Sicherheit*

Folien zur
Stammvorlesung „Praktische Informatik“
Universität des Saarlandes
WS 97/98, 4+2h

Birgit Pfitzmann

* © Birgit Pfitzmann, Saarbrücken, 1997, 1998

Grobe Gliederung

1 Einführung: Sicherheitsbegriffe

TEIL I Grundlegende kryptographische Systeme

2 Systemtypen
3 Sicherheitsgrade
4 Überblick konkrete Systeme
5 Zahlentheoretische Grundlagen
6 Konkrete Systeme
7 Wichtige Kombinationen und Anwenderschnittstelle

TEIL II Kryptographie in der Praxis

8 Nötig: Zufallszahlen, Schlüsselverteilung
9 Techniken für einfache Protokolle, v.a. Authentikationsprotokolle
10 Juristische Aspekte
11 Produkte

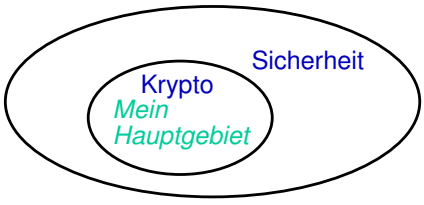
TEIL III Nichtkryptographische Sicherheit

12 Generelle Sicherheitsaspekte
13 Betriebssysteme
14 Datenbanken
15 Netze (außer Kryptographie)
16 Vertrauenswürdiger Entwurf

20.10.97: Heute ...

- Überblick
- Verwaltung
- Einführung

0.A Überblick



- **Sicherheit**: Erreichen gewisser Ziele trotz gewisser Bedrohungen

  meist absichtliche (↔ **Fehlertoleranz**)

- **Kryptographie**:
  - **klassisch** ≈ Geheimschriften
  - **mittelklassisch** ≈ Sicherheit bei Nachrichtenversand
    (v.a. auch „Unterschriften“)
  - **modern**: allg. Mehrparteienprobleme ohne zentrales Vertrauen
    z.B. auch Münzwurf, Wahl, Zahlungssystem

- **Sicherheit** \ **Krypto** v.a.:
  - physisch (Smartcard …)
  - Benutzerschnittstelle (Paßwort …)
  - Prozessor
  - Betriebssystem
  - Einbettung von Kryptographie

  Andere Dimension: „Assurance“
  - Sicherheitskriterien, Evaluation
  - Spezifikationen, Beweise

Warnung 1: Modethema

\+ • Berufschancen (derzeit ;–)
  • Mitreden können

– Modische Sachen sind
  • inhaltlich alt, eher (Industrie-)Standardisierung und Politik
  • schnell veraltet

⇒ **Lehre darf nicht zu „aktuell“ sein**

⇒ **lieber Vertiefung nach wirklichem Interesse entscheiden!**

Warnung 2: Kein Bastelthema mehr

Niemand braucht private Geheimschriften

⇒ Basteln an Basisalgorithmen kein „Berufsbild“

- **Wissenschaft eher:** Neue Anwendungen, Beweisbarkeit …
- **Beruf eher: Systematischer** Einsatz (objektorientiert; Schnittstellen wichtig; Risikoanalyse; Qualitätskontrolle; Benutzerfreundlichkeit; …)
- Kryptographie **benutzt** elementare Zahlentheorie und Komplexitätstheorie.

0.B Verwaltung

Birgit Pfitzmann:
- Geb. 45, Raum 536
- pfitzmann@cs.uni-sb.de

Schein
- 50% aus Übungsaufgaben
- Klausur (oder mündliche Prüfung bei kleiner Anzahl)

Übungen
- Ab 2. Woche, Aufgaben Mo.-Mo.
- 2 Gruppen, 4 Termine zur Auswahl
- Leiter Ammar Alkassar, Daniel Kröning, Jan-Holger (Max) Schmidt, Rainer Schmid

Material
- Folienkopien jeweils im Web
- Reicht für Prüfung
- Generelle kommentierte Literaturliste
- Zusätzliche Zitate zum Vergleich bei den jeweiligen Kapiteln

0.C Literatur

Überblicksartikel Kryptographie

1. Ronald R. Rivest: Cryptography; in: J. van Leeuwen (ed.): Handbook of Theoretical Computer Science; Elsevier 1990, 717-755
   Vorsicht: Sammelband gut, aber teuer!

Bücher Kryptographie

1. Alfred J. Menezes, Paul v. Oorschot, Scott Vanstone: Handbook of Applied Cryptography; CRC Press 1997 (≈ 800 S, 200 $ oder 200 DM)
   - *Endlich mal Top-down Sicht*
   - *Endlich mal wichtigste Angriffe*
   - *Relativ lang und teuer: Nachschlagewerk*
   - *Keine echte Theorie, keine höheren Protokolle*
2. Bruce Schneier: Applied Cryptography: Protocols, Algorithms, and Source Code in C; 2nd ed. Wiley 1996 (≈ 750 S., 60 $).
   - *Große Sammlung von Algorithmen.*
   - *Hübsche Einführungen in Alltagssprache.*
   - *In Theorie naturgemäß Lücken.*
   - *Schon lange Listen von Fehlern.*

3. Gustavus J. Simmons (ed.): Contemporary Cryptology – The Science of Information Integrity; IEEE Press 1992 (≈ 600 S., 55 $).
   - *Im Wissenschaftsbereich ziemlich bekannt.*
   - *Kapitel von verschiedenen Autoren.*
   - *Wichtige Kapitel „Signaturen“, „public-key“ m.E. ziemlich veraltet.*
4. Dominic Welsh: Codes and Cryptography; Oxford University Press 1989 (≈ 250 S., 85 DM).
   - *Lehrbuch.*
   - *1. Hälfte Info- und Kodierungstheorie.*
   - *Für wenig Platz Stoffauswahl nicht schlecht.*
   - *Keine Sicherheitsdefinitionen; Abschnitt „Komplexität“ etwas irreführend:*
     „hart“ in Kryptologie > „worst-case-hart“!
5. Gilles Brassard: Modern Cryptology - A Tutorial; Springer 1988 (≈ 100 S., 30 DM).
   - *Kurz und billig (evtl. aber vergriffen).*
   - *Hübsche Einführung für Anfänger (1988).*
   - *Nicht sehr rigoros.*
   - *Zu sehr, primär sei Kryptologie Verschlüsselung, Rest (Signaturen z.B.) seien „Anwendungen“ davon, selbst wenn dort kein bißchen verschlüsselt wird.*

6. Neal Koblitz: A Course in Number Theory and Cryptography; Springer 1987.
   - *Hübsche Einführung in Zahlentheorie auf Kryptologie-Niveau.*
   - *Kryptologie spärlich.*
7. Oded Goldreich: Foundations of Cryptography (Fragments of a Book); Weizmann Institute, Rehovot, Israel 1995 <ftp.wisdom.weizmann.acil//pub/oded/bookfrag> (≈ 300 S., 0 DM).
   - *Von ca. 1990, nie fertiggestellt.*
   - *Einziger ernsthafter Versuch zu Buch über theor. Kryptographie*
   - *Theoretischer als diese Vorl., aber interessant.*

Sonstige entweder wesentlich spezieller, oder meines Erachtens keine Vorteile gegen obige. Insbes. keine passenden deutschen:

- Bauer 1993 v.a. klassische Verschlüsselung.
- Beutelspacher 1991 lustig, aber eher für Laien, z.T. nicht ganz richtig.
- Horster 1985 einfach schon relativ alt.

Bücher sonstige Sicherheit

1. Dorothy Denning: Cryptography and Data Security; Addison-Wesley 1982 u. 1983.
   - *Klassiker.*
   - *Im Bereich Systematik von nichtkryptogr. Sicherheit kenne ich nichts sinnvolles danach.*

2. Simson Garfinkel, Gene Spafford: Practical UNIX and Internet Security; (2nd ed.) O'Reilly 1996 (≈ 950 S., ? DM)
   - *Wenig generelle Probleme und Lösungstechniken*
   - *Aber konkrete aktuelle Probleme*
3. Philip R. Zimmermann: The Official PGP User's Guide; MIT Press 1995.
   - *Wenig generelle Probleme und Lösungstechniken*
   - *Aber konkrete aktuelle Probleme*
4. William Cheswick, Steven Bellovin: Firewalls and Internet Security; Addison-Wesley1994 (≈ 300 S., ? DM)
   - *Ähnlicher Buchtyp wie 2.*
   - *Firewalls sind insgesamt weniger wichtig*

Im Zahlungssystem- und Ecommercebereich bisher kein brauchbares Buch. Wenigstens mittelgut, aber stark juristisch:

5. Ford/Baum: Secure Electronic Commerce, Prentice-Hall, 1996.

Tagungsreihen u. Journals Kryptologie

Für Überblick über Aktuelles besser als Bücher.

Kryptologie v.a. in

- "Crypto": Advances in Cryptology — Proceedings of CRYPTO ’xy); Springer LNCS,
- "Eurocrypt": Advances in Cryptology — Proceedings of EUROCRYPT ’xy; Springer LNCS.
- Journal of Cryptology (seit 1988, Springer).

Alles in Bibliothek.

Tagungsreihen Sonstige Sicherheit

1. IEEE Symposium on Security and Privacy; IEEE Computer Society Press. (*Älteste Reihe, im Grundlagenbereich angesehenste; ältere Jahrgänge etwas militärbetont*)
2. ACM Conference on Computer and Communications Security; ACM Press. (*Seit 93, Mischung angewandte Krypto. und voriges.*)
3. Computer Security Foundations Workshop, IEEE Computer Society Press. (*Ähnlich 1., Einladungsworkshop*)
4. USENIX Security Symposium
   Praktischer als vorige.

News-Gruppen

Lange nicht alles darin glauben!

Moderiert:

- comp.risks (*Allgemein zu Sicherheit*)
- comp.security.announce (*Reale Sicherheitslücken, von CERT*)

Nicht moderiert:

- sci.crypt (*Kryptographie allgemein*)
- comp.security.misc (*Allgemeinere Sicherheit*)
- alt.privacy, alt.security und alt.security.<x> (*Auch allgemeinere Sicherheit*)

- talk.politics.crypto (*Politische Diskussionen*)

Mailing-Listen u.ä.

- VIS *{GI-FachgruppeVerläßfl. Info.-Sys., unmoderiert}*
  [Un]Subscribe: vis-request@darmstadt.gmd.de
- CIPHER (*on-line Magazin*), vgl.
  http://www.itd.nrl.navy.mil/ITD/5540/ieee/cipher

1 Einführung: Sicherheitsbegriffe

1.1 Überblick

Sicherheit: Erreichen

- gewisser Ziele
- trotz gewisser Bedrohungen bzw.
  unter nicht uneingeschränktem Vertrauen.

Zentrale Anforderungsanalyse:

- **Wer** hat Ziele? („Betroffene“, „Interessensgruppe“)
- **Welche Ziele jeweils?**
- **Wem** / **was** vertraut **dieser** Betroffene für dieses Ziel?

(Bsp. Zahlungssysteme: Kunden ↔ Händler ↔ Bankangestellte ↔ Netz ↔ Hersteller)

1.2 Einteilung von Schutzzielen

Üblichste:

1. **Vertraulichkeit (*Confidentiality*)**
2. **Integrität (*Integrity*)**
3. **Verfügbarkeit (*Availability*)**

„Definition“:

2. & 3 **Dienstziele**: Was garantiert System für *gerade betrachtete* Teilnehmer?
   - Integrität: System tut nichts Falsches. (≈ partielle Korrektheit)
   - Verfügbarkeit: System tut was.

1: Geheimhaltungsziele: Was hält System vor *anderen* geheim?

Bsp.:

- Integrität: Bahn hat keinen Unfall.
  Auch „Safety“ als Pseudogegensatz zu Sicherheit („security“) genannt.
- Verfügbarkeit: Bahn fährt.

Manche unterteilen feiner. (Dann willkürlicher oder anwendungsabhängiger.) Z.B:

1a. Vertraulichkeit von Nachrichteninhalten.

1b. Anonymität/Pseudonymität: Vertraulichkeit von Sender/Empfänger: Anonymität.

1c. Unbeobachtbarkeit: Unbefugte bemerken Ereignis gar nicht.

2a. Integrität von Nachrichteninhalten.

2b. Authentizität: Integrität des Absenders.

(Ohne 2a wenig Sinn ...)

Bei einzelner Anforderungsanalyse nur Gliederungshilfen für konkrete Ziele

Bsp. wieder Zahlungssysteme …

1.3 Vertrauen

Dasselbe oft umgekehrt ausgedrückt:

- „**Angreifermodell“** (*adversary*, *enemy*)
- oder Modell für „**Bedrohungen**“ (*threat*).

Begriff „Vertrauen“ betont drei Aspekte:

1. **Vertrauen ist Beziehung (zweistellig)**:
   - Nicht nur: wird jemand vertraut?
   - Auch: wer vertraut hier?
2. **Defaultwert: man vertraut *nicht***.

   ¡ Vertrauen im Modell *zwingt* reale Betroffene zum Vertrauen !

   „Angreifer“ oder „Bedrohung“ klingt dann hart.
3. **Auch unabsichtliche Fehler.**

1.3.1 Konkrete Angriffs- und Fehlerquellen

Natur, z.B.

- Spannungsausfall
- Bauteile altern
- Überspannung (Blitz, Straßenbahn, …)
- Wassereinbruch

→ **„Fehlertoleranz“**
(Statistische Methoden möglich)

Menschen
(Versehen ↔ Absicht, oft: Lücke + Ausnutzung)

- Außenstehende,
- Benutzer des Systems,
- Betreiber des Systems,
- Wartungsdienst (inkl. Reinigung)
- Produzenten, Entwerfer, Auslieferer
  - des Systems,
  - der Hilfsmittel (Editor, Compiler …)
  - der Hilfsmittel für diese Hilfsmittel,
  - • •

1.3.2 Abstrakte Vertrauensmodelle

Technischer Schutz vor allmächtigem Angreifer unmöglich

⇒ bei Entwurf **Vertrauensmodell**
(≈ maximal tolerierte Angreifer)

Optimales: Mehrseitige Sicherheit (*multi-party security*):
Nur technisch unvermeidliches Vertrauen:

- In eigene Hard- und Software
  - eigene Auswahl
  - selbstgewählte Entscheidungshilfen

  (Problem!)
- In Rechtssystem als Ganzes (Gerichte, Öffentlichkeit derselben …)

Nicht in vorgegebene einzelne andere.
(Insbes.: Große Betriebssystem und Entwürfe derzeit nicht zu sichern ⇒ „trust center“ Illusion.)

⇒ Jeder braucht **Beweismittel** (*evidence*), wie sonst im Leben.

Risikoanalyse: Abwägung

Sicherheit ↔ Preis, Bequemlichkeit.

Völlig ok, solange eigene Sicherheit.

Allgemeine Vertrauensmodelle:

Abbildung von: „Wer und für welches Ziel“ auf „in wen, evtl. mit Vertrauensgrad“.

„In wen“ → technisch oft: In welche Komponenten.

Vertrauensgrade:

- Nicht beliebig viele wirklich angreifende unter nicht ganz vertrauten Komponenten:
  - ***k*-aus-*n*:** Man vertraut: höchstens k–1 von insgesamt n inkorrekt.
  - Allg.: Monotone **Vertrauensstruktur** („Zugangsstruktur“, *access structure*):

    $$V \subseteq P(\{1, \ldots, n\})$$

    mit „P“ Potenzmenge und

    $$A \in V \wedge B \supset A \Rightarrow B \in V$$

    Bedeutung: Man vertraut: In jeder Menge aus V mindestens eine Komponente ok.

- Kein beliebig schlimmes Verhalten:
  - **Komplexitätstheoretisch** beschränkt (bei Kryptographie mehr hierzu).
  - **Rein beobachtend**: Komponenten-spezifikation eingehalten.
  - **Risikoscheu**: Kein
    a) entdeckbares
    b) oder nur kein nachweisbares

    Fehlverhalten.
  - **Finanziell beschränkt**. (Z.B. „kann kein Chipanalyselabor benutzen“.)
  - **Mit / ohne Einflußnahme auf ehrliche Benutzer.**
- Manchmal abhängig von Zeit
  - Z.B. „Workstation-Software nur manipuliert, wenn Benutzer weg“
  - „Mobile-Virus“-Modell: z.B. max. k–1 inkorrekte Komponenten zu jeder Zeit, aber wechselnde.

