	Betrag
Schlüsseltextblöcke $shl_7 \ldots shl_{\texttt{blocks}}$	Kreditinstitut, BLZ, Kto.-nr., Ort, Datum, Name, Adresse

4. cod_i ist der in shl_i verschlüsselte Codeblock ($i \in \{1, \ldots, \texttt{blocks}\}$). Die Codeblöcke $cod_i \in \{0,1\}^{\texttt{RSA\_PLAIN\_LEN}}$ haben folgende Gestalt: $cod_i = (0, - - -, 0, c_i)$, wobei c_i die letzten (niederwertigsten) 8 Bit des i-ten Codeblocks belegt. Von `RSA_PLAIN_LEN` Bytes des Codewortes cod_i trägt also nur das letzte Byte Information über den Klartext (die Nullen zu Beginn jedes Blockes sind gezielt eingefügte Redundanz, um die erschöpfende Suche im Klartextraum zu erleichtern).

Ablauf eines möglichen Angriffs

Mit den oben genannten Vorgaben könnte ein passiver Angriff etwa wie folgt ablaufen. (Dabei werden bereits die Methoden genannt, die für diesen Versuch bereitgestellt werden. Ihre Spezifikation findet sich in dem folgenden Unterkapitel.)

(1) Rufe `serv.GetCipher()` auf, um `targetPkt`, den RSA-Schlüsseltext $shl_1 \| \ldots \| shl_{\texttt{blocks}}$ eines ausgefüllten Eurochequeformulares zu erhalten.

(2) Bestimme die Anzahl `blocks` von Schlüsseltextblöcken.

(3) Führe auf jeden dieser Blöcke shl_i einen passiven Angriff durch erschöpfende Suche aus:

(3.1) Gib unter Berücksichtigung der Suchraumbeschränkungen einen möglichen Codeblock cod_i vor, verschlüssele diesen und vergleiche ihn mit dem erhaltenen Schlüsseltextblock shl_i. Sind sie gleich, so fahre mit dem nächsten Block shl_{i+1} bei (3) fort, anderenfalls wiederhole (3.1).

(4) Sobald alle Codeblöcke ermittelt sind, konkateniere sie in `codePkt` zum Gesamtcode

$$cod = cod_1 \| \ldots \| cod_{\texttt{blocks}}$$

(5) und komprimiere sie (durch Entfernen der eingefügten Redundanz).

5.2 Aufgaben

Aufgabe 1

Das vorbereitete Programm bietet einen Rahmen für den oben beschriebenen Ablauf eines möglichen Angriffs. Die Oberpunkte (1) bis (6) sind bereits implementiert. Die Aufgabe besteht in der Implementation der für Schritt (3.1) notwendigen Funktion `attackBlock()`. Die Realisierung könnte etwa so aussehen:

```
public BigInteger attackBlock(BigInteger targetBlk, Service serv)
```

→`targetblk`	ein gegebener Schlüsseltextblock
→`serv`	das Serviceobjekt, das unter anderem die RSA-Verschlüsselungsfunktion und den öffentlichen Schlüssel des Angegriffenen enthält.
←`return:`	der passende Codeblock, wenn der Angriff erfolgreich war, sonst `null`.

gibt genau dann `null` zurück, wenn kein passender Codeblock gefunden werden konnte, sonst gibt sie den passenden Codeblock zurück. Ein Codeblock gilt genau dann als passend, wenn er verschlüsselt mit dem Schlüssel des Angegriffenen mit `targetBlk` übereinstimmt.

Aufgabe 2

Die Eintragungen in den Formularfeldern $shl_1 \| \ldots \| shl_{\texttt{blocks}}$ kann in zwei Gruppen eingeteilt werden:

Feldgruppe I:	Betrag	Schlüsseltextblöcke $shl_1 \ldots shl_6$
Feldgruppe II:	Kreditinstitut, BLZ, Kto.-nr, Ort, Datum, Name, Adresse	Schlüsseltextblöcke $shl_7 \ldots shl_{\texttt{blocks}}$

Für die in shl_i verschlüsselten Codeblöcke cod_i ($i \in \{1, \ldots, 6\}$) der Feldgruppe I soll gelten:

Die Codeblöcke enthalten den Code eines Betrages $v \leq DM9.999,-$. Der Betrag ist in folgendem Format in den Codeblöcken abgelegt: $\text{DM} \| * \cdots * \| \ldots v_1 v_0$, wobei $* \cdots *$ für eine bestimmte Anzahl Leerstellen, und v_i für die Ziffern von v steht.

In Aufgabe 1 wurde davon ausgegangen, daß ein Angreifer nur den Schlüsseltext genau eines Formulars erhält. Hört er jedoch das Netz ab, so kann er normalerweise nur *einen Strom von aufeinanderfolgenden Schlüsseltextblöcken* empfangen.

(a) Überlege, wie ein Angreifer aufgrund der Informationen in Feldgruppe I den Beginn eines Formulars erfahren kann, um dann wie oben beschrieben, angreifen zu können.

(b) Implementiere diesen Angriff in der Methode `Aufgabe2` der Klasse `Aufgabe2`. (Hierbei soll nur ein Algorithmus zum Finden des Beginns des Codetextblockes implementiert werden, nicht der gesamte Angriff. Es gelten außerdem die oben angegebenen Suchraumbeschränkungen, d.h. in jedem Schlüsseltextblock ist genau ein Codeblock verschlüsselt.)

Weitere Hinweise zur Implementierung und zu den zur Verfügung stehenden Methoden befinden sich im Quelltext des Rahmenprogramms.

5.3 Diskussion weiterer Sicherheitsschwächen

Durch Erhöhung der Entropie je Block (z.B. durch Verwendung längerer Blöcke) kann der oben beschriebene Schlüsseltext-Angriff entschärft werden. Dennoch kann der Angreifer bestimmte Informationen über den Klartext erhalten.

Beispielsweise bleibt das Jacobi-Symbol des Klartextes beim Verschlüsseln mit RSA erhalten. Denn sei m ein Klartextblock, so gilt (vgl. Versuch ZA.6):

$$\left(\frac{\mathbf{potmod}_{e,n}(m)}{n}\right) \underset{\text{Abschnitt 2}}{=} \left(\frac{m^e (\mathrm{mod}\ n)}{n}\right) \underset{\text{Versuch ZA.6.2}}{=} \left(\frac{m^e}{n}\right)$$

An dieser Stelle wird die Multiplikativität des Jacobi-Symbols ausgenutzt, vergleiche z.B. [Bund_88]:

$$\left(\frac{m}{n}\right)^e \underset{e\ \text{ungerade}}{=} \left(\frac{m}{n}\right)$$

Weiterhin ist ein aktiver Angriff auf RSA bekannt [Davi_82, Denn_84, Ort_91], der in der Betriebsart ECB funktioniert.

Aus diesen Gründen wird empfohlen, RSA nicht im ECB-Modus zu betreiben, sondern in der Betriebsart CBC [Zim1_86]. Dadurch ist es möglich, den zu durchsuchenden Klartextraum ausreichend zu vergrößern und auch die Information über das Jacobi-Symbol des Klartextes nicht preiszugeben. Diese Betriebsart ermöglicht zwar passive Angriffe auf jeden einzelnen Schlüsseltextblock (vgl. ECB), es ist jedoch kein aktiver, zum ECB-Modus vergleichbar schwerwiegender Angriff bekannt, und diese Betriebsart ist nur wenig ineffizienter als die Betriebsart ECB.

6 Probabilistische Verschlüsselung

Der beschriebene passive Angriff durch Raten (Abschnitt 5) kann nur erfolgreich sein, wenn er auf ein deterministisches asymmetrisches Konzelationssystem angesetzt wird. Es allein garantiert, daß derselbe Klartext stets in denselben Schlüsseltext verschlüsselt wird. Vier Jahre nach Veröffentlichung des RSA-Konzelationssystems stellten Goldwasser und Micali das Konzept probabilistischer bzw. indeterministischer asymmetrischer Verschlüsselung vor [GoMi_82]. Der dafür verwendete Chiffrieralgorithmus arbeitet indeterministisch, so daß der Schlüsseltext nicht allein vom Klartext, sondern von weiteren zufälligen Parametern abhängt. Dadurch wird derselbe Klartext praktisch nie ein zweites Mal in denselben Schlüsseltext verschlüsselt. Das von Goldwasser und Micali vorgeschlagene System bietet beweisbar[7] **perfekte komplexitätstheoretische Konzelation**[8]. D.h. für einen in seiner Rechenleistung polynomial beschränkten Angreifer, der einen Schlüsseltext abgehört hat, sind die a priori

[7] Der Beweis setzt voraus, daß **SQRTMOD*** eine Einwegfunktion mit Geheimnis ist (Vergleiche Abschnitt 2 dieses Versuches).

[8] Die Bezeichnung erfolgte in Analogie zur **perfekten informationstheoretischen Konzelation** (perfect secrecy) nach Shannon [Sha1_49]. Vgl. [Pfit_89, S.43]

Wahrscheinlichkeiten für alle Klartexte, daß sie gesendet werden, „gleich“ den a posteriori Wahrscheinlichkeiten, daß sie gesendet wurden[9]. Das von ihnen vorgeschlagene Konzelationssystem ist allerdings in der Praxis unbrauchbar, da die erzeugten Schlüsseltexte gegenüber den verschlüsselten Klartexten um den Faktor 500-1000 länger sind und es gegen aktive Angriffe nicht sicher ist.

Blum und Goldwasser veröffentlichten dann ein effizienteres, asymmetrisches, probabilistisches Konzelationssystem (im folgenden kurz BlG-System), das auf einem kryptographisch starken Pseudozufallsbitfolgengenerator beruht[10]. Damit beschäftigt sich der Versuch.

6.1 Das BIG-System[11]

Die Idee des BlG-Systems ist, eine Vernam-Chiffre mit einem Pseudozufallsbitfolgengenerator zu kombinieren, um damit ein asymmetrisches Konzelationssystem zu realisieren.