Anm. zu risikoscheuem Angreifer:

- „b)“ für Systementwurf äquivalent zu

  Zurechenbarkeit oder **Verantwortungs-zuweisung** (*accountability*): Falls Ziel verletzt, Fehlverhalten nachweisbar.
- Eine Ebene genauer: Entdeckung / Nachweis müssen Systemteile sein

  ⇒ Deren Korrektheit für beide Seiten als Ziel, mit normaler mehrseitiger Sicherheit

1.4 Entdecken / verhindern

Vertraulichkeitsverletzung meist

- nicht erkennbar,
- verhinderbar
- nicht rückgängig zu machen.

Dienstzielverletzung meist

- erkennbar
- verhinderbar?
  - Integritätsverletzung ja (im obigen Sinn)
  - Verfügbarkeit nur eingeschränkt
- rückgängig zu machen

  (aber Konsequenzen nicht immer)

1.5 Technisch / juristisch

Zunächst Anforderungen nötig

- Welcher Dienst ist erwünscht?
- Welcher Informationsgewinn ist unbefugt?

Nicht technisch:

- Verfassung
- Gesetze
- Tarifverträge
- Berufsständische Ethik

Trotzdem dann technische Maßnahmen nötig:

Verbot nur wirkungsvoll, wenn

- Einhaltung überprüfbar (und durch Strafverfolgung erzwungen)
- Möglichst Originalzustand wiederherstellbar.

Gerade bei Verlust von Geheimhaltung nicht erfüllt.

1.6 Weiterführende Literatur zu Kap. 1

- Systematische Behandlung selten (sowohl generell als auch bei konkreten Systemen).

  (⇒ oft inkonsequenter Systementwurf, z.B. tolle Kryptographie gegen jemand, dem man wegen anderer Systemteile dann doch vertrauen muß).
- Allg. Sicherheitsbegriffe am ehesten bei Diskussion über Kriterienkataloge (vgl. Kap. Bewertung), z.B. [ITSEC2_91, GIPr1_92, CECGB4_93, Dier2_91, Bisk_93].
- „Trust management“ gerade modern, oft nur für Zertifikate bzw. Applet-Laden (s. später). Vor allem Fragen transitiven Vertrauens.
- Meine Sicht genauer: [PfWa_94].

Bisk_93 Joachim Biskup: Sicherheit von IT-Systemen als "sogar wenn – sonst nichts – Eigenschaft"; VIS'93, GI-Fachtagung Verläßliche Informationssysteme, Vieweg, Wiesbaden 1993, 239-254.

CECGB4_93 Commission of the European Communities: Green Book on the Security of Information Systems; Directorate-General XIII/B; Draft 4.0; Brussels, 1993. (Damals ftp.uni-stuttgart.de:pub/doc/security).

Dier2_91 Rüdiger Dierstein: The Concept of Secure Information Processing Systems and Their Basic Functions; Computer Security and Information Integrity, Proc. 6th IFIP/Sec '90, North-Holland, Amsterdam 1991, 133-149.

GIPr1_92 Präsidiumsarbeitskreis "Datenschutz und Datensicherung" der Gesellschaft für Informatik: Stellungnahme zu den Kriterien für die Bewertung der Sicherheit von Systemen der Informationstechnik (ITSEC) V1.2; Informatik-Spektrum 15/4 (1992) 221-224.

ITSEC2_91 European Communities - Commission: ITSEC: Information Technology Security Evaluation Criteria; Office for Official Publications of the European Communities, Luxembourg 1991 (ISBN 92-826-3004-8).

PfWa_94 Birgit Pfitzmann, Michael Waidner: A General Framework for Formal Notions of "Secure" System; Hildesheimer Informatik-Berichte 11/94, Institut für Informatik, Universität Hildesheim, April 1994, siehe auch <www.semper.org/sirene/>.

TEIL I Grundlegende kryptographische Systeme

Was ist Kryptologie (= Kryptographie)?

- **Klassisch** (etwa ≤ 1975)

  Verschlüsseln = Geheimschriften: Nachrichteninhalte verbergen.
- **Mittelklassisch**

  Alles algorithmische zu Sicherheit in Rechnernetzen.
- **Gegenwärtiger Gebrauch**

  Wie verschiedene Parteien
  - für beliebige Protokollziele zusammenarbeiten können,
  - ohne ganzem System zu vertrauen — meist auch einander nicht,
  - mit algorithmischen Methoden.

Im folgenden eher mittelklassisch.

Abgrenzung zu sonstiger Sicherheit

- Eigenen Rechnern und Algorithmen hier voll vertraut („Alice rechnet ...“)
  - Gegensatz z.B. Betriebssystemsicherheit, physische, Bewertung, …
- Ziele als gegeben angenommen
  - Gegensatz z.B. Datenbanksicherheit: v.a. Spezifikationstechniken …

Abgrenzung „Grundlagen“

- Systeme mit wenig Interaktion
- Entweder direkt sehr wichtig (Verschlüsselung, Signaturen) oder Standardbausteine.
- *Hier nicht* im Sinne „ganz formal“

Einteilungsmöglichkeiten

A. Nach Zweck (≈ Ziele, Vertrauensmodell grob).

B. (Oft) Nach Schlüsseln

(Etwas technischer, maßnahmennäher.)

Bei Systemen mit 2 Parteien:

- **Symmetrisch**: Beide Parteien haben selben „Schlüssel“ (→ selbe Möglichkeiten im System).
- **Asymmetrisch:** Parteien haben verschiedene Schlüssel (→ eine hat mehr Möglichkeiten).

Bei 1 Partei: Überhaupt Schlüssel?

C. Nach Sicherheitsgrad

Vertrauensgrad, vor allem Berechnungsfähigkeiten.

2 Systemtypen

„Typen“ wonach:

- Nach Zweck und Schlüsseln
- Je 1 Standardstruktur
- Etwa wie API (Application Programming Interface) ≈ abstr. Datentyp ≈ Definitionsmodul ≈ Headerdatei ≈ abstrakte Klasse.

Überblick

- **Verschlüsselung** („Nachrichtengeheimhaltung“, „Konzelation“)
  - sym.
  - asym.
- **Authentikation** (Nachrichtenintegrität, „Authentisierung“)
  - sym.
  - Signatursysteme (asym. ++)

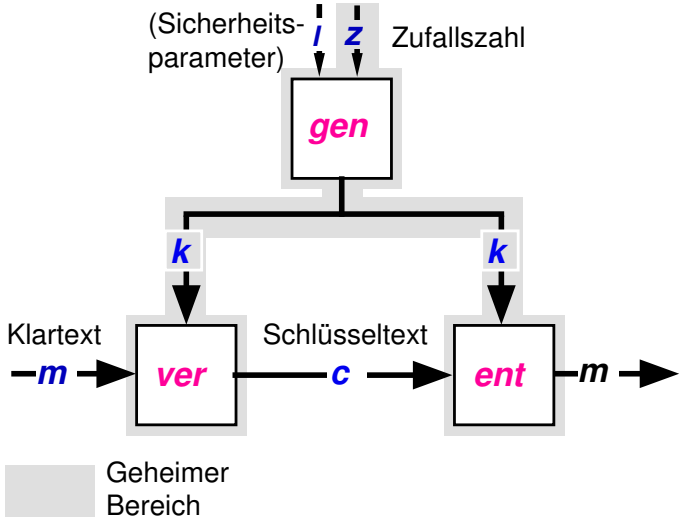
direkt nützlich

eher Bausteine

- **Einwegfunktionen**, auch Falltür-
- **Hashfunktionen**
- **Pseudozufallsgeneratoren**

2.1 Verschlüsselung

2.1.1 Symmetrische Verschlüsselung



- **Ziel:** Geheimhaltung von *m*.
- **Vertrauen:** Alles außer Leitung vertraut. (Aber Algorithmen immer bekannt!)

- **Bsp.:** undurchsichtiger Postsack mit Schloß, zwei gleiche Schlüssel
- **Spezifikation** etwas genauer:
  - 3 Algorithmen
  - Definitionsbereiche aller Parameter
  - $\forall$...: $ent(k, ver(k, m)) = m$
  - Geheimhaltung …
- **Beachte:** Keine Integrität als Ziel / garantiert! (Sack, keine Kiste)

  Falsches Argument: Wenn ein „sinnvolles“ m rauskommt, muß es Original sein.

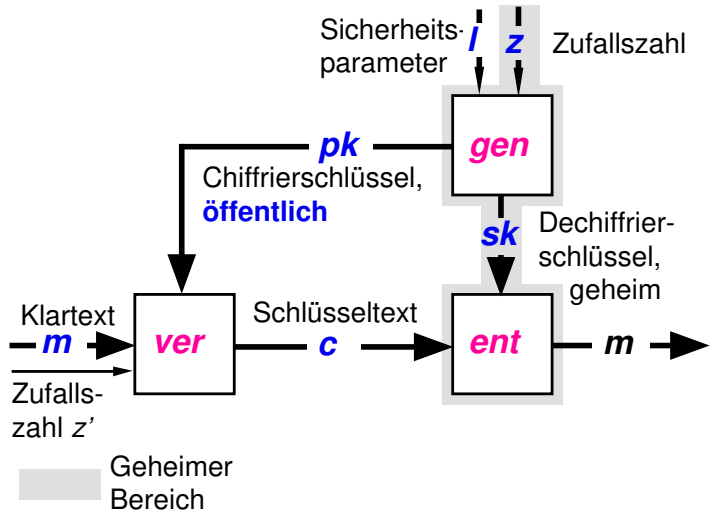
  a) „sinnvoll“?

  b) Echtes Gegenbeispiel: One-time-Pad (Kap. 6.1.1)
- **Für Einsatz noch nötig** (vgl. Kap. 8)
  - Zufallszahlerzeugung
  - **Geheimer und integrer** Austausch von k.

- **Weitere Namen:**
  - Manchmal $k(m)$ für $ver(k, m)$ und $k^{-1}(c)$ für $ent(k, c)$
  - c auch Chiffretext
  - Engl. m: cleartext o. plaintext, k key, c: ciphertext, en- und decrypt o. en- und decipher
  - „private-key“ statt „symmetrisch“
  - *gen*: Schlüsselgenerierung, Schlüsselerzeugung

2.1.2 Asymmetrische Verschlüsselung

Zweck der Asymmetrie: Einfacherer Schlüsselaustausch.



- **Bsp.:** Postsack mit Schnappschloß

- **Hauptunterschied:** Verteilung von pk:
  - Empfänger braucht nur 1 Paar (sk, pk) für alle Sender.
  - **Kein geheimer** Austausch von pk nötig.
- **Schlüsselverteilung:** Austausch von pk muß **integer** sein: Sonst kann Angreifer eigenen Schlüssel pk' statt pk einschleusen.

  Empfänger merkt's nicht mal, wenn Angreifer dann $ver(pk, m)$ weiterleitet. (***Middle-person attack***, „Abfangangriff“.)
- **„Probabilistische Verschlüsselung“:** Wozu Zufallszahl z'?

  Sonst einfacher **Probierangriff**: Angreifer probiert kurze oder sonstwie geratene Nachrichten m^* durch:

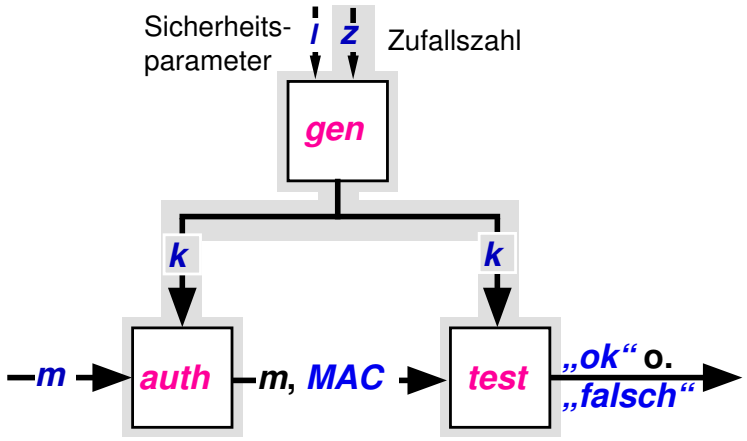
  Ist $ver(pk, m^*) =$ Schlüsseltext?

- **Weitere Namen:**
  - *pk*: öffentlicher Schlüssel, public key, encryption key
  - *sk*: geheimer Schlüssel, secret key, private key, decryption key.

    Anm.: „secret“ o. „private“ manchmal „sym.“
  - asymmetrisch: „public key“, „mit öfftl. Schlüsseln“.

2.2 Authentikation

2.2.1 Symmetrische Authentikation



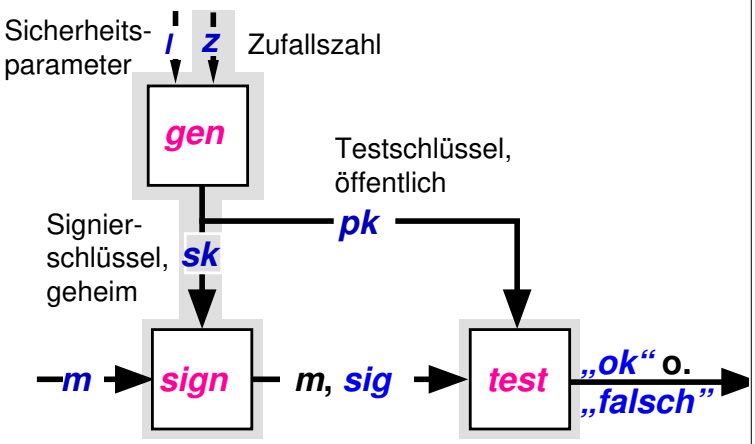
„MAC“ = message authentication code

- **Ziel:** Integrität und Authentizität von *m*.
- **Vertrauen:** Alles außer Leitung vertraut. (Aber Algorithmen bekannt)
- **Bsp.:** ≈ Glasvitrine mit Schloß
- ***test* meist:**

  *MAC** := *auth*(*k*, *m*); prüfe *MAC* = *MAC**

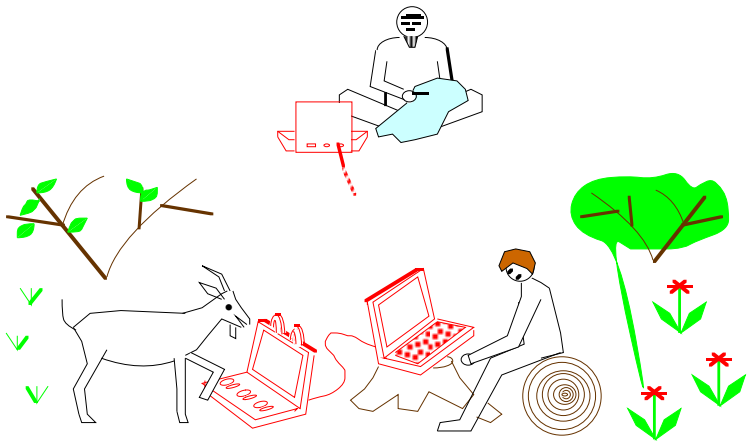
- **Spezifikation genauer:** Analog oben +
  - ∀... *test*(*k*, *auth*(*k*, *m*)) = *ok*
  - ≈ Empfänger akzeptiert kein *m*, das Sender nicht authentisiert hat.
- **Schlüsselaustausch:** Genau wie bei sym. Verschlüsselung: *k* **geheim und integer**.
- **Weitere Namen:**
  - „MAC“ für Verfahren selbst
  - Authentikationscode
  - „keyed hash function“ (nicht empfohlen!)

2.2.2 Signatursysteme



Ziele:

- Signatursystem ist asymmetrische Authentikation (d.h. Auth. mit einfacherer Schlüsselverteilung).
- Aber nicht nur: **Disput vor Dritten** möglich. Bsp:



Ziele genauer:

- **Empfänger:**
  Akzeptable Nachricht erhalten
  ⇒ gewinnt Disput vor beliebigem Dritten.

  Vertrauen: In jeweils diesen Dritten.