6.1.1 Die Vernam-Chiffre

Die Vernam-Chiffre ist zunächst eine symmetrische Chiffre. Sie verwendet zu einem gegebenen Klartext m eine ebensolange zufällige Bitkette b als Schlüssel. Die Verschlüsselung geschieht durch bitweise XOR-Verknüpfung von Klartext- und Schlüsseltextbitkette. Der Schlüsseltext kann dem Empfänger über einen unsicheren Kanal mitgeteilt werden, während für den Schlüssel ein sicherer Kanal erforderlich ist. Der Empfänger gewinnt aus Schlüsseltext und Schlüssel durch erneute XOR-Verknüpfung beider Bitketten den Klartext zurück. Bei diesem System ist der Schlüsseltext genauso lang wie der Klartext, der Schlüssel jedoch auch ($|m| = |c| = |b|$) (Bild 4).

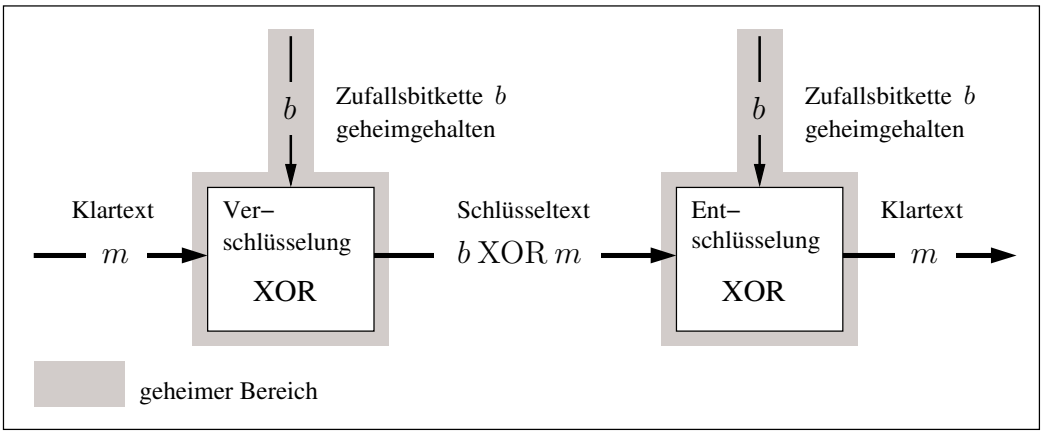

Abbildung 4: Prinzip der VERNAM-Chiffre

[9] „gleich“ heißt hier, daß sich die Wahrscheinlichkeiten weniger als polynomial unterscheiden. Zur exakten Definition eines probabilistischen Kryptosystems siehe [GoMi_84].

[10] zu finden in [BlGo_85]

[11] entspricht dem s^2-mod-n-Generator als asymmetrisches Konzelationssystem. [Pfit_98, Kapitel 3.4.4]

6.1.2 Der Pseudozufallsbitfolgengenerator

Ein Pseudozufallsbitfolgengenerator (PBG) tut folgendes: Er erzeugt aus einem echt zufällig gewählten *kurzen* Startwert (seed) eine *lange* Pseudozufallsbitfolge.

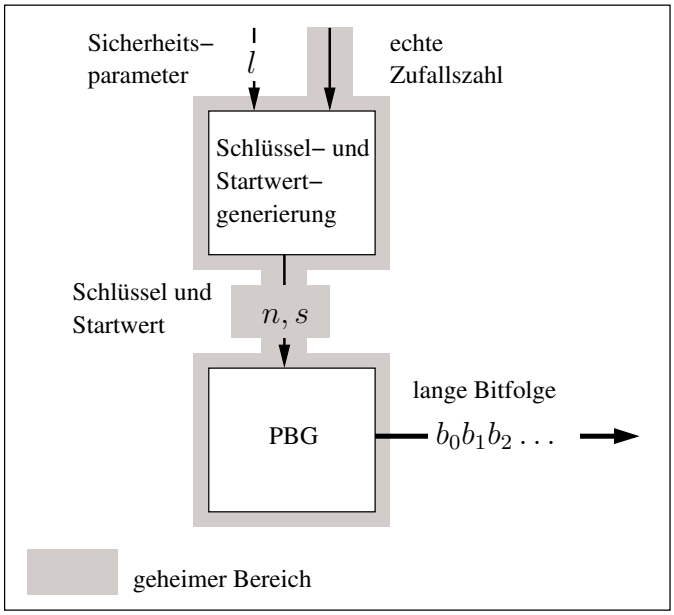

Abbildung 5: Pseudozufallsbitfolgengenerator

Also (siehe Bild 5):

- Es gehört dazu ein Schlüssel- und Startwertgenerieralgorithmus. Er bilde aus dem Sicherheitsparameter l einen Schlüssel n und einen Startwert s der Länge l.

  (An dieser Stelle würde ein Schlüssel *oder* ein Startwert genügen. Aber bei späterer Verwendung in asymmetrischen Systemen ist es interessant, daß der Schlüssel n öffentlich sein darf, während der Startwert s immer geheim bleiben muß.)