- **Unterzeichnerin**:
  Bestimmte Nachricht nie signiert
  ⇒ gewinnt Disput vor beliebigem Dritten.

  Vertrauen: In jeweils diesen Dritten.
- Beide zusammen:

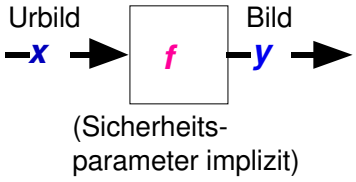
Signierte Nachrichten akzeptabel.

<Ende Ziele u. Vertrauen>

- **Ziele vereinfacht** (für diese Struktur)**:**
  - Empfänger automatisch: Dritter verwendet selben Test.
  - Zusammen: ∀...: *test*(*pk*, *sign*(*sk*, *m*)) = ok.
  - Signierer: Unfälschbarkeit: Ehrlicher Empfänger/Dritter akzeptiert kein *m*, das Sender nicht signiert hat.
- **Schlüsselaustausch:**
  - Für reine asym. Authentikation (d.h. ohne Dispute): Wie bei asym. Verschlüsselung: *pk* **integer**.
  - Für Dispute zusätzlich **konsistent**: Selbst wenn Signiererin betrügt, haben alle Empfänger und Dritte selbes *pk*.
- **Weitere Namen:**
  - Elektronische / digitale Unterschriften /Signaturen
  - *sign*: signieren, unterschreiben, unterzeichnen
  - *test*: verifizieren, prüfen, ...

2.3 Einwegfunktionen u.ä.

2.3.1 Normale Einwegfunktionen



- ***f* leicht, f^{-1} schwer.**
  - Geg. *y*; gesucht beliebiges $x \in f^{-1}(y)$.
  - „Schwer“ ≈ Komplexitätstheorie, aber siehe Kap. 3.
- Oft Zusatzeigenschaften verlangt, z.B.
  - Einwegpermutation: *f* eingeschränkt auf *n*-bit-Strings jeweils bijektiv.
  - Schwächer: grob längenerhaltend
- **Bsp.** (hoffentlich): *f*(*p*, *q*) = *pq* für große Primzahlen.

- **Nicht** verlangt, daß man keine partielle Information über $f^{-1}(x)$ findet!
  - Z.B. dürfte letztes Bit von x erhalten bleiben.
  - Echtes Gegenbeispiel: Falls f einweg, dann auch g mit

    $$g(x, y) = (f(x), y)$$

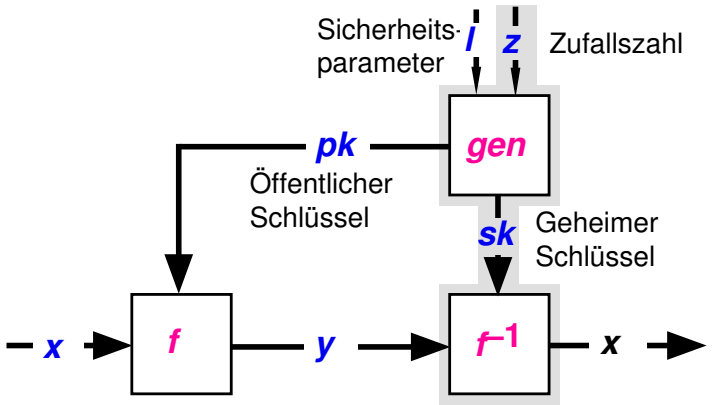
    für jeweils gleich lange x, y.
    Großer Teil des „Klartexts“ sichtbar!

  ⇒ Per se nicht als Verschlüsselung geeignet.
- **Weitere Namen:**
  - Engl. one-way.
  - Deutsch netter: „Einbahn-“ wie „-straße“:
    - ≈ „nur in einer Richtung befahrbar“,
    - nicht „nach einmaligem Gebrauch wegwerfen“.

- **Anm.:** Existenz einer Einwegfunktion f nützlich als **abstrakte kryptographische Annahme**.
  - Sehr schwach, denn aus allem bekannten konkreten kann man Einwegfunktionen bauen.
  - Dann Versuch: Existiert „alles“ unter dieser Annahme?
    - Z.B. Signatursystem bewiesen.
    - Asymmetrische Verschlüsselung nicht bewiesen.
  - Hübschere Annahme wäre P≠NP (noch bekannter, evtl. schwächer). Aber daraus bisher nichts Kryptographisches beweisbar.

2.3.2 Trapdoor-Einwegpermutationen

≈ Grundannahme für asymmetrische Verschlüsselung.



- **Ziele**
  - Pro öffentlichem Schlüssel eine Permutation f_{pk} (= $f(pk, \bullet)$)
  - Algorithmen für f, f^{-1} wie im Bild gegeben,
  - aber f_{pk}^{-1} ohne sk „schwer“.

- **Wieder nicht** per se als Verschlüsselung geeignet!
  - Nicht verlangt, daß man keine partielle Information über $f_{pk}^{-1}(y)$ weiß!
  - Probierangriff geht.
- **Bsp.** (s. später, hoffentlich!): Potenzierungen mod n, d.h. die RSA-Funktionen

  $$f_{(n,e)}: x \to x^e \bmod n.$$

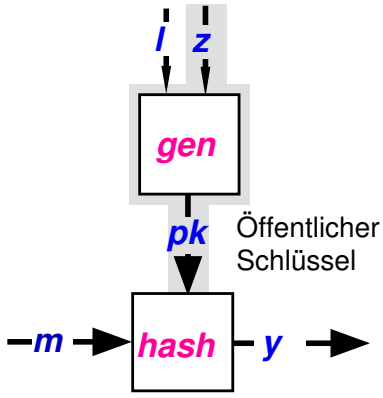
  Umkehrung mit $sk = (p, q)$, wobei $n = pq$.
- **Weitere Namen:**
  - Falltür-Einwegfunktion (besser wäre „Hintertür“)
  - Einwegpermutationen mit Geheimnis.

2.4 Hashfunktionen

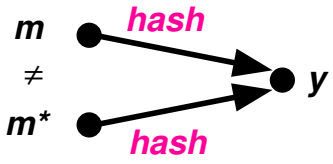
Spezielle Einwegfunktionen:

- Verkürzen Nachrichten
- „Kollisionsfreiheit“.

Dafür theoretisch Schlüssel nötig, obwohl keine Umkehrfunktion:



- **Kollision:**



- **Kollisionsfreiheit:**

  Geg. *pk*. Gesucht *m*, *m**mit

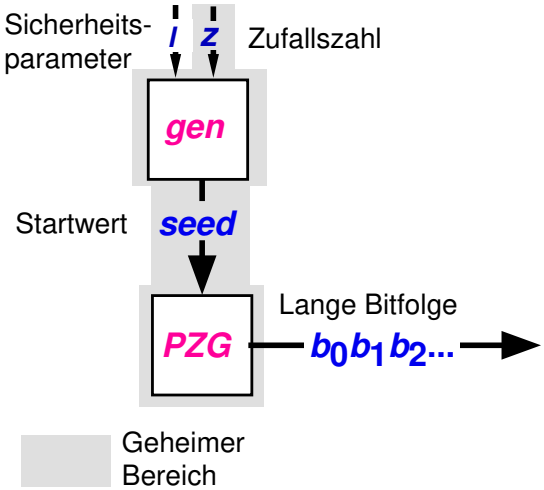
  $$hash(pk, m) = hash(pk, m^*).$$

- **Ohne Schlüssel?** (Einfach *hash*(*m*).)
  - In Praxis meist gemacht.
  - Formal geht kollisionsfrei dann nicht:
    - Kollision existiert wegen Verkürzung
    - $\Rightarrow$ Es *existiert* hocheffizienter Algorithmus *A*, der diese Kollision ausgibt: Konstant! Bsp:

```
PROGRAM FindeKollision;
BEGIN
    m  := 928347829873412381239838;
    m* := 2039959096583804842924;
RETURN (m, m*);
END.
```

- **Weitere Namen:** „Hash“ auch für nur verkürzend oder nur kollisionsfrei.

2.5 Pseudozufallsgeneratoren



- **Ziel:**
  - **Deterministische** Erzeugung
  - von langer, „zufällig aussehenden“ Bitfolge
  - aus kurzem, zufälligen Startwert

    (Von nichts kommt nichts!)

  Etwas genauer: Angreifer „kann nicht unterscheiden“, ob Bitfolge echt oder pseudozufällig.

- **Einsatz:** V.a. statt echter Zufallsbits in Systemen, die sowieso nicht „informations-theoretisch“ (s. Kap. 3) sicher sind.

  (Erzeugung vieler echter Zufallsbits nicht einfach, s. Kap. 8.)

- **Weitere Namen:**
  - engl. meist PRNG = pseudo-random number generator
  - auch Pseudozufalls**zahlen**generator
  - korrekter: Pseudozufalls**bitfolgen-**generator

2.6 Etwas genauer ...

- **Struktur:** Bisher einfache, übliche Systemstrukturen.

  Für selbe Ziele auch kompliziertere Struktur möglich (manchmal z.B. für Beweise oder Zusatzeigenschaften gut), z.B.
  - Algorithmen mit Gedächtnis (z.B. Anzahl bisher signierter Nachrichten)
  - Interaktion
  - Gericht mit an Schlüsselgenerierung beteiligt
- **Ziele etwas ungenau**
  - Z.B.: Verschlüsselungssysteme halten i.allg. *Länge* der Nachricht nicht geheim.
  - Genauere Erfolgsarten des Angreifers in Kap. 3.

2.7 Weiterführende Literatur

Andere Überblicken über Ziele zum Vergleich:

- Buch v. Menezes et al. (vgl. Kap 0.C), Kap. 1
- Buch v. Schneier, Kap. 2.
- Benutzerschnittstellen von Kryptoprogrammen, z.B. PGP (Voreinstellungen + Befehle), s. http://www.ifi.uio.no/~staalesc/PGP/
- Krypto-APIs oder Klassenbibliotheken, z.B. http://www.semper.org/sirene/publ/BaBl_95CryptoMan++.ps.gz

3 Sicherheitsgrade

Vorweg: Immer angenommen: **Angreifer kennen alle verwendeten Algorithmen.**

- **Warum?**
  - Sichere Seite.
  - In offenen Systemen realistisch.
  - Zur Not Bestechung oder physischen Schutz brechen.
- **Sogar absichtlich veröffentlichen:**
  - Erleichtert Sicherheitsevaluation.
  - Verbessert Vertrauen.

Trifft auf manche Telekomalgorithmen nicht zu.

3.1 Erfolgsarten eines Angreifers

Eigentlich jetzt klar (nur noch nicht präzis): Jeweils

Angreifererfolg = Ziel verletzt.

Klassische Unterscheidung, v.a. für Verschlüsselung und Authentikation:

a) **Schlüssel finden** (total break).

b) Zum Schlüssel **äquivalentes Verfahren** (universal break).

c) **Einzelne Nachrichten** entschlüsseln bzw. fälschen.

Einzelne Nachrichten genauer:

- Für **Authentikation**:
  - c1) Eine **gewählte** Nachricht (= *selective break*, selektive Fälschung).
  - c2) **Irgendeine** Nachricht (= *existential break*, existentielle Fälschung).
- Für **Verschlüsselung**:
  - c1*) **Ganze** Nachricht.
  - c2*) **Partielle Information** (z.B. einzelne Bits, Quersumme).

a, b, c1, c1* offensichtlich unakzeptabel.

c2 vermeiden? Etwas stark, aber auf sicherer Seite, und unklar, was sonst auf sicherer Seite.

Beispiele

- Authentikation:
  - Meßdaten ohne Redundanz
  - Unterschriebener Unsinn untergräbt Vertrauen
- Verschlüsselung:
  - Bestimmte Bits in Formularen
  - Wenn nur wenig Klartexte möglich: Jegliche Info hilft zur Unterscheidung.

3.2 Angriffstypen

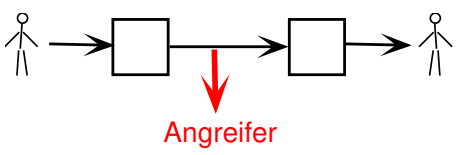
Aus Vertrauensgrad:

- Rein beobachtend: Komponenten-spezifikation eingehalten.
- Mit / ohne Einflußnahme auf ehrliche Benutzer.

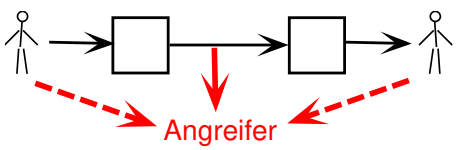
Klassische Einteilung:

A. „Passiv“:

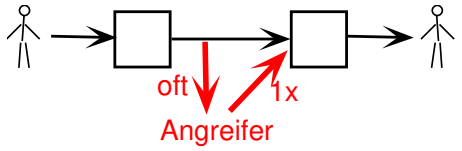
- Bei Verschlüsselung: Nur Zuschauen.
  - **A1. Reiner Schlüsseltext**-Angriff (*ciphertext-only*)



  - **A2. Klartext-Schlüsseltext**-Angriff (*known-plaintext*)



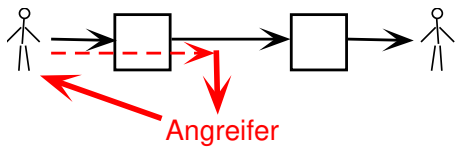
- Bei Authentikation: Nur Zuschauen, **außer** beim Fälschen selbst.



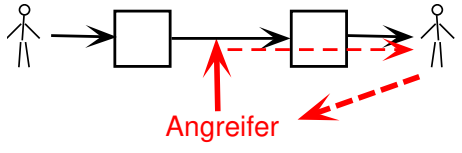
B. „Aktiv“:

Beachte: Einfluß lohnt nur auf Teilnehmer mit Geheimnis.

- **Bei Verschlüsselung**:
  - **B1.** Mit **gewähltem Klartext**: (*chosen-plaintext*)

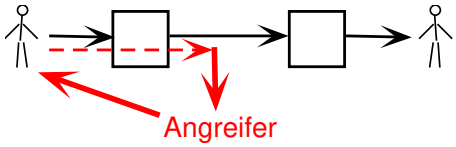


    Bei asym. sinnlos!
  - **B2.** Mit **gewähltem Schlüsseltext**: (*chosen-ciphertext*)

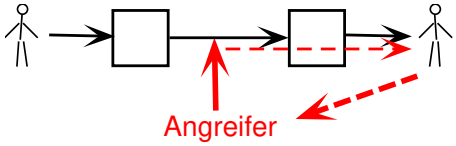


- **Bei Authentikation**:

  B1. Mit **gewählter Nachricht** (*chosen-message*)



  B2. Mit gewähltem MAC:



  Nur 1 bit erfahren; selten betrachtet.

  Bei asym. sinnlos!

Adaptivität:

- Nicht adaptiv: alles aktive auf einmal.
- Adaptiv: schrittweise; angepaßt an Ergebnis des vorigen Schritts.

Eher technische Unterscheidung; praktisch meist adaptiv.

Spezialangriffe für Protokolle
(eher zu Kap. 9, 18 als 6)

- **Replay:** aktiver Angriff durch Nochmalschicken einer Nachricht.
- **Interleaving:** Gleichzeitige Ausführung mehrerer Protokolle und „Mischen“ der Nachrichten.

(In allgemeinem aktiven Angriff enthalten, aber in Praxis oft vergessen.)

3.3 Rechenfähigkeiten

3.3.1 Hauptunterscheidung

a) **Informationstheoretisch** (engl. oft *unconditional*): Sicherheit unabhängig von Ressourcen des Angreifers.
   - Bei Verschlüsselung: Kann Klartext nicht besser als mit A-priori-Wahrscheinlichkeit raten.
   - Bei Authentikation: Kann MAC bzw. Signatur nicht besser als durch blindes Raten fälschen.

   Wo immer möglich, solche Systeme nehmen!

b) **Kryptographisch** (oder „komplexitäts-theoretisch“, engl. *computational*):
   - Sicherheit beruht darauf, daß Angreifer nicht beliebig viel und schnell rechnen kann.
   - Leider auch auf *Annahmen* (s. unten).

3.3.2 Was geht informationstheoretisch?

\+ **Symmetrische** Verschlüsselung u. Authentikation: (s. später).

– **Asymmetrische nicht.**

(Signatursysteme im Prinzip, aber viel kompliziertere Struktur und nicht effizient.)

– **Einweg-, Hash-, PRNG** („sinnvolle“) nicht

Warum nicht?

Vollständige Suche: (*brute-force search*)

- Bsp. Signatur-Fälschen:
  - Angreifer weiß: *test*(*pk*, *m*, *sig*) = *„ok“*.
  - Probiere *sig* nach steigender Länge; terminiert.
- Oder „universal break“:
  - Probiere *z*, bis *gen*(*z*) = (*sk*, *pk*) mit richtigem *pk*.
  - Oft gibt’s Prädikat *paßt*(*sk*, *pk*); probiere *sk*

Angriffe passiv!

3.3.3 Mehr über kryptographische Sicherheit

Fragen:

- Details von „schwer“ in Kryptographie
- Wieviel Angreiferressourcen maximal tolerierbar?
- Dann beweisbar?

A. Angriffe durch vollständige Suche

Wenn gesuchtes Objekt von Max-Länge l^*:

$\leq 2^{l^*}$ Probierschritte.

\+ **Exponentiell** $\Rightarrow$ z.B. für $l^* > 100$ zu aufwendig.

– Angreiferaufwand ca. 2^{l^*} ist Maximum, was wir hoffen können.

Angreiferaufwand genauer:

1. Jeder Probierschritt „leicht“ (legaler Algorithmus, z.B. *test*, *gen*.)
2. Falls l^* nicht offensichtlich, aus l herleitbar:
   - Länge von z, sk, $pk \leq$ Laufzeit($gen(l)$)
   - Länge von $sig \leq$ Laufzeit($sign(pk, m)$).

Z.B. wenn

- „leicht“ = polynomial
- pol_1 Laufzeit von *gen*,
- dann Aufwand zum Suchen von z

$$2^{l^*}\, pol_1(l) = 2^{pol_1(l)}\, pol_1(l).$$

(In EXPTIME.)

Sogar NP-leicht: „Durchprobieren“ $\rightarrow$ „raten“.

(Davon nicht schneller, nur zweifelhafter!)

B. Nicht nur Worst-case-Komplexität

- In Komplexitätstheorie normale Def.:

  „Problem mit Aufwand x lösbar

  :$\Leftrightarrow$ Algorithmus löst alle Instanzen.“
- Hieße:

  „Brechen geht nicht mit Aufwand x

  :$\Leftrightarrow$ Es *gibt* Instanzen, die nicht gebrochen werden.“

  (Aber die meisten vielleicht doch!).

  Für Sicherheit unbrauchbar, ebenso Average-case-Komplexität.
- Ziel: Problem **fast überall** schwer, d.h. bis auf verschwindenden Bruchteil der Fälle.

  Konkret z.B.
  - *W'keit* $\leq 2^{-l}$ (**„exponentiell klein“**)
  - *W'keit* $\leq 1/pol(l)$ für jedes Polynom *pol* (**„vernachlässigbar“**)

C. Probabilistische Algorithmen

Muß man für Angreifer erlauben.

D. Leicht und schwer

- Legale Algorithmen (*gen*, *ver*, *ent*, …): leicht (*easy*)
- Brechen: schwer (*infeasible*)

Komplexitätstheorie meist asymptotisch:

leicht := poly(l)

schwer := nicht poly(l)

Warum?

a) Kaum Aussagen für feste l bekannt.

b) Schwerer als exponentiell geht nicht, s. A.

c) Klasse „polynomial“ ist
   - abgeschlossen gegen übliche Kombination von Programmen (Einsetzen von Polynomen in Polynome ergibt Polynome).
   - Relativ maschinenunabhängig.

Für Praxis würde bekanntes Polynom hohen Grades für Angreifer auf realistischer Maschine reichen.

E. Echt beweisbare Sicherheit?

Bei *kryptographischer* Sicherheit bisher nicht!

- Beweis würde P≠NP implizieren.

  Hoffentlich richtig, aber kann man nicht drauf warten.

⇒ **Annahmen**kryptographische **machen**, unter denen beweisen.

Nutzen?

- Annahmen möglichst gut untersucht.

  (im Gegensatz zu neuem Kryptosystem)
- Annahme möglichst kompakt

  (Kryptosystem komplex, aktive Angriffe, partielle Information ...)

Z.B.

+ „Faktorisieren schwer“,
+ „Existenz von Einwegfunktion“.
– „P≠NP“ wäre netter, aber wegen „fast-überall-schwer“ geht's bisher nicht.

F. Beweisziel

Wenn Angreiferalgorithmus Kryptosystem brechen kann,

dann kann man das als schwer angenommene Problem lösen.

Beweis meist konstruktive **Reduktion**. Etwas allgemeiner als in Info 3:

- Angenommenen Angreiferalgorithmus *A* beliebig oft als Unterprogramm aufrufen.

  PROGRAM *brichAnnahme*;
  USES PROCEDURE *A*(<*parameters_A*>);
  …
  BEGIN *brichAnnahme*
  …
  CALL *A*(…);
  …
  CALL *A*(…);
  …
  END *brichAnnahme*.

  (Theor. Begriffe „Orakel“; „Turing-Reduktion“)
- Beachte probabilistischen und (inter)aktiven Angreifer.

Bsp. in Kap. 6.

3.3.4 Notation und Beispieldefinition

A. Probabilistische Algorithmen

P(*E***)**: W'keit des Ereignisses *E* (wenn W'raum klar)

$x \leftarrow D$: Zuweisung eines Werts an Variable *x* mit W'verteilung *D*

Speziell

$x \in_{\mathbf{R}} M$: Zufällige und gleichverteilte Auswahl von *x* aus Menge *M*

$x \leftarrow A(in)$: Ausführen von prob. Algo. *A* mit Eingabe *in*, zuweisen an *x*

P(*präd*(*x*) **::** $x \leftarrow A(in)$**)**: W'keit, daß *x* das Prädikat *präd* erfüllt, wenn es mittels $A(in)$ errechnet wurde.

; Hintereinanderausführen v. Algorithmen

$P(\ldots(l)) \leq \mathbf{1/poly(}l\mathbf{)}$: W'keit vernachlässigbar klein:

$\forall$Polynome *pol* $\exists l_0\ \forall l \geq l_0$: $P(\ldots(l)) \leq 1/pol(l)$.

Genaues Modell z.B.:

Probabilistische Turingmaschine:

- Extra-Band voll Zufallsbits gegeben, read-only, jedes Bit nur einmal.
- Laufzeit für feste Eingabe ist Zufallsvariable. Meist Erwartungswert betrachtet, d.h.
  - polynomial in *l*: E'wert polynomial
  - Manchmal deterministisch poly verlangt.

B. Definition Einwegfunktion als Beispiel

Funktion f: $\{0,1\}^* \to \{0,1\}^*$ einweg gdw.

- **Effektiv:** $f \in$ P (poly. berechenbar)
- **Sicherheit** (nicht invertierbar)

  $\forall$ prob. poly. Algo. A (Angreifer)

  $P\big(f(x) = y ::$ (W'keit v. Angreifer-ziel wenn…)

  $x^* \in_R \{0,1\}^l$; (Passiver Angriff: Brave wählen …)

  $y := f(x^*)$; (")

  $x \leftarrow A(l, y)\big)$ (Angreifer rechnet)

  $\leq 1/\text{poly}(l)$ (Vernachlässigbar)

Anm.: Bsp. $f(p, q) = pq$ (Kap. 2.3.1)

Definitionsbereich nicht $\{0,1\}^*$:

- Entweder erweitern: $f(s) = s_1 s_2$,

  wobei s_1, s_2 Hälften des Strings s als Binärzahlen.

  (Ann. wohl äquivalent, aber nicht trivial.)
- Oder Folge $(In_l)_{l \in \mathbb{N}}$ dazu,
  - $f: \bigcup_l In_l \to \{0,1\}^*$
  - Zeile „passiver Angriff": $x^* \in_R In_l$;
- Noch besser: Eingabegenerieralgorithmus *gen* dazu (auch für Praxis!).
  - f auf Bildbereich von *gen* definiert
  - Zeile „passiver Angriff": $x^* \leftarrow gen(l)$;

Viele Definitionen in Kryptographie ähnlich zu einfach für echte Systeme.

3.4 Weiterführende Literatur

Üblicherweise Sicherheitsbegriffe nur im Kontext spezieller Systemtypen; deshalb sinnvolle Lit. erst in Kap. 6.

Vgl. höchstens:

- Eher zu wenig:
  - Buch von Menezes et al., Kap. 1.13.
- Schon viel zu viel:
  - Oded Goldreich: On the Foundations of Modern Cryptography; Crypto '97, LNCS 1294, Springer-Verlag, Berlin 1997.
  - Birgit Pfitzmann: Digital Signature Schemes; LNCS 1100, Springer-Verlag, Berlin 1996, Kap. 4 u.5.4.
  - Birgit Pfitzmann: Kryptographie; Skripten des Fachbereichs Mathematik, Informatik, Naturwissenschaften Nr. 5, Universität Hildesheim, April 1995.

4 Überblick konkrete Systeme

≈ Gliederung von Kap. 6

Einwegfunktionen nur implizit

	inf osi	Be-weis	Basis
Sym. Verschlüsselung			
• One-time-Pad	X[1]	X	
• DES			Chaos
• DES als Blockchiffre			"
• Verschlüsselungsmodi (aller) Blockchiffren			"
Asym. Verschlüsselung			
• RSA-Verschlüsselung			Fakt.
• ElGamal-Verschl.			DLog
• Variante: Diffie-Hellman--Schlüsselaustausch			"
• Ausblick (*inkl. Def.*)		(X)[2]	

1 Schlüssellänge sehr problematisch

2 Halbwegs effizient nur gegen passive Angriffe

Sym. Authentikation			
• Ein Auth.code	X[3]	X	
• DES-Auth.			Chaos
Signatursysteme			
• RSA-Signaturen			Fakt.
• ElGamal-artige			DLog
• Ausblick: (*inkl. Def.*): GMR-artig; Fail-stop		X; X	
Hashfunktionen			
• Aus Blockchiffre			Chaos
• MD-Versionen (*Verweis*)			"
• ChHP-Hash (*mit Beweis*)		X	DLog
Pseudozufallsgen.			
• aus Blockchiffre			Chaos
• Schiebereg. erwähnen			"
• BBS (*ohne Beweis*)		X	Fakt.
• Ausblick (*zur Def.*)			
Verweise auf Sonstiges			

[3] Schlüssellänge viel geringeres Problem, benutzen!

5 Zahlentheoretische Grundlagen

5.1 Rechnen in Restklassenringen $\mathbb{Z}_n$

$\mathbb{Z}_n$: Restklassenring modulo n.

5.1.1 Grundlegende Definitionen

- **Zeichen „|“** für „ist Teiler von“: Für $a, b \in \mathbb{Z}$:

$$a \mid b \;:\Leftrightarrow\; \exists\, z \in \mathbb{Z}: b = a \bullet z$$

- **Kongruenzrelation „≡“**: Für $a, b \in \mathbb{Z}$:

$$a \equiv b \bmod n \;:\Leftrightarrow\; n \mid (a - b).$$

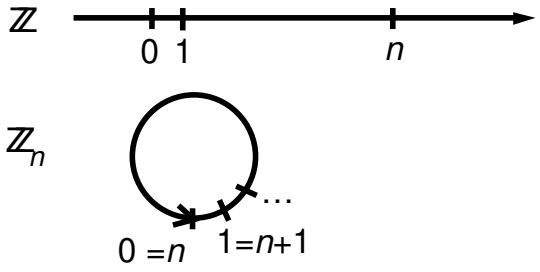
Es gilt $a \equiv b \bmod n \Leftrightarrow$ selber Rest bei Division durch n.

Bsp.: $27 \equiv 132 \bmod 5$,
denn $5 \mid (132\text{-}27) = 105$,
bzw. beide Rest 2 mod 5.

- **Zu jedem $a \in \mathbb{Z}$ Restklasse**

$$\overline{a} := \{b \in \mathbb{Z} \mid b \equiv a \bmod n\}.$$

$\mathbb{Z}_n := \{\overline{a} \mid a \in \mathbb{Z}\}$ Menge dieser Klassen.



Bsp.: $\overline{27} \pmod 5 = \{\ldots, -8, -3, \mathbf{2}, 7, 12, \ldots\}$

- **Operationen auf Restklassen:**
Es gilt

$$a_1 \equiv a_2 \bmod n \;\wedge\; b_1 \equiv b_2 \bmod n$$
$$\Rightarrow \quad a_1 \pm b_1 \equiv a_2 \pm b_2 \bmod n$$
$$\wedge \quad a_1 \bullet b_1 \equiv a_2 \bullet b_2 \bmod n,$$

Sogenannte **Kongruenzeigenschaft** (leicht zu zeigen).

$\Rightarrow$ Man kann Klassen addieren u.ä.:

$$\overline{a} \pm\bullet\, \overline{b} := \overline{a \pm\bullet b}$$

(2 beliebige Elemente nehmen, immer selbes Ergebnis.)

Bsp. 1: Modulo 5:
$27 \bullet 318$
$\equiv 2 \bullet 3 = 6 \equiv 1.$

Bsp. 2: Modulo 11:
$13^4 + 34^3$
$\equiv 2^4 + 1^3 \equiv 5 + 1 \equiv 6.$

- **Vertretersystem** (Repräsentantensystem):
:= Eine Zahl pro Klasse.
  - Für $\mathbb{Z}_n$ meist $\{0,\ldots,n-1\}$
  („**kleinstes nichtnegatives** V.system“)
    - MOD-Operator bildet darauf ab.

- Auch „**symmetrische**“ Repräsentation:
  (Z.B. zum Rechnen von Hand):
  - Für n ungerade:
    $$\{-(n-1)/2, \ldots, 0, \ldots, (n-1)/2\}.$$
  - Für n gerade:
    $$\{-n/2+1, \ldots, 0, \ldots, n/2\}.$$

  Bsp.: Modulo 11:
  $$10^4 + 31^3 = ((-1)^4 + (-2)^3) = 1 - 8 = 4.$$

- **Rechenregeln:** Die meisten üblichen gelten.

  Genauer: $\mathbb{Z}_n$ ist kommutativer Ring mit 1.

  Bsp.: $a(-c + b) \equiv ab - ac \bmod n$.

5.1.2 Algorithmen

- **Addition, Subtraktion, Multiplikation** in $\mathbb{Z}_n$ leicht:
  - Mit Langzahlen aus mehreren Rechnerwörtern rechnen etwa wie mit Dezimalzahlen auf Papier („Schulmethoden“):

    „Ziffern“ bei 32-bit-Worten:
    $$\{0, 1, \ldots, 2^{32}-1\}.$$

    Zunächst in $\mathbb{Z}$: Sei L Länge von n:
  - Aufwand von „**+**“ und „**–**“: $O(L)$
  - Aufwand von „•“ und „/“: $O(L^2)$.
  - $(+, -, \bullet)$ in $\mathbb{Z}_n$ genauso.
    - Insbesondere
      $$a + b \bmod n$$
      ohne Division durch n, da $a + b \le 2n$
      $\Rightarrow$ höchstens 1 mal n abziehen.
    - „•“ klar: 1 Mult, 1 Mod.

  Anm.: $\exists$ schnellere Algorithmen, aber für Zahlen wie in Kryptographie ist Unterschied gering.

- **Exponentiation modulo n** auch leicht:

  Etwa $a^b \bmod n$:

  – b mal Multiplizieren viel zu langsam.

  \+ „**Square-and-multiply**“-Algorithmen: Exponent (binär-)stellenweise bearbeitet: Sei $b = (b_{L-1} \ldots b_0)_2$ Binärdarstellung. Hier „von links“:

  Bsp.: $7^{22} \bmod 11$. $22 = (10110)_2$.
  $$7^{(1)_2} := 7;$$
  $$7^{(10)_2} := 7^2 \equiv 5;$$
  $$7^{(101)_2} := 5^2 \bullet 7 \equiv 3 \bullet 7 \equiv -1;$$
  $$7^{(1011)_2} := (-1)^2 \bullet 7 \equiv 7;$$
  $$7^{(10110)_2} := 7^2 \equiv 5.$$

Allg.: Der Reihe nach $a^{(b_{L-1})_2}$, $a^{(b_{L-1}b_{L-2})_2}$ usw. berechnen.