- Der eigentliche Bitfolgengenerieralgorithmus PBG ist ein Algorithmus, der mit dem Schlüssel n aus dem kurzen Startwert s eine (beliebig lange) Bitfolge $b_0 b_1 b_2 \ldots$ erzeugt. Allerdings darf nur ein polynomial (in l) langes Anfangsstück dieser Folge verwendet werden.

- PBG ist **deterministisch**, d.h. aus dem gleichen Startwert und Schlüssel wird jedesmal dieselbe Bitfolge.

- PBG ist **effizient**, also in polynomialer Zeit berechenbar.

- PBG ist **sicher**, d.h. die Ergebnisse sind durch keinen polynomialen Test von echten Zufallsfolgen unterscheidbar. (Genauer siehe unten.)

6.1.3 Prinzip des BIG-Systems

Für ein asymmetrisches Konzelationssystem (hier das BlG-System) soll der *PBG* nun die Schlüssel der VERNAM-Chiffre derart erzeugen, daß als Schlüssel für das BlG-System allein der Schlüssel für den *PBG* genügt. Die aus seinem zufälligen Anfangszustand (*seed*) erzeugten Bitketten müssen für die VERNAM-Chiffre „genauso gut" geeignet sein wie zufällige Bitketten, d.h., daß sie in polynomialer Zeit nicht von einer echten Zufallsbitkette zu unterscheiden sind. Derartige *PBG* heißen **kryptographisch stark**[12].

Durch die Verwendung des *PBG* ist die Länge der Schlüsselbitkette unabhängig von der Nachrichtenlänge und verhält sich damit so, wie man es aus praktischen Gründen von einem Schlüssel erwartet. Dieses System bietet ebenfalls beweisbar perfekte komplexitätstheoretische Konzelation, der Schlüsseltext ist jedoch nur um eine additive Konstante länger als der Klartext, so daß dieses Verfahren auch praktisch einsetzbar ist.

Im folgenden wird der (vermutlich) kryptographisch starke *PBG* **s^2-mod-n-Generator** nach BLUM, BLUM und SHUB (i.f. kurz BBS-Generator) beschrieben, der auf einer (vermuteten) Einwegfunktion mit Geheimnis beruht: Der Generator arbeitet mit Quadraten (und der Schwierigkeit des Wurzelziehens bzw. des Entscheidens, ob etwas ein Quadrat ist):

- Der geheime Schlüssel besteht aus den Faktoren p und q von n.
- Der öffentliche Schlüssel ist n.
- Ein Sender und ein Empfänger haben keinen geheimen Startwert s mehr miteinander ausgetauscht, sondern zu jeder Nachricht wählt sich der Sender einen neuen, zufälligen Startwert s. (Dadurch wird die Verschlüsselung *indeterministisch*, d.h. gleiche Klartexte werden nicht notwendigerweise als gleiche Schlüsseltexte verschlüsselt). Damit der Empfänger trotzdem entschlüsseln kann, soll ausgenutzt werden, daß er mittels p und q Wurzeln modulo n ziehen kann.
- Damit der Empfänger auch etwas hat, woraus er Wurzeln ziehen kann, schicken ihm die Sender außer dem bisherigen Schlüsseltext

  $$x_0 \oplus b_0, x1 \oplus b1, \ldots, x_k \oplus b_k$$

  auch den Wert

  $$s_{k+1}$$

  zu, also den Wert, dessen letztes Bit b_{k+1} geworden wäre, wenn man das noch gebraucht hätte.
- Der Empfänger entschlüsselt nun so: Er berechnet aus s_{k+1} rückwärts die ganze Folge der s_i, jeweils

  $$\begin{aligned} s_{i-1} &:= \text{die Wurzel aus } s_i, \text{ die selbst ein Quadrat ist} \\ &= CRA(s_i^{(p+1)/4} \bmod p, s_i^{(q+1)/4} \bmod q).^{13} \end{aligned}$$

[12] Ein effizienter, kryptographisch starker Pseudozufallsgenerator ist von BLUM, BLUM und SHUB angegeben worden [BlBS_86].

Dann kann er die letzten Bits b_i von s_i bestimmen und damit die Klartextbits x_i zurückerhalten.