Dabei aus $a^{(b_{L-1}b_{L-2} \ldots b_{L-i})_2}$ nächster Wert, je nach nächstem Exponentenbit, als

$$a^{(b_{L-1}b_{L-2} \ldots b_{L-i}\mathbf{0})_2} = a^{2\bullet(b_{L-1}b_{L-2} \ldots b_{L-i})_2} = (a^{(b_{L-1}b_{L-2} \ldots b_{L-i})_2})^2;$$

$$a^{(b_{L-1}b_{L-2} \ldots b_{L-i}\mathbf{1})_2} = a^{2\bullet(b_{L-1}b_{L-2} \ldots b_{L-i})_2+1} = (a^{(b_{L-1}b_{L-2} \ldots b_{L-i})_2})^2 \bullet a.$$

Also

- pro Binärstelle quadrieren (square),
- bei „1“ zusätzlich multiplizieren (multiply).

5.2 Multiplikative Gruppe $\mathbb{Z}_n^*$

- **Multiplikative Gruppe** R^* (allg. in Ring R) := Menge aller Elemente, die Inverses haben.

  Leicht zu zeigen: wirklich Gruppe:

  $$(ab)^{-1} \equiv a^{-1}b^{-1}$$

  u.ä.

- Es gilt:

  $$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \text{ggT}(a, n) = 1\}.$$

  (Beweis ergibt sich unten.)

  Bsp.: $\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$.

Algorithmen

- **Invertieren** leicht: **Erweiterter Euklidscher Algorithmus.** Berechnet allgemein zu $a, b \in \mathbb{Z}$
  - ggT und
  - Zahlen u, v mit

    $$u\,a + v\,b = \text{ggT}(a, b).$$

Hier am Beispiel: $a = 11$, $b = 47$.

1. **Normaler Euklidscher Algorithmus:** Immer vorigen Divisor durch vorigen Rest teilen:

   $$\begin{aligned} 47 &= 4 \cdot 11 + 3; \\ 11 &= 3 \cdot 3 + 2; \\ 3 &= 1 \cdot 2 + 1. \end{aligned}$$

2. **Nun ab letzter Zeile** ggT als Linearkombination von Divisor und Dividenden der jeweiligen Zeile darstellen:

$\mathbf{1} = \mathbf{3} - 1\cdot\mathbf{2}$	(Jetzt 2 durch 3 und 11 ersetzen.)
$= \mathbf{3} - 1\cdot(\mathbf{11} - 3\cdot\mathbf{3})$ $= -\mathbf{11} + 4\cdot\mathbf{3}$	(Jetzt 3 durch 11 und 47 ersetzen.)
$= -\mathbf{11} + 4\cdot(\mathbf{47} - 4\cdot\mathbf{11})$ $= 4\cdot\mathbf{47} - 17\cdot\mathbf{11}$.	(Probe: 4•47= 188, 17•11=187.)

(Noch optimierbar, v.a. Speicherplatz; siehe z.B. Knuth: Seminumerical Algorithms, 1981.)

- **Invertieren** von a mod n:
  Mit erweitertem Euklidschen Algorithmus u, v mit

  $$u\,a + v\,n = 1.$$

  Dann offenbar

  $$u\,a \equiv 1 \bmod n,$$

  d.h. u ist Inverses von a.

  Unser Beispiel: $1 = 4\cdot\mathbf{47} - 17\cdot\mathbf{11} \Rightarrow$

  $$11^{-1} \equiv -17 \equiv 30 \bmod 47.$$

  (Und $47^{-1} \equiv 4 \bmod 11$.)

- **Bruchschreibweise** erlaubt, z.B.

  $$\frac{3}{11} \equiv 3\cdot 11^{-1} \equiv 3\cdot 30 \equiv -4 \bmod 47.$$

  (Probe: $11\cdot(-4) \equiv -44 \equiv 3 \bmod 47$.)

- **Damit Charakterisierung von $\mathbb{Z}_n^*$ bewiesen:**

  $$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \text{ggT}(a, n) = 1\}:$$

  - Oben für jedes a mit $\text{ggT}(a, n) = 1$ Inverses bestimmt.
  - Umgekehrt: Wenn es Inverses gibt:

    $$\begin{aligned} & a\cdot a^{-1} \equiv 1 \bmod n \\ \Rightarrow\quad & n \mid (a\cdot a^{-1} - 1) \\ \Rightarrow\quad & \exists v \text{ mit } a\cdot a^{-1} - 1 = v\cdot n \\ \Rightarrow\quad & a\cdot a^{-1} - v\cdot n = 1. \end{aligned}$$

    Hätten a und n gemeinsamen Teiler d, so würde d auch 1 teilen ↯.

- **Elementanzahl** von $\mathbb{Z}_n^*$
  Sei

  $$\phi(n) := |\{a \in \{0,\ldots,n-1\} \mid \text{ggT}(a, n) = 1\}|.$$

  (**Eulersche Phi-Funktion**, *engl. totient oder phi function*.) Also

  $$|\mathbb{Z}_n^*| = \phi(n).$$

  Bsp.: $\phi(10) = 4$.

Allgemeiner:

$$\phi(p) = p - 1 \qquad \text{für } p \text{ prim.}$$

$$\mathbb{Z}_p^* = \{1, \ldots, p-1\}.$$

Und für $n = pq$:

$$\phi(pq) = (p-1)(q-1)$$

wenn p, q prim, $p \neq q$.

Bew.: *Nicht* zu pq teilerfremd sind:

$\{0\}, \{1p, 2p, \ldots, (q-1)p\}, \{1q, 2q, \ldots, (p-1)q\}$.

Anzahl: $1 + (q-1) + (p-1) = p + q - 1$.

$\Rightarrow |\mathbb{Z}_{pq}^*| = pq - p - q + 1 = (p-1)(q-1)$.

Anm.: Analog zeigt man

$$\phi(p^r) = (p-1)p^{r-1}.$$

5.3 Körper $\mathbb{Z}_p^*$

Modulo Primzahlen p:

$$\mathbb{Z}_p^* = \{a \in \mathbb{Z}_p \mid \text{ggT}(a, p) = 1\}$$
$$= \mathbb{Z}_p \setminus \{0\}.$$

Also $\mathbb{Z}_p$ Körper.

Wichtigste Eigenschaften:

- $|\mathbb{Z}_p^*| = p - 1.$
- **Normale lineare Algebra** möglich, v.a. Lösung linearer Gleichungssysteme.
- Über Körper hat **Polynom** vom Grad k maximal k Nullstellen.

  Anm.: Sonst nicht, z.B.

  $x^2 - 1 \bmod 8$: {1, 3, 5, 7}!

5.4 Grundlagen für Systeme mit Faktorisierungsannahme

- **Geheimnisse:**
  (zum Entschlüsseln, Signieren …):
  p, q prim
- **Öffentliche Version:**
  (zum Verschlüsseln, Testen …):
  $n := pq$
- ***p*, *q* groß**, z.Zt. ca. $l := 512$ bit.

  (Theorie: $l \to \infty$.)
- **Nötig:**
  1. Nachrichtenabhängige Dinge mit p, q.
     Für Umkehrung muß n reichen.
  2. Faktorisieren schwer (Annahme)
  3. Primzahlen erzeugen leicht (Schlüssel)

5.4.1 Chinesischer Restsatz

- **Zusammenhang** von Rechnen mod n und Rechnen mod p, q.
- **Nützlich** v.a., wenn Rechnen mod Primzahlen einfacher (weil $\mathbb{Z}_p$ Körper).

$$\left.\begin{matrix} f(x) \bmod p \\ f(x) \bmod q \end{matrix}\right\} \; f(x) \bmod n$$

Chin. Restsatz *(chinese remainder theorem)*

- Für $n = p \cdot q$ mit p, q prim, $p \neq q$:

  $$\mathbb{Z}_n \cong \mathbb{Z}_p \times \mathbb{Z}_q.$$

- Allg.:

  $$\mathbb{Z}_n \cong \mathbb{Z}_{t_1} \times \ldots \times \mathbb{Z}_{t_x},$$

  wenn $t_i = p_i^{e_i}$ Primzahlpotenzen aus Primzerlegung von n.

Konkretere Formeln und Beweis:

1. Für alle $a, b \in \mathbb{Z}$: $(n = p \cdot q;\ p \neq q)$

$$\boldsymbol{a \equiv b \bmod n}$$
$$\Leftrightarrow\ \boldsymbol{a \equiv b \bmod p \wedge a \equiv b \bmod q.}\quad (*)$$

Bew.: Nach Definition von „≡“ äquivalent mit

$$n \mid (a-b) \ \Leftrightarrow\ p \mid (a-b) \ \wedge\ q \mid (a-b).$$

Gilt, weil $p \cdot q$ Primfaktorzerlegung von n.

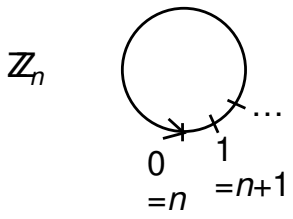
Bsp.: $1027 \equiv 37 \bmod 55$
weil $1027 \equiv 2 \equiv 37 \bmod 5$
und $1027 \equiv 4 \equiv 37 \bmod 11$.

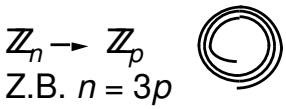
2. D.h., die „kanonische“ Abbildung

$$\kappa: \quad \mathbb{Z}_n \to \mathbb{Z}_p \times \mathbb{Z}_q,$$
$$\overline{a} \pmod n \ \to\ (\overline{a} \pmod p, \overline{a} \pmod q)$$

ist wohldefiniert und injektiv.

Bsp.: $a \equiv 19 \bmod 21 \Rightarrow a \equiv 5 \bmod 7$.
Aber $a \bmod 5$ nicht bekannt.

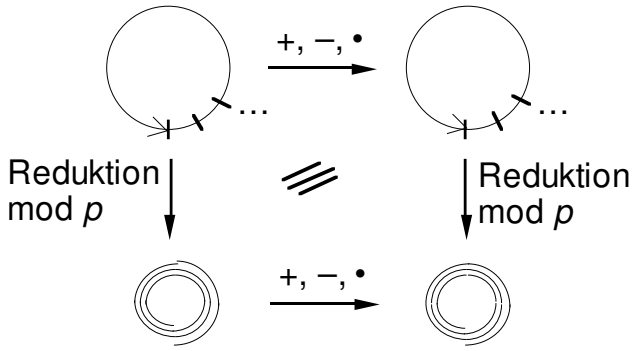




3. **Surjektiv:** Folgt aus injektiv, da

$$|\mathbb{Z}_n| = |\mathbb{Z}_p \times \mathbb{Z}_q| = pq.$$

4. **Homomorphismus:**



$\kappa(\overline{a} + \overline{b} \pmod n)$ (+ in $\mathbb{Z}_n$)

$= \kappa(\overline{a+b} \pmod n)$ (+ in $\mathbb{Z}$)

$= (\overline{a+b} \pmod p, \overline{a+b} \pmod q)$ (+ in $\mathbb{Z}$)

$= (\overline{a} + \overline{b} \pmod p, \overline{a} + \overline{b} \pmod q)$
(+ in $\mathbb{Z}_p, \mathbb{Z}_q$)

$= (\overline{a} \pmod p, \overline{a} \pmod q)$
$+ (\overline{b} \pmod p, \overline{b} \pmod q)$ (+ in …×…)

$= \kappa(\overline{a} \pmod n) + \kappa(\overline{b} \pmod n)$
(+ in …×…).

Analog für –, •.

Chinesischer Restealgorithmus (CRA)

Bisher Surjektivität nicht konstruktiv. Jetzt expliziter Algorithmus für κ^{-1}.

Gegeben: $x_p \bmod p$, $x_q \bmod q$.

Gesucht: $x \bmod n$ mit

$$x \equiv x_p \bmod p,\ x \equiv x_q \bmod q.$$

Bsp.: $x \equiv 5 \bmod 7$, $x \equiv 6 \bmod 11$.

x? (mod 77 oder in $\mathbb{Z}$)

CRA: BEGIN

- Bestimme mit erweitertem Euklidschen Algorithmus Zahlen u, v mit
$$u\,p + v\,q = 1.$$
- $x := u\,p\,x_q + v\,q\,x_p \bmod n$

END.

Zu zeigen: $x \equiv x_p \bmod p \wedge x \equiv x_q \bmod q$:

Schrittweise auswerten:

	mod p	mod q
up	0	1
vq	1	0
$up\,x_q + vq\,x_p$	$0 \cdot x_q + 1 \cdot x_p$ $\equiv x_p$	$1 \cdot x_q + 0 \cdot x_p$ $\equiv x_q$

Obere Zeilen folgen aus $u\,p + v\,q = 1$. □

Anm.: up und vq Art „Basisvektoren“.

Effizienz:

- up und vq einmalig zu p, q berechnen.
- Dann pro CRA nur 2 Multiplikationen (und 1 Addition).

Bsp.: $x \equiv 5 \bmod 7$, $x \equiv 6 \bmod 11$.

Euklid …: $1 = u\bullet 7 + v\bullet 11$ mit $u = -3$, $v = 2$.

$\Rightarrow$ „Basisvektoren“ -21, $+22$

$\Rightarrow x \equiv 5\bullet 22 + 6\bullet(-21) \equiv 5\bullet 1 - 21 \equiv -16 \equiv 61$.

Folge: $\phi(n)$ allgemeiner herleiten:

$$|\mathbb{Z}_n^*| = \phi(n).$$

Dabei $\mathbb{Z}_n^* \cong (\mathbb{Z}_p \times \mathbb{Z}_q)^* = \mathbb{Z}_p^* \times \mathbb{Z}_q^*$

$$\Rightarrow \quad |\mathbb{Z}_n^*| = |\mathbb{Z}_p^*| \bullet |\mathbb{Z}_q^*|$$

$$\Rightarrow \quad \phi(n) = \phi(p) \bullet \phi(q) = (p-1)(q-1).$$

Analog für $n = p_1^{e_1} \bullet \ldots \bullet p_x^{e_x}$:

$$\phi(n) = \phi(p_1^{e_1}) \bullet \ldots \bullet \phi(p_x^{e_x})$$

$$= (p_1-1)p_1^{e_1-1} \bullet \ldots \bullet (p_x-1)p_x^{e_x-1}.$$

5.4.2 Faktorisierung, Annahme

≈ Es ist schwer, große Zahlen n in ihre Primfaktoren zu zerlegen.

Genauer:

- Kleine Primfaktoren 2, 3, 5 finden leicht.
- $\Rightarrow$ Meist nur betrachtet

  $$n = p\bullet q$$

  für zwei „große“ Primzahlen p, q.

A. Standardannahme

Genau wie: Multiplikation Einwegfunktion auf diesem Definitionsbereich:

$\forall$ prob. poly. Algo. A (Angreifer)

$P\big(n = p'\bullet q' ::$ (W'keit v. Angreifer-ziel wenn…)

$p, q \in_R$ Menge der l-bit-Primzahlen; (Brave wählen)

$n := p\bullet q$; (")

$p', q' \leftarrow A(l, n)\big)$ (Angreifer rechnet)

$\leq 1/\text{poly}(l)$ (Vernachlässigbar)

B. Varianten der Annahme

- p, q nicht genau gleich groß.
- Zusatzforderungen an p, q, z.B.
  - $p \equiv q \equiv 3 \bmod 4$ (für manche Rechnungen).
  - $p-1$, $q-1$, $p+1$ und $q+1$ haben je mindestens einen großen Primfaktoren p', q'.

    (Sonst spezielle Faktorisierungsalgorithmen anwendbar, die asymptotisch keine Rolle spielen, aber in Praxis noch.)
- Gewisse Abweichung von Gleichverteilung erlauben. (In Praxis sind ganz gleichverteilte Primzahlen schwer zu erzeugen.)