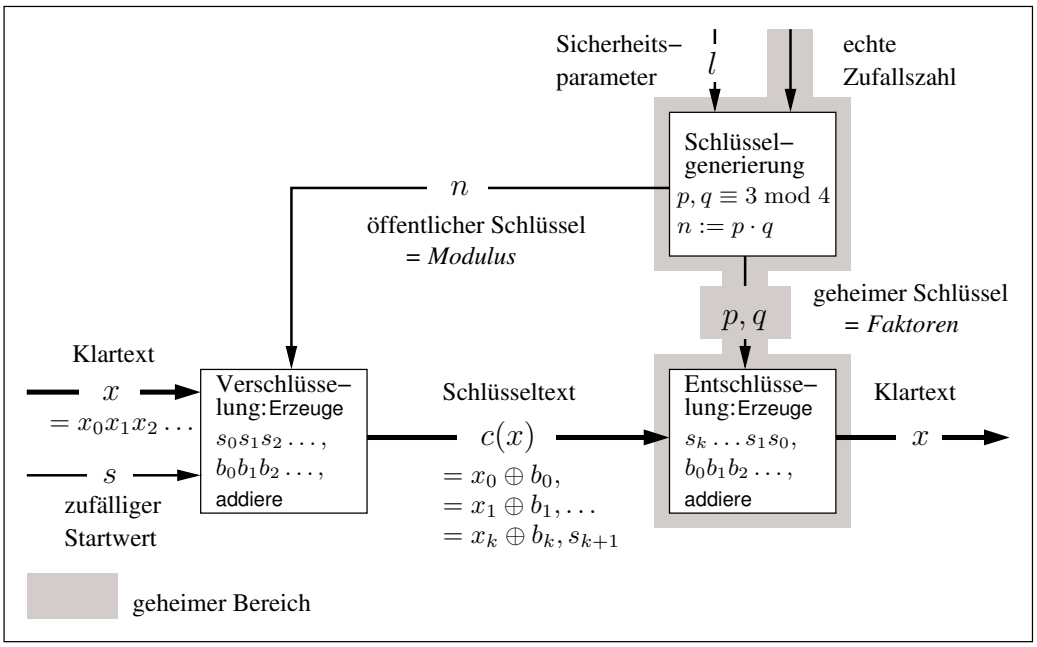

Abbildung 6: s^2-mod-n-Generator als asymmetrisches Konzelationssystem

Im Versuch

$(Z_i), i \geq 0$, sei die Folge der inneren Zustände eines kryptographisch starken *PBG*. Dieser generiere in jedem Zustand genau ein neues (pseudozufälliges) Bit. Ferner sei $k_d(r)$ sein geheimer, $k_c(r)$ sein öffentlicher Schlüssel. Dann existiert ein effizienter Algorithmus **TICK**, der zu jedem Zustand Z_i den Folgezustand Z_{i+1} und gleichzeitig das entsprechende (pseudo-) zufällige Bit b_{i+1} berechnet.

$$(Z_{i+1}, b_{i+1}) := \mathbf{TICK}_{k_c(r)}(Z_i)$$

Weiter ist es (unter kryptographischen Annahmen) für einen in seiner Rechenzeit polynomial beschränkten Angreifer, der $k_d(r)$ nicht kennt, unmöglich, zu gegebenem Z_{i+1} den Vorgängerzustand Z_i zu berechnen. Schließlich existiert ein effizienter Algorithmus **TICKBACK** zur Berechnung des Vorgängerzustandes, wenn $k_d(r)$ bekannt ist:

$$(Z_i, b_i) := \mathbf{TICKBACK}_{k_d(r)}(Z_{i+1})$$

Die Anforderungen an **TICK** legen die Verwendung einer (bewiesenen) Einwegfunktion mit Geheimnis nahe.

[13] Hierfür benötigt man eigentlich noch die Bedingung $p \neq q$. Sie kann bei der Schlüsselgenerierung aber weggelassen werden, da sie bei zufälliger und unabhängiger Wahl von p und q nur in einem exponentiell kleinen Anteil aller Fälle verletzt wäre.

Der Generator nach BLUM, BLUM und SHUB [BlBS_86] verwendet die Funktion **sqrtmod** und soll nun als effizienter kryptographisch starker Pseudozufallsbitfolgengenerator vorgestellt werden:

- Berechne aus der Zufallszahl r zunächst zwei Primzahlen p und q, für die gilt:

$$p \equiv 3 \,(\mathrm{mod}\, 8),\;\; q \equiv 7 \,(\mathrm{mod}\, 8)$$

- Der *geheime Schlüssel* besteht aus den Faktoren p und q von n:

$$k_d(r) = p, q$$

- Der *öffentliche Schlüssel* ist n. $\quad k_c(r) = n = p \cdot q \in \mathrm{BLUM}$ $\quad$ (vgl. Versuch ZA)

- Zur *Erzeugung* einer zufälligen Bitkette $(b_0, b_1, \ldots, b_{t-1})$, wähle (zufällig) einen quadratischen Rest $Z_0 \in \mathbf{QR}_n$ (s. Versuch ZA.6) und wende dann rekursiv die Funktion

$$\begin{aligned}\mathbf{TICK}_n(Z_i) = (Z_{i+1}, b_{i+1}) &:= (\mathbf{sqrmod}_n(Z_i),\ \text{niederwertigstes Bit von } Z_{i+1}),\ \text{d.h.}\\ &:= (Z_i^2 \,(\mathrm{mod}\, n),\ \text{niederwertigstes Bit von } Z_{i+1})\end{aligned}$$

  auf die Zustände Z_i an.

- Zur *Regenerierung* dieser Bitkette ausgehend vom Endzustand Z_t wende rekursiv die Funktion

$$\begin{aligned}&\mathbf{TICKBACK}_n(Z_{i+1}) = (Z_i, b_i)\\ &\quad := (\mathbf{sqrtmod}_n(Z_{i+1}),\ \text{niederwertigstes Bit von } \mathbf{sqrtmod}_n(Z_{i+1}))\end{aligned}$$

  an.