C. Stand der Faktorisierung

Praxis:

- **Größe:** Zahlen dieser Form bis etwa 130 Dezimalstellen ≈ 430 bits.
- Z.Zt. mit „**quadratischem Sieb**“ [Pome_85]. Laufzeit

  $$\approx L(n) := e^{\sqrt{\ln(n)\,\ln(\ln(n))}}.$$

  Beachte: Eingabelänge = $\log_2(n) \approx \ln(n)$, d.h. „subexponentiell“ !
- Demnächst wohl überlegen: Relativ neuer Algorithmen mit 3-ter Wurzel im Exponent: „**number field sieve**“ [LeLe_93].
- „Algorithmus“ steht jeweils für Klasse mit vielen Varianten $\Rightarrow$ Forschungsfeld
- **Hochparallele** Varianten auf unabhängigen Rechner ("Factoring by email")

5.4.3 Primzahlerzeugung

A. Rahmenalgorithmus:

WHILE „noch keine Primzahl gefunden“
DO
  wähle Zufallszahl p mit l bits
    (ggf. z.B. mit $p \equiv 3 \bmod 4$);
  teste, ob p prim
ENDWHILE.

B. Grobe Laufzeit:

Wieviel Versuche im Mittel?

- Wenn $1/t$ der l-bit-Zahlen prim, dann t.
- **Primzahlsatz:** Anzahl $\pi(x)$ der Primzahlen bis zur Zahl x:

$$\frac{\pi(x)}{x} \approx \frac{1}{\ln(x)}.$$

  Also unter Zahlen bis l Bits: Im Mittel jede $(l \cdot \ln(2))$-te prim

  $\Rightarrow$ im Mittel **ca. $l \cdot \ln(2)$ Versuche**.

- **(Dirichletscher Primzahlsatz:** Im Mittel etwa gleichviel $\equiv 3 \bmod 4$ wie $\equiv 1 \bmod 4$. (Analog modulo jeder Zahl).**)**

C. Primzahltest für Rahmenalgorithmus:

Viel schneller, als man p faktorisieren könnte!

- **Rabin-Miller-Test** (probabilistisch):
  Hier nur Fall $p \equiv 3 \bmod 4$ (einfacher):
  - Es gilt (ohne Bew.; 1. Hälfte in Kap. 5.5.1):
    - p prim $\Rightarrow \forall a \in \mathbb{Z}_p^*: a^{\frac{p-1}{2}} \equiv \pm 1 \bmod p$.
    - p nicht prim: dasselbe höchstens für 1/4 der a modulo p.

  $\Rightarrow$ Prüfe diese Formel für σ zufällige a.
  - Gilt einmal nicht $\Rightarrow$ p nicht prim.
  - Andernfalls p „vermutlich prim“.

    (**A priori**: Geht nur mit Wahrscheinlichkeit $\leq 4^{-\sigma}$ schief.)

  Diese Fehlerwahrscheinlichkeit stört bzgl. Faktorisierungsannahme nicht.
- In Praxis **Trial Division** vor Rabin-Miller:
  - Tabelle der ersten Primzahlen anlegen, z.B. alle von höchstens 16 bit.
  - Durch alle diese probeweise dividieren.

5.5 Grundlagen für Systeme mit Diskreter-Log-Annahme

- **Grob:** Schwer, in Ringen $\mathbb{Z}_p$ Gleichungen

$$g^x = h$$

  nach x zu lösen. (Auch in $\mathbb{Z}_n$ allgemein und einigen anderen Gruppen.)

- x heißt **diskreter Logarithmus** von h zur Basis g modulo p:

$$x = \log_g(h) \text{ in } \mathbb{Z}_p.$$

Bsp.: In $\mathbb{Z}_7$, $g = 5$, $h = 4$: $x = 2$, denn

$$5^2 \equiv 4 \bmod 7.$$

Also:

- **Geheimnisse:**
  (zum Entschlüsseln, Signieren …):

$$x$$

- **Öffentliche Version:**
  (zum Verschlüsseln, Testen …):

$$p, g, h = g^x.$$

- **Nötig:**
  1. Nachrichtenabhängige Dinge mit x:
     Für Umkehrung muß h reichen.
     $\rightarrow$ **Rechnen mit Zahlen g^x mod p.**
     (Auch manchmal bei Fakt.annahme!)
  2. Diskrete Logarithmen ziehen schwer (Annahme).
  3. Details der Schlüsselerzeugung.

5.5.1 Zyklische Gruppen

A. Gruppenordnungen

G Gruppe.

- **Elementanzahl** $|G|$ heißt Ordnung von G.
- **Satz von Lagrange:** Wenn H Untergruppe von G, dann ist $|H|$ Teiler von $|G|$.

  Als Formel:

  $$H \leq G \;\Rightarrow\; |H| \,|\, |G|.$$

  (*Beweisskizze*: G zerfällt in „Nebenklassen“, alle genauso groß wie H. Siehe jedes Algebrabuch.)
- **Folge:** Falls $|G|$ prim, hat G nur triviale Untergruppen G und $\{1\}$.

  (G multiplikativ geschrieben.)

B. Elementordnungen, zyklische Gruppen

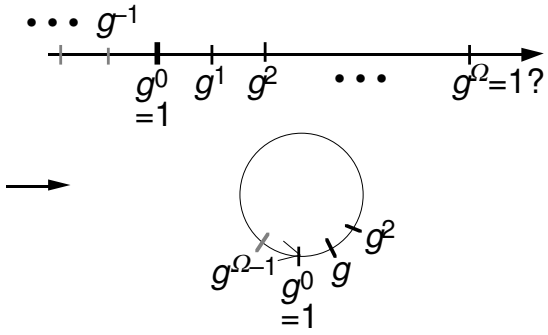
Sei $g \in G$.

Kleinstes $\Omega \in \mathbb{N} \cup \{\infty\}$ mit $g^\Omega = 1$ heißt Ordnung von g, geschrieben ord(g).

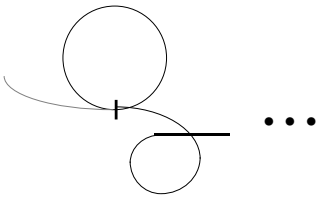
Sätzchen 1: Für alle $x, y \in \mathbb{Z}_p$:

$$\boldsymbol{g^x = g^y \Leftrightarrow x \equiv y \bmod \mathrm{ord}(g).}$$

D.h.



Nicht z.B.



Beweis: Sei $\Omega = \mathrm{ord}(g)$.

„$\Leftarrow$“ : Sei $x \equiv y \bmod \Omega$.

Dann gibt's k mit $y = x + k \bullet \Omega$.

$$\Rightarrow\; g^y = g^{x+k\bullet\Omega} = g^x \bullet (g^\Omega)^k = g^x \bullet 1^k = g^x.$$

„$\Rightarrow$“: Sei $g^x = g^y$, also $g^{x-y} = 1$.

Sei $x - y = q \bullet \Omega + r$ mit $0 \leq r < \Omega$ (Division mit Rest).

Dann $\quad 1 = g^{x-y} = g^{q\bullet\Omega+r} = g^r$

$\Rightarrow r = 0$, sonst ↯ zu „Ω minimal“

$\Rightarrow x \equiv y \bmod \Omega$.

Aus Bildern: Potenzen von g sehen aus wie $(\mathbb{Z}_\Omega, +)$

$\Rightarrow$ Werden zu $\mathbb{Z}_\Omega$ isomorphe Gruppe sein.

Sätzchen 2:

a) Potenzen von g bilden **Untergruppe** von G.

   Sie wird $\langle g \rangle$ geschrieben.

b) Es gilt $|\langle g \rangle| = \mathrm{ord}(g)$.

Beweis: a) $g^x \bullet g^y = g^{x+y}$; $\quad (g^x)^{-1} = g^{-x}$.

b) Aus Sätzchen 1.

Def.:

- g heißt **Generator** von G, falls $\langle g \rangle = G$.
- Eine Gruppe mit Generator heißt **zyklisch**. (Kreis!)

C. Lemmas über zyklische Gruppen

Lemma 1: Sei G zyklisch und g Generator:

a) Für $\Omega := |G|$:

$$\boldsymbol{G \cong (\mathbb{Z}_\Omega, +)}.$$

b) Ein Isomorphismus von $\mathbb{Z}_\Omega$ nach G ist

$$exp_g(\overline{x}) := g^x.$$

c) Invers: diskreter Logarithmus log_g.

Beweis:

- Wohldefiniert u. bijektiv siehe Sätzchen 1.
- Homomorphismus:

$$g^{x+y} = g^x \bullet g^y, \quad g^{-x} = (g^x)^{-1}.$$

Lemma 2: Für alle $g \in G$:

$$\boldsymbol{ord(g) \mid |G|}.$$

Beweis: ord(g) = |<g>|, Satz v. Lagrange.

Bsp.: $|G| = 10$: Immer $g^{10} = 1$, evtl. schon

$$g = 1 \vee g^2 = 1 \vee g^5 = 1.$$

Lemma 3: Kleiner Fermatscher Satz

Für alle $g \in G$:

$$\boldsymbol{g^{|G|} = 1}.$$

Bew.: Sei ord(g) = Ω und $|G| = n$.

Lemma 2.: $\Omega \mid n$, d.h. $n = k \bullet \Omega$,

$$\Rightarrow \quad g^n = (g^\Omega)^k = 1^k = 1.$$

Spezialfall 1: Für alle $g \in G = \mathbb{Z}_p^*$ (mit p prim):

$$\boldsymbol{g^{p-1} = 1}.$$

Spezialfall 2: Für alle $g \in G = \mathbb{Z}_n^*$ (allgemein):

$$\boldsymbol{g^{\phi(n)} = 1}.$$

Folge: Hälfte von Beweis **Rabin-Miller-Test**: In $\mathbb{Z}_p^*$:

$$g^{\frac{p-1}{2}} \equiv \pm 1,$$

denn

$$\left(g^{\frac{p-1}{2}}\right)^2 \equiv 1,$$

und 1 hat in *Körper* nur 2 Wurzeln ± 1.

Lemma 4: Für $|G| = p$ prim:
Alle $g \in G$ außer 1 sind **Generatoren**.

Beweis: Ordnung von <g> kann nur 1 oder p sein (Lagrange).

Lemma 5: Untergruppen: Sei G = <g> und $|G| = n$:

a) Zu jedem $\boldsymbol{t \mid n}$ gibt's genau eine Untergruppe $\boldsymbol{G_t}$ der Ordnung t.

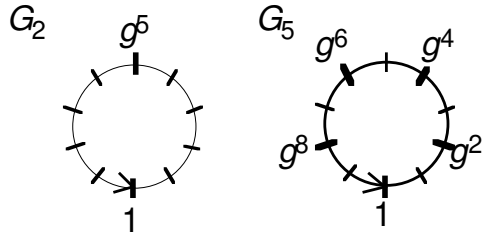
b) G_t auch zyklisch, generiert von $g^{n/t}$.

c) G_t besteht genau aus den g^* mit $g^{*\,t} = 1$.

d) Andere Generatoren: Genau die $h = g^x$ mit ggT(x, n) = 1 sind auch Generatoren von G.

Beweisskizze: (Genau wie im isomorphen ($\mathbb{Z}_n$, +), siehe jedes Algebrabuch.)

Bsp.: Untergruppen, wenn $|G| = 10$:



Aber z.B. g^3 erzeugt wieder ganz G.

Satz vom primitiven Element (aus Algebra, hier ohne Beweis):

Alle Gruppen $\boldsymbol{\mathbb{Z}_p^*}$ sind zyklisch !

Generatoren davon heißen auch Primitivwurzeln.

- Sogar die $\mathbb{Z}_{p^r}^*$ für p ungerade.
- Aber nicht die $\mathbb{Z}_n^*$!

Bsp.

$$\mathbb{Z}_{15}^* \cong \mathbb{Z}_3^* \times \mathbb{Z}_5^*,$$

$$\mathbb{Z}_3^* \cong (\mathbb{Z}_2, +), \qquad (\{1, 2\})$$

$$\mathbb{Z}_5^* \cong (\mathbb{Z}_4, +). \qquad (\{1, 2, 2^2 = 4, 2^3 = 3\})$$

$\Rightarrow$ Alle Elemente haben Ordnung ≤ 4, denn $(g_3, g_5)^4 = ({g_3}^4, {g_5}^4) = (1, 1)$. Also nicht zyklisch von Ordnung 8.

Lemma 6: Anzahl der Generatoren:

$\mathbb{Z}_p^*$ hat $\phi(p-1)$ Generatoren.

Bew.: $|\mathbb{Z}_p^*| = p - 1$. Dann Lemma 5d: Die Generatoren sind die g^x mit ggT(x, $p-1$) = 1.

5.5.2 Diskreter Logarithmus, Annahmen

A. „Standard“

$\forall$ prob. poly. Algo. A (Angreifer)

$P(g^x = h \bmod p ::$ (W'keit v. Angreifer-ziel wenn…)

$p \in_R$ Menge der l-bit-Primzahlen; (Brave wählen …

$g \in_R$ Generatoren von $\mathbb{Z}_p^*$;

$h \in_R \mathbb{Z}_p^*$;

$x \leftarrow A(l, p, g, h))$ (Angr. rechnet)

$\leq 1/\text{poly}(l)$ (Vernachlässigbar)

B. In Untergruppe

Ziel

- Oft Gruppen **primer Ordnung** praktisch.
- $\mathbb{Z}_p^*$ hat aber $p-1$ Elemente: Nie prim.
- Nehme Untergruppe G_q für großen Primteiler $q \mid (p-1)$.

  (G_q existiert und eindeutig nach Lemma 5a.)

Details

- 2 Parameter $l^* < l$ für Längen von q, p.
- Angreiferaufwand auch in l^* höchstens exponentiell (x in G_q raten und prüfen).

  (Sofern l polynomial in l^*.)
- Konkrete Funktion *len_p* mit $l = len_p(l^*)$ vorgeben.

  ($\Rightarrow$ in Annahme wieder nur 1 Sicherheitspar.)
- Konkret aber l wie vorn, l^* kleiner, s. Kap. D.

Annahme dann

$\forall$ prob. poly. Algo. A (Angreifer)

$P(g^x = h \bmod p ::$ (W'keit v. Angreifer-ziel wenn…)

$l := len_p(l^*)$ (Brave wählen …

$q \in_R$ Menge der l^*-bit-Primzahlen;

$p \in_R$ Menge der l-bit-Primzahlen mit $q \mid (p-1)$;

$g \in_R$ Generatoren von $G_q < \mathbb{Z}_p^*$;

$h \in_R G_q$;

$x \leftarrow A(l^*, p, q, g, h))$

(Angr. rechnet)

$\leq 1/\text{poly}(l^*)$ (Vernachlässigbar)

C. Sonstige Varianten

- **Andere Gruppen(-familien)**

  V.a.:
  - Multiplikative Gruppen anderer Körper, meist GF(2^n).
  - „Elliptische Kurven“.
- **Kleine Änderungen an Verteilungen**
  - Nicht ganz gleichverteilte Wahl von q, p wegen Generierung
  - $h \neq 1$ verlangt. Annahme äquivalent zur obigen.

(• Zertifiziert (engl. „certified discrete log“)

Standardannahme, aber andere Partei muß Korrektheit von p, g prüfen. Letzteres geht nur mit Zusatzinfo vom Generierer (Primfaktoren von p) $\Rightarrow$ entsprechende Annahme.**)**

D. Stand des Berechnens diskreter Logarithmen

- **In $\mathbb{Z}_p^*$:** Erstaunlich parallel zu Faktorisierung (aber nicht als äquivalent bewiesen).

- **Größe:** Noch nicht so nah an 512 Bits, aber viel weniger Aufwand reingesteckt.
- **Beste Algorithmen:** Bis vor kurzem / jetzt Laufzeit (vgl. [LaOd_91]):

$$\approx L(n) := e^{\sqrt{\ln(n)\ln(\ln(n))}}.$$

  Dann Algorithmen mit 3-ter Wurzel im Exponent [Gord1_93], „number field sieve“.
- Wieder sollte p–1 großen Primfaktor haben.

- **In Untergruppen:** Bisher kein in $l^* = |q|_2$ subexponentieller Algorithmus bekannt, nur
  - mit Laufzeit $\approx \sqrt{q}$
  - und der in p subexponentielle

  $\Rightarrow$ man traut sich mit 160-bit q in $\mathbb{Z}_p^*$ für 512-bit p.

  (**Aber** noch nicht so gut untersucht $\Rightarrow$ vielleicht eines Tages?)
- **In anderen Gruppen:**
  - GF(2^n): Schon mit 3-ter Wurzel im Exponenten.
  - Elliptische Kurven: Kein subexponentieller bekannt, jedenfalls nur spezielle Sorten. (**Aber** noch nicht so gut untersucht … .)

5.5.3 Schlüsselerzeugung

A. Standard

Problem: Suchen des Generators g von $\mathbb{Z}_p^*$

- Def. war $\langle g \rangle = \mathbb{Z}_p^*$ bzw. $\text{ord}(g) = p - 1$.
- Direkter Test wäre exponentiell.

Ansatz:

- Verwende $\text{ord}(g) \mid |G|$ (Lemma 2).

  $\Rightarrow$ Wenn

$$g^t \neq 1$$

  für alle echten Teiler t von $|G| = p$–1, dann g Generator.
- Hierzu Primfaktoren von (p–1) nötig, sei etwa

$$p\text{–}1 = q_1^{e_1} \bullet \ldots \bullet q_x^{e_x}.$$

  Primfaktoren **zuerst** generieren, da p–1 groß!
- Hinreichend: maximale Teiler t_i betrachten:

$$t_i = (p\text{–}1)/q_i.$$

  Denn $g^t = 1 \wedge t | t_i \Rightarrow g^{t_i} = 1$.

Algorithmus also:

Typische Version mit nur einem großen q_i:

- Generiere Primzahl $\boldsymbol{q_1}$ einer Länge $l - c$, z.B. $c = 10$.
- Suche unter Zahlen

$$\boldsymbol{p} = tq_1 + 1$$

  mit $|t|_2 = c$ nach Primzahl.
- Faktorisiere t. (Einfach, da klein.)

  Seien $q_2, \ldots, q_x$ die Primteiler.
- Wähle $\boldsymbol{g} \in_R \mathbb{Z}_p^*$. Teste, ob

$$\boldsymbol{g^{(p-1)/q_i} \neq 1}$$

  für alle i. Falls nicht, wiederhole Wahl von g.

Laufzeit der probabilistischen Suche nach g:

- Anzahl Generatoren $\phi(p-1)$. (Lemma 6).
- Hier $\phi(p-1) = (q_1-1)\phi(t)$. (Chin. Restsatz)
- Also Anteil

$$\frac{\phi(p-1)}{p-1} = \frac{(q_1-1)\phi(t)}{q_1 t} \approx \frac{\phi(t)}{t}$$

- Also im Mittel $t/\phi(t)$ Versuche.

Hier t klein, kein Problem.

(Allgemeiner Satz $\phi(n)/n \geq 1/(6 \ln(\ln(n)))$.)

Anmerkung:

Art der Suche nach p ändert angenommene Verteilung etwas.

B. In Untergruppe

(Im Prinzip einfacher wegen primer Ordnung)

Ansatz für Generator diesmal:

- Jedes $g \in G_q$ außer 1 ist Generator (Lemma 4).

  $\Rightarrow$ Es reicht Zufallszahl in G_q.

Algorithmus:

- ***q***: Generiere q **zuerst**.
  (Denn $p-1$ zu groß zum Faktorisieren.)
- ***p***: Primzahltests für Zahlen $p := t\,q + 1$;
  t zufällig mit $|t|_2 = l - l^*$.
- **Generator:** Wähle $g^* \in_R \mathbb{Z}_p^*$;

$$g := g^{*(p-1)/q}.$$

Korrektheit:

- $g \in G_q$ folgt aus Lemma 5c:

$$g^q = g^{*((p-1)/q)\cdot q} = g^{*p-1} = 1$$

  (mit Lemma 3.)

- **Warum zufällig?**

  Beh.: Zu jedem $g \in G_q$ gibt's gleich viele $g^* \in \mathbb{Z}_p^*$ mit $g^{*(p-1)/q}$.

  - Maximal $(p-1)/q$.
    (Polynomgleichung in Körper).
  - Potenzierung bildet die $p-1$ Elemente von $\mathbb{Z}_p^*$ auf die q von G_q ab $\Rightarrow$ Im Mittel hat jedes Bild $(p-1)/q$ Urbilder.
  - $\Rightarrow$ jedes hat genau so viele. □

  Anderer Beweis: Potenzierung ist Homomorphismus; immer alle Urbilder gleich groß.

5.6 Weiterführende Literatur

Zitiert:

Gord1_93 Daniel M. Gordon: Discrete logarithms in GF(p) using the number field sieve; SIAM Journal on Discrete Mathematics 6/1 (1993) 124-138.

Knut_81 Donald E. Knuth: The Art of Computer Programming, Vol. 2: Seminumerical Algorithms (2nd ed.); Addison-Wesley, Reading 1981.

LaOd_91 Brian A. LaMacchia, Andrew M. Odlyzko: Computation of Discrete Logarithms in Prime Fields; Designs, Codes and Cryptography 1/1 (1991) 47-62.

LeLe_93 A. K. Lenstra, H. W. Lenstra: The Development of the number field sieve; Lecture Notes in Mathematics 1554, Springer Verlag, Berlin 1993.

Pome_85 Carl Pomerance: The Quadratic Sieve Factoring Algorithm; Eurocrypt '84, LNCS 209, Springer-Verlag, Berlin 1985, 169-182.

Andere Einführungen:

- Buch v. Menezes et al. (vgl. Kap 0.C), Kap. 2.4, 2.5, 4.1, Teile von 4.2, 4.4, 4.6
  (Viel mehr Details, aber nicht viel math. Gründe).
- (• Schneier: zu wenig)
- Anfang des Koblitz-Buchs (vgl. Kap 0.C).
- Echte Einführung in Algebra oder elementare Zahlentheorie.

6 Konkrete Systeme

6.1 Symmetrische Verschlüsselung

6.1.1 Informationstheoretische Geheimhaltung; One-time-Pad

Überblick:

- Informationstheor. sym. Verschlüsselung möglich (genaue Def. unten).
- Aber Schlüssellänge ≈ Nachrichtenlänge.
- Einfaches optimal effizientes System bekannt: Vernam-Chiffre = One-time pad.

Anm. zur Geschichte:

- Vernam-Chiffre 1926.
- Allg. Theorie Shannon 1949 (mit Informationstheorie) → Echte Def., Optimalität.

⇒ Lange die einzige theor. Kryptographie.

(• 1974 sym. Authentikation,

- 1976 asym. Verfahren.)

A. Definition

Angriffs- und Erfolgsart:

- **Passiv** (nur eine Nachricht betrachtet, Erweiterung unten.)
- Aber **partielle Information** betrachtet.

  Grob: Angreiferwissen über Klartext durch Schlüsseltext unverändert.

Generelle Bezeichnung:

- Für diskrete W'verteilung D sei

  $$[D] = \{x \mid \mathrm{P}(x) > 0\}.$$

  („Trägermenge" = Menge möglicher Werte.)
- Insbesondere für Zufallsvariablen und prob. Algorithmen, z.B. ist

  $$[gen(I, \bullet)]$$

  Menge mögl. Schlüssel bei Parameter I.

Spezielle Bezeichnungen:

- $\boldsymbol{M_I}$ sei Klartextraum zu I,

  $\boldsymbol{K_I} = [gen(I, \bullet)]$ Schlüsselraum,

  $\boldsymbol{C_I} = ver(K_I, M_I)$ Schlüsseltextraum.
- Sei $\boldsymbol{D}$ beliebige W'keitsverteilung auf M_I.

  (Modelliert beliebiges Vorwissen des Angreifers, Bsp. „deutsche Email").
- Zusammen mit zufälliger Wahl von z für gen ergibt sich W'raum.

  Sei $\mathbf{P_D}$ W'keit in diesem W'raum.
- Darin sind Schlüssel und Schlüsseltext Zufallsvariablen:

  $$k = gen(z)$$
  $$c = ver(gen(z), m)).$$

Definition:

$\forall$W'verteilungen D auf M_I (Vorwissen)

$\forall c \in C_I$ (mögl. Schlüsseltext)

$\forall m \in M_I$ (Nachrichten)

$$\mathrm{P}_D(m|c) = \mathrm{P}_D(m). \quad (1)$$

Zentrale Geheimhaltungsformel:

A-posteriori-W'keit von m, gegeben Angreiferbeobachtung c	=	A-priori-W'keit

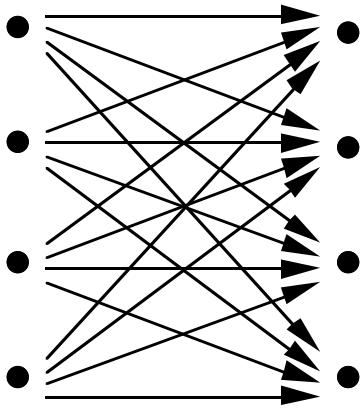
Auch bei anderen Geheimnissen und Beobachtungen.

B. Hinreichendes Kriterium

- Alle Schlüssel in K_l gleichwahrscheinlich.
- Und $\forall c \in C_l \ \forall m \in M_l$

  $\exists! k \in K_l$: (genau ein)

  $$ver(k, m) = c. \quad (2)$$



Anm.: Geht auch mit jeder anderen festen Zahl von Schlüsseln pro Paar (m, c).

Beweis:

- **Bayes'sche Formel** (Ausrechnen von A-posteriori-W'keiten aus A-priori-W'keiten):

$$\begin{aligned} P_D(m|c) &= \frac{P_D(m)P_D(c|m)}{P_D(c)} \\ &= \frac{P_D(m)P_D(c|m)}{\sum_{m^*} P_D(m^*)P_D(c|m^*)}. \end{aligned}$$

- **Kriterium** impliziert $\forall m, c$

$$P_D(c|m) = 1/|K_l|.$$

- **Zusammen**

$$\begin{aligned} P_D(m|c) &= \frac{P_D(m)/|K_l|}{\sum_{m^*} P_D(m^*)/|K_l|}. \\ &= \frac{P_D(m)}{\sum_{m^*} P_D(m^*)}. \\ &= P_D(m). \end{aligned}$$

□

C. Vernam-Chiffre = One-time-Pad

- k ist Zufallszahl der Länge l

  (d.h. *gen* tut weiter nichts).
- $M_l = \{0, 1\}^l$.
- $ver(k, m) = m \oplus k$.
- $ent(k, c) = c \oplus k$.

Bsp.:

$$\begin{array}{rcl} m & = & 0\,0 \ \ 0\,1 \ \ 0\,0 \ \ 1\,0 \\ \oplus \ k & = & 1\,0 \ \ 1\,1 \ \ 0\,1 \ \ 0\,0 \\ \hline c & = & 1\,0 \ \ 1\,0 \ \ 0\,1 \ \ 1\,0 \end{array}$$

Beweis:

- **Effektiv:** Klar: $(m \oplus k) \oplus k = m$.
- **Sicher:** Kriterium erfüllt: $\forall m \, \forall c \ \exists! k$: $ver(k, m) = c$, nämlich

$$k = m \oplus c.$$

Effizienz: Sehr schnell, aber Schlüssel so lang wie Nachricht. (Aber z.B. Diskette reicht für viele Emails.)

D. Mehrere Nachrichten

- Insgesamt langes k austauschen.
- Kein Stück von k mehrfach verwenden! („**One-time-Pad**")
- Jetzt System mit Gedächtnis! (Andere Struktur als vorn und in Def.)
- Aktive Angriffe dann Problem: Jede Nachricht mit ihrem Schlüsselstück unabhängig betrachtbar.

E. Vernam-Chiffre allgemeiner

Gruppe G (oben $\{0, 1\}^l$)

- $k \in G$ zufällig.
- $ver(k, m) = m + k$.
- $ent(k, c) = c - k$.

Lohnt hier nicht, aber innerhalb von Protokollen.

F. Optimalität

Beh.: Für informationstheoretische Sicherheit müssen Schlüssel so lang sein. Genauer:

$$|K_l| \geq |M_l|.$$

Bew.: Betrachte festes c.

- „Kann jeden Klartext enthalten“ $\Rightarrow \forall m\ \exists k$: $ent(k, c) = m$.

  (≈ Umkehrung von Kriterium in B.)

- Obige k alle verschieden, da Entschlüsselung eindeutig.

$$\Rightarrow |K_l| \geq |M_l|. \qquad \square$$

Folge: Falls Klartext geschickt codiert, folgt:

Schlüssel mindestens so lang wie Klartext.

G. Literatur

Das Original (enthält mehr):

Sha1_49 Claude E. Shannon: Communication Theory of Secrecy Systems; The Bell System Technical Journal 28/4 (1949) 656-715.

6.1.2 DES

- „Data Encryption Standard“; seit ca. 1977.
- Erster Kryptostandard, überhaupt mit erste nichtgeheime Bemühungen um sym. Kryptographie.
- Nicht patentiert ⇒ frei benutzbar
- Schlüssel jetzt zu kurz ⇒ Tripel-DES

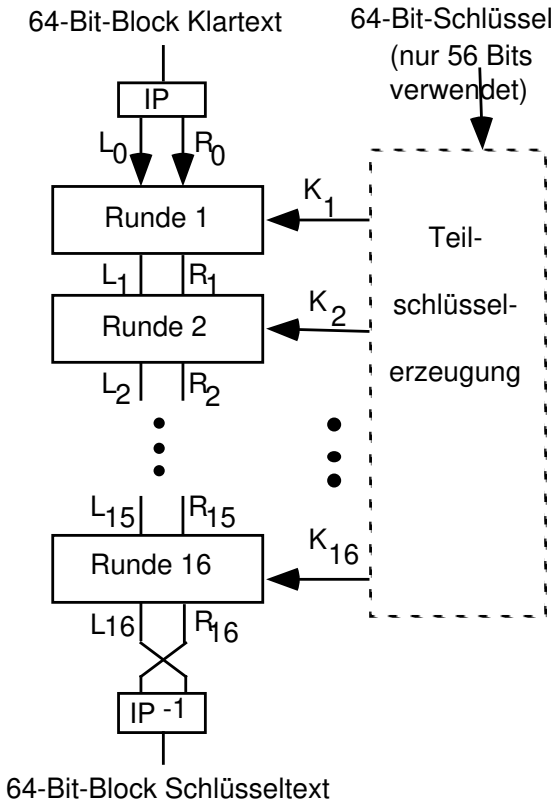
Grundprinzipien

- Blockchiffre, d.h. Funktion auf kurzen Blöcken. (⇒ Betriebsarten, siehe 6.1.3)
- Iteration ähnlicher Runden
- Feistelprinzip für Entschlüsselung.
- Substitutions-Permutations-Prinzip

  (Konfusion — Diffusion)

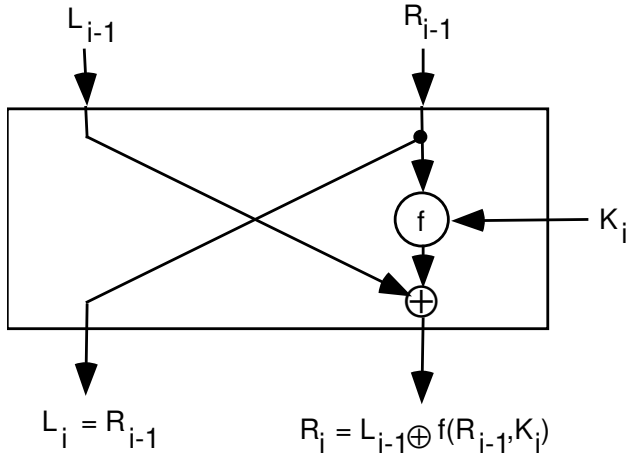
A. Konstruktion

Gesamtstruktur



- IP, IP^{-1} Ein- und Ausgangspermutation, nicht sehr wichtig.