Als Abkürzung für den Algorithmus zur Erzeugung einer t-Bit langen Bitkette $(b_0 \| b_1 \| \ldots \| b_{t-1}) \in \{0,1\}^t$ sei definiert:

$$\mathbf{BBSTick}_{n,t}(Z_0) := (b_0 \| b_1 \| \ldots \| b_{t-1}) \qquad \text{(Generator)} \tag{1}$$

Und für den umgekehrten Algorithmus zur Regenerierung der Bitkette bei gegebenem Endzustand Z_t sei definiert:

$$\mathbf{BBSTickback}_{p,q,t}(Z_t) := (b_0 \| b_1 \| \ldots \| b_{t-1}) \qquad \text{(Regenerator)} \tag{2}$$

Bild 7 veranschaulicht noch einmal das Prinzip. Nach dieser Vorbereitung kann nun das probabilistische Konzelationssystem nach BLUM, GOLDWASSER beschrieben werden. Berechne aus der Zufallszahl r zunächst

- die zwei Primzahlen p und q,
- den öffentlichen Schlüssel $k_c(r)$ und
- den geheimen Schlüssel $k_d(r)$.

Gegeben sei der Klartext: $m = (m_0 \| m_1 \| \ldots \| m_{t-1})$

Chiffrieralgorithmus $\mathbf{C}_{k_c(r)}$:

- Wähle zufällig einen Anfangszustand $Z_0 \in \mathbf{QR}_n$
- Berechne

$$c = \mathbf{C}_{k_c(r)}(m) := (Z_t, c^*) \qquad \text{mit } \; c^* := m \text{ XOR } \mathbf{BBSTick}_{n,t}(Z_0)$$

  (wobei XOR wieder die bitweise exklusiv-oder Verknüpfung zweier Bitketten bezeichnet.)

Dechiffrieralgorithmus $\mathbf{D}_{k_d(r)}$:

- Berechne

$$m = \mathbf{D}_{k_d(r)}(c) := (c^* \text{ XOR } \mathbf{BBSTickback}_{p,q,t}(Z_t))$$

Es sei darauf hingewiesen, daß es effizientere Algorithmen zur Berechnung von $\mathbf{BBSTickback}_{p,q,t}(Z_t)$ als den hier skizzierten gibt (siehe [Bras_88, S. 38]).

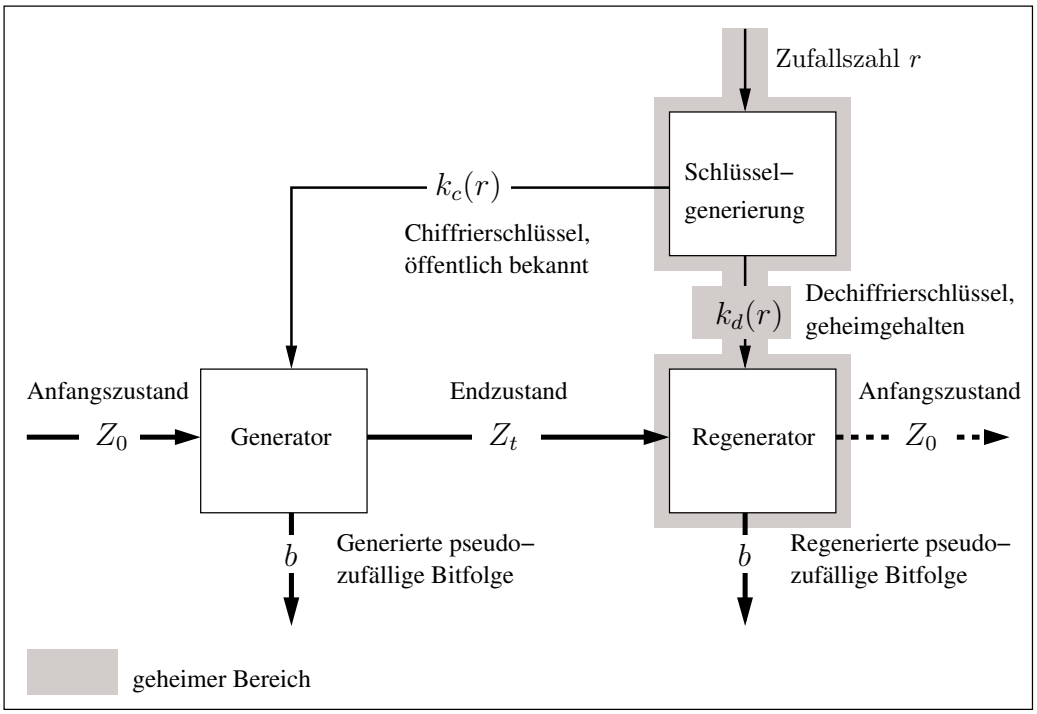

Abbildung 7: Prinzip des s^2-mod-n-Generators nach Blum, Blum und Shub

6.2 Ein aktiver Angriff

Ein aktiver Angriff auf das BlG-System sei an dieser Stelle mehr zum tieferen Verständnis als zur Versuchsvorbereitung vorgestellt. Er beruht darauf, daß prinzipiell nicht ausgeschlossen werden kann, daß zwei BBS-Generatoren mit demselben öffentlichen Schlüssel beim Verschlüsseln (zufällig oder absichtlich) denselben Zustand Z durchlaufen. Würde der Eigner des geheimen Schlüssels beim Entschlüsseln diese Kollision bemerken und das Entschlüsseln verweigern, so müßte er einen (oder evtl. mehrere) erhaltene Schlüsseltexte nachträglich für ungültig erklären. Das ist im BlG-System nicht vorgesehen und kann daher von einem aktiven Angreifer ausgenutzt werden. Genauer: Sei

$$c = (Z_t, c^*) = (Z_t, m \text{ XOR } \mathbf{BBSTick}_{n,t}(Z_0))$$

der (unbemerkt) abgehörte Schlüsseltext eines BlG-Systems, zu dem der t Bit lange Klartext m ermittelt werden soll. Dazu bereite ein Angreifer den Schlüsseltext

$$c' = (Z_t, r)$$

vor, wobei r eine beliebige (z.B. zufällige) Bitkette der Länge t Bit sei. Ein aktiver Angriff ermöglicht dem Angreifer, sich diesen Schlüsseltext c' entschlüsseln zu lassen und

$$\begin{aligned} m' &= r \text{ XOR } \mathbf{BBSTickback}_{p,q,t}(Z_t) \\ &= r \text{ XOR } \mathbf{BBSTick}_{n,t}(Z_0) \qquad \text{(gemäß (1,2))} \end{aligned}$$

zu erhalten. Die Verknüpfung

$$m' \text{ XOR } c^* \text{ XOR } r = m$$

liefert schließlich den zu ermittelnden Klartext.

6.3 Versuchsaufbau und Aufgaben

Konkret sollen nun innerhalb der Chiffrierroutine des geschilderten BlG-Systems der Aufruf zum Erzeugen einer Bitkette und innerhalb der Dechiffrierroutinen der Aufruf zum Regenerieren der Bitkette implementiert werden. Vorgegeben seien dafür:

1. der BBS-Generator mit folgenden Methoden:

```
BigInteger Tick(BBSPublicKey pubKey, BigInteger zustand,
                int bitLen)
```

→`pubkey`	der öffentliche Schlüssel,
→`zustand`	der Anfangszustand Z_0 des BBS-Generators oder `null`, wenn ein zufälliger Zustand des Generators genutzt werden soll,
→`bitLen`	die Länge der angeforderten Zufallsbitfolge,
←`return:`	die kryptographisch sichere Zufallsbitfolge.

```
BigInteger TickBack(BBSPrivateKey privKey, BigInteger zustand,
                    int bitLen)
```

→`privKey`	der geheime Schlüssel,
→`zustand`	der Zustand Z_t des BBS-Generators vor dem Regenerieren der Bitkette oder `null`, wenn ein zufälliger Zustand genutzt werden soll,
→`bitLen`	die Länge der angeforderten Zufallsbitfolge,
←`return:`	die kryptographisch sichere Zufallsbitfolge.

2. die Schlüsselstruktur samt einigen Dienstroutinen des BlG-Systems und
3. ein Testprogramm, mit dessen Hilfe die Ergebnisse überprüft werden können.
4. Die Methodenköpfe der Chiffrier- und Dechiffriermethoden sehen folgendermaßen aus:

   Chiffriermethode:

   ```
   BlGCipherText encryptStream(BlGPublicKey pubKey, BigInteger plainStream,
                               int len)
   ```

   erwartet in `BlGPublicKey` einen öffentlichen Schlüssel eines BlG-Systems. Die Bitfolge des Klartextes wird in `plainStream` in Form eines `BigInteger` und die Länge des Klartextes in Bit in `len` erwartet. Zurückgegeben wird ein `BlGCipherText`-Objekt, das folgende Felder enthält:

   `BigInteger cipherText:` der erzeugte Schlüsseltext,

   `int cipherLen:` die Länge des Schlüsseltextes,

   `BigInteger zustand:` der Zustand Z_t des BBS-Generators nach dem Erzeugen der zur Verschlüsselung genutzten Zufallsbitfolge.

   Dechiffriermethode:

   ```
   BigInteger decryptStream(BlGPrivateKey privKey, BlGCipherText ciph)
   ```

   erwartet in `privKey` den geheimen Schlüssel eines BlG-Systems, mit dem der Schlüsseltext `ciph` verschlüsselt wurde. Der Schlüsseltext enthält die oben beschriebenen Felder `cipherText`, `cipherLen` und `zustand`. Zurückgegeben wird der ermittelte Klartext in Form eines `BigInteger`.

Aufgaben

Aufgabe 3

Implementiere die im Quelltext von `BlG.java` gekennzeichneten Teile der Verschlüsselungsmethode `encryptStream()` und der Entschlüsselungsmethode `decryptStream()`. Teste das Progamm !

Überprüfe die Richtigkeit durch Änderung des zu chiffrierenden Textes „Hello World“! (dieser Text steht in `Aufgabe3.java`) Nutze zum Test die von einer anderen Praktikumsgruppe implementierte Chiffrier- oder Dechiffrierfunktion!

Literatur

[BDPR_98] Mihir Bellare, Anand Desai, David Pointcheval, Phillip Rogaway: Relations Among Notions of Security for Public-Key Encryption Schemes. In: Crypto '98, LNCS 1462, Springer-Verlag, Berlin (1998), 26–45.

[BlBS_86] L. Blum, M. Blum, M. Shub: A Simple Unpredictable Pseudo-Random Number Generator. SIAM J. Comput., Band 15(2) (1986), 364–383.

[BlGo_85] Manuel Blum, Shafi Goldwasser: An Efficient Probabilistic Public-Key Encryption Scheme Which Hides All Partial Information. In: Crypto '84, LNCS 196, Springer-Verlag, Berlin (1985), 289–299.

[Bras_88] Gilles Brassard: Modern Cryptology - A Tutorial. LNCS 325, Springer, Berlin (1988).

[Bund_88] Peter Bundschuh: Einführung in die Zahlentheorie. Hochschultext, Springer, Berlin (1988).

[CrSh1_98] Ronald Cramer, Victor Shoup: A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In: Crypto '98, LNCS 1462, Springer-Verlag, Berlin (1998), 13–25.

[Davi_82] G. Davida: Chosen Signature Cryptanalysis of the RSA (MIT) Public-Key-Cryptosystem. Technischer Bericht TR-CS-82, University of Wisconsin, Milwaukee (October 1982).

[Denn_82] Dorothy Denning: Cryptography and Data Security. Addison-Wesley Publishing Company, Reading (1982), reprinted with corrections, January 1983.

[Denn_84] Dorothy E. Denning: Digital Signatures with RSA and other Public-Key-Cryptosystems. Communications of the ACM, Band 27(4) (1984), 388–392.

[DiHe_76] Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography. IEEE Transactions on Information Theory, Band 22(6) (1976), 644–654.

[GoMi_82] Shafi Goldwasser, Silvio Micali: Probabilistic Encryption & How to Play Mental Poker Keeping Secret All Partial Information. In: 14th Symposium on Theory of Computing (STOC) 1982, ACM, New York (1982), 365–377.

[GoMi_84] Shafi Goldwasser, Silvio Micali: Probabilistic Encryption. Journal of Computer and System Sciences, Band 28 (1984), 270–299.

[Knut_81] Donald Ervin Knuth: Seminumerical Algorithms, Band 2 von *The Art of Computer Programming*. 2. Auflage, Addison Wesley, Reading (1981).

[Maur_89] Ueli M. Maurer: Fast Generation of Secure RSA–Products with almost Maximal Diversity. In: Eurocrypt '89; Houthalen, 10.–13. April 1989, A Workshop on the Theory and Application of Cryptographic Techniques, 305–314.

[Merr_83] Michael John Merritt: Cryptographic Protocols. Dissertation, School of Information and Computer Science, Georgia Institute of Technology (February 1983).

[Ort_91] Andreas Ort: Implementierung eines MIX-Netzes sowie Konzeption und Realisierung des MIX-Netz-Versuches. Diplomarbeit, Institut für Rechnerentwurf und Fehlertoleranz der Universität Karlsruhe (Frühjahr 1991).

[OSI1_85] WG1 (Siegfried Herda) OSIS European Working Group: OSIS Security Aspects. Technischer Bericht, OSIS European Working Group (October 1985), final Report.

[Pfit_89] Andreas Pfitzmann: Diensteintegrierende Kommunikationsnetze mit teilnehmerüberprüfbarem Datenschutz; Universität Karlsruhe, Fakultät für Informatik, Dissertation, Feb. 1989. IFB 234, Springer-Verlag, Heidelberg (1990).

[Pfit_98] Andreas Pfitzmann: Sicherheit in Rechnernetzen: Mehrseitige Sicherheit in verteilten und durch verteilte Systeme (Oktober 1998), TU Dresden; Fakultät Informatik, Script.

[Rabi_78] Michael O. Rabin: Digitalized Signatures. In: R.A. DeMillo, D.P. Dobkin, A.K. Jones, R.J. Lipton (Hg.), Foundations of Secure Computation, Academic Press, N.Y. (1978).

[RSA_78] R. L. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, Band 21(2) (1978), 120–126, reprinted: 26/1 (1983) 96-99.

[Sha1_49] C. E. Shannon: Communication Theory of Secrecy Systems. The Bell System Technical Journal, Band 28(4) (1949), 656–715.

[Simm_79] Gustavus J. Simmons: Symmetric and Asymmetric Encryption. ACM Computing Surveys, Band 11(4) (1979), 305–330.

[Zim1_86] Philip Zimmermann: A Proposed Standard for RSA Cryptosystems. Computer, Band 23(9) (1986), 21–34.