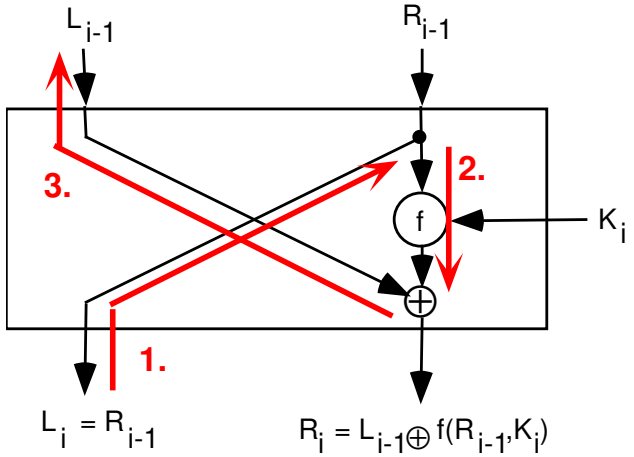
Eine Runde



- L_i, R_i 32-bit-Blöcke
- Exor bitweise
- f noch zu bestimmen
- Name dieser Rahmenkonstruktion: Feistelprinzip

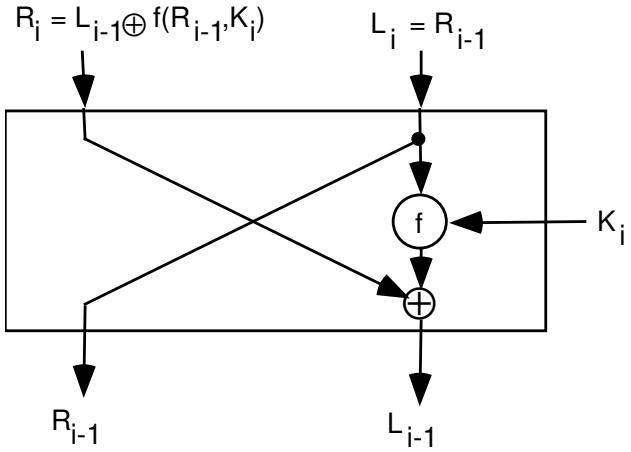
Entschlüsselung von Feistel-Chiffren

Trick am Feistel-Prinzip: Kein f^{-1} nötig!



1. $R_{i-1} := L_i$
2. Berechne *Zwischen* := $f(R_{i-1}, K_i)$
3. $L_{i-1} := R_i \oplus$ *Zwischen*.

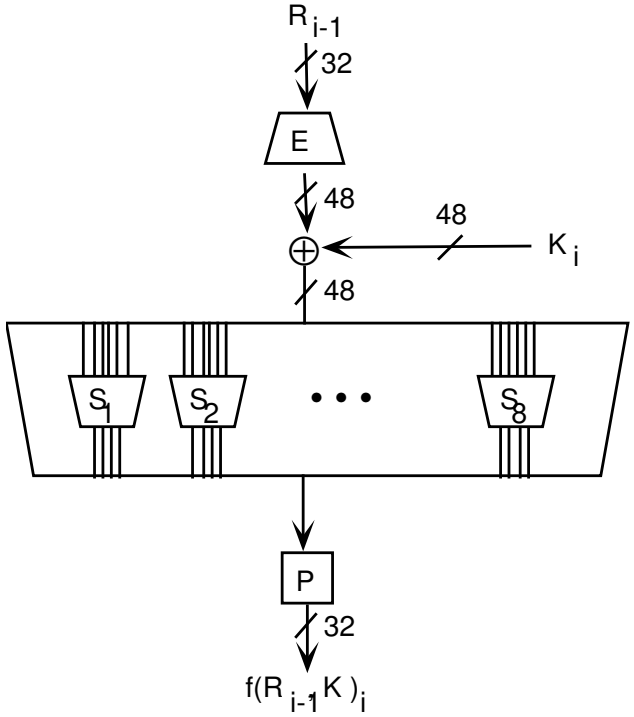
Geht sogar mit selber Hardware!



Gesamtentschlüsselung:

- Teilschlüssel in umgekehrter Reihenfolge
- IP und IP^{-1} deshalb invers gewählt
- Bei Entschlüsselung R_i links, deshalb in Grundaufbau schon Schlußvertauschung.

Funktion *f* (Hauptverschlüsselungsanteil)



- *E* „Expansionspermutation“: Vertauschung der Bits, manche kommen doppelt raus.
- *P* Permutation.
- $S_1, \ldots, S_8$ Substitutionsboxen: Austabellierte Funktionen.

Substitutions-Permutations-Prinzip

- *E*, *P* „Diffusion“ (Verteilung von Bits)
- S_i „Konfusion“ (chaotische Funktionen)
  - Schafft insbesondere Nichtlinearität (Permutationen und ⊕ sind linear)
- Insgesamt hofft man, daß nach 16 Runden alles mit allem gemischt ist und eine „strukturlose“ 64-auf-64-Bit Funktion rauskommt.

Teilschlüsselerzeugung

Zur Auswahl von 48 aus 56 Bits für jede Runde:

Folge von Shifts und Auswahlen (d.h. jedes Bit eines K_i ist ein einzelnes Bit aus *K*)

B. Effizienz

- Für Spezialhardware entworfen.
- In Software am besten mit großen Tabellen (siehe z.B. [PfAß_93]).

1993 grob 1 Mbit/s in Software, 20 Mbit/s in Hardware.

C. Sicherheit

Keine Beweise oder Reduktionen (außer [LuRa_88] u.ä. für Spezialisten).

- **Hauptangriff: Vollständige Suche**
  - Nach Schlüsseltext-Klartext-Angriff: Vollständige Suche nach Schlüssel.
  - Mit 10^6\$ max. 7 Stunden pro Schlüssel [Wien_93]

  ⇒ **Schlüssel zu kurz.**
- **Mathematische oder einfache statistische Angriffe**

  Keine erfolgreichen bekannt. (Waren Entwurfskriterium.)
  - Nötig z.B. Nichtlinearität:

    Nicht: $f(R_{i-1}, K_i)$ = Matrix • (R_{i-1}, K_i).
  - Keine statistische Abhängigkeit bestimmter Ausgabebits von bestimmten Schlüsselbits bei bekanntem Klartext.

- **Differentielle oder lineare Kryptoanalyse**
  - Komplexe statistische Angriffe mit vielen bekannten Klartext-Schlüsseltext-Paaren [BiSh_93, Mats_94].
  - Grundidee: Betrachte Paare von Klartexten mit bestimmter Beziehung (z.B. nur 1 Bit verschieden.)
  - Man hörte z.T. „DES gebrochen“. Aber nur: „Erstmals effizienterer Angriff als vollständige Suche“.
  - Zudem z.Zt. ca. 2^{43} Klartext-Schlüsseltext-Paare, was unrealistisch ist.

  Aber Entwicklung weiterverfolgen.

D. Gegenmaßnahmen gegen vollständige Suche

- **Direkte Verallgemeinerungen**:
  - Mehr verschiedene Schlüsselbits (Teilschlüssel haben ja 768)
  - Substitutionsboxen als Teil des Schlüssels
  - Permutationen als Teil des Schlüssels
  - Mehr Runden

  Im Zusammenhang mit differentieller Kryptoanalyse v.a. noch mehr Runden empfohlen.
- **Tripel-DES:** Nachrichten 3 mal verschlüsseln (3 unabhängige Schlüssel).

  Warum gleich 3? Sog. meet-in-the-middle Angriffe (vgl. Menezes-Buch).

Auch vorgeschlagen:

- $ver(k_1, \mathbf{ent}(k_2, ver(k_3(m))))$

  wegen Kompatibilität mit 1-fach-DES (setze ggf. $k_3 = k_2$).
- $ver(k_1, ent(k_2, ver(\mathbf{k_1}(m))))$

  Sicherheit mittel, nur zu Schlüsselkürzung (scheint mir nicht lohnend).

- Andere, ähnlich entworfene Systeme, z.B. **IDEA** [LaMa_91].
  - 128-bit Schlüssel.
  - In PGP 2 verwendet.
  - Eher für Software: Statt *S* und *P* Multiplikationen und Additionen.

E. Literatur

DES_77 Specification for the Data Encryption Standard; Federal Information Processing Standards Publication 46 (FIPS PUB 46), January 15, 1977.

- *Zumindest bei Implementierung eines Standards am besten immer Standard-Dokument!*
- *Ausführlich in fast allen Bücher n, z.B. wieder Schneier oder Menezes et al.*

Zitiert

BiSh_93 E. Biham, A. Shamir: Differential Cryptanalysis of the Data Encryption Standard; Springer-Verlag, New York 1993.

LaMa_91 X. Lai, J. L. Massey: A Proposal for a New Block Encryption Standard; Eurocrypt '90, LNCS 473, Springer-Verlag, Berlin 1991, 389-404.

LuRa_88 M. Luby, Ch. Rackoff: How to construct pseudo-random permutations from pseudorandom functions; SIAM J. on Computing 17/2 (1988) 373-386.

Mats_94 M. Matsui: Linear Cryptanalysis Method for DES Cipher; Eurocrypt '93, LNCS 765, Springer-Verlag, Berlin 1994, 386-397.

PfAß_93 A. Pfitzmann, R. Aßmann: More Efficient Software Implementations of (Generalized) DES; Computers & Security 12/5 (1993) 477-500.

Wien_93 M. J. Wiener: Efficient DES Key Search; presented at Rump Session of Crypto 93, TR-244, School of CS, Carleton Univ., Ottawa.

6.1.3 Betriebsarten von Blockchiffren

Auch Modi genannt.

A. Ziele

Geg.: Blockchiffre = etwas wie DES:

- Familie von Permutation auf z.B. 64-bit-Strings.
- Mit z.B. 128 Bits Schlüssel k eine Permutation π_k aus Familie wählbar.
- π_k effizient zu berechnen.
- Nicht immer gebraucht: π_k^{-1} auch effizient zu berechnen.

Gesucht: Nützliche Systeme für richtige Nachrichten (immer noch effizient).

- Hier symmetrische Verschlüsselungssysteme.
- (• Später auch sym. Authentikation u.a.)

Zur Sicherheit:

- Übliche Voraussetzung: Familie von Permutationen sei sicheres Verschlüsselungssystem für 64 bits.

  Damit ist keine der folgenden Konstruktionen beweisbar.

  (Explizite Gegenbeispiele im Skript.)

- Alternative: Tun, **als ob** Permutationen echt zufällig wären, und prüfen, ob Konstruktionen wenigstens damit sicher [BeKR_94].

B. Begriffe

- **Blockchiffre:** Verschlüsselungssystem für Zeichenketten *fester* Länge.
- **Stromchiffre:** (sehr mehrdeutig gebrauchter Begriff).
  - **(Hier):** Verschlüsselungssystem für Zeichenketten *variabler* Länge.
  - **Etwas enger:** Nur solche, die
    a) Nachrichtenanfang unabhängig vom Ende codieren
    b) nicht einfach Blockchiffre auf jeden Block der Nachricht anwenden.
  - **Noch enger:** Nur Pseudo-one-time-Pad, (= Vernam-Chiffre mit Pseudozufallsbits statt echten) oder gar den Pseudozufallsgenerator selbst.

- Für Stromchiffren (≈ **Fehlerausbreitung**)
  - **Synchron:** Verschlüsselung eines Zeichens abhängig von ganzer bisheriger Nachricht.

    ⇒ Sender und Empfänger dürfen nicht aus dem Takt kommen, Fehler auf Leitung nicht toleriert.
  - **Selbstsynchronisierend:** Verschlüsselung eines Zeichens nur abhängig von wenigen vorigen Schlüsseltextzeichen.

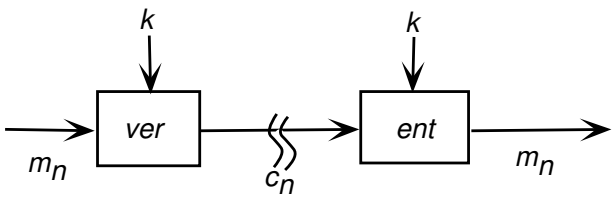
  Allerdings Problem, wenn Zeichen ≠ Bit und die Zeichensynchronisation verlorengeht.

C. „Electronic Codebook“ (ECB)

(„Elektronisches Codebuch“)

- „Triviale“ Betriebsart
- Nicht empfehlenswert, deshalb überhaupt die anderen.

ECB-Schema:



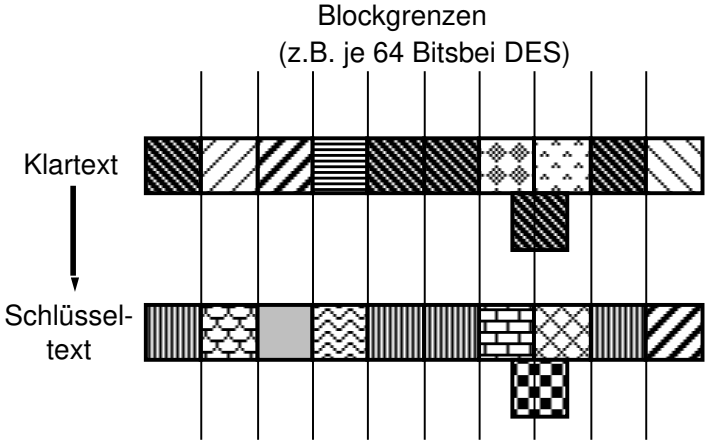
- m_n: n-ter Nachrichtenblock
- c_n: n-ter Schlüsseltextblock
- k: Schlüssel

Fehlerausbreitung: 1-bit → 1 Block.

(Aber Verlust von Blockgrenzen nicht toleriert.)

Problem:

Gleiche Blöcke immer gleich codiert



(64-bit Muster, die nicht auf Blockgrenzen liegen, stören nicht.)

Bsp.: Grobes Fax, häufigster Block ganz weiß

⇒ Angreifer unterscheidet

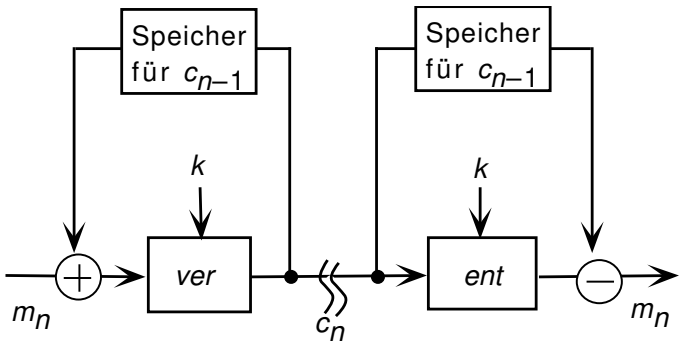
ganz weiß — nicht ganz weiß

Je nach Fax erkennt er das Bild!

Anm.: Kompression hilft.

D. „Cipher-Block Chaining“ (CBC)

(„Blockverkettung“)



⊕ Addition, z.B. bitweise modulo 2
⊖ Subtraktion, z.B. bitweise modulo 2

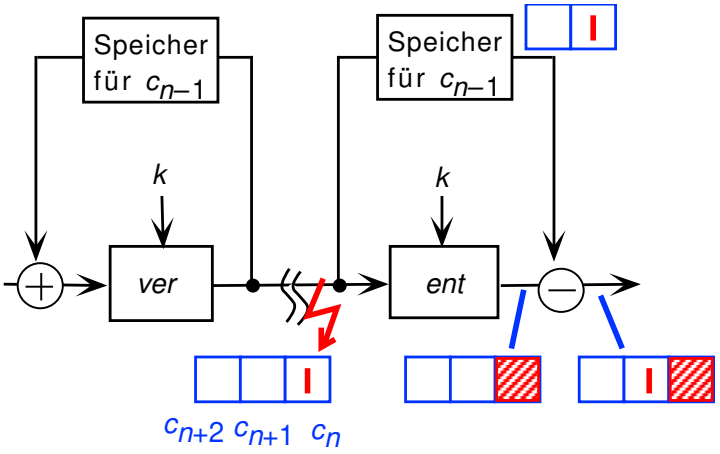
Restproblem von ECB: Gleiche Nachrichtenanfänge gleich codiert. (Nur Problem, falls Schlüssel oft verwendet.)

⇒ zufälligen Klartextblock voranstellen

oder gleich zufällige Speicherinitialisierung („Initialisierungsvektor“) austauschen.

Fehlerausbreitung: Selbstsynchronisierend

(aber Verlust der Blockgrenzen wieder nicht toleriert)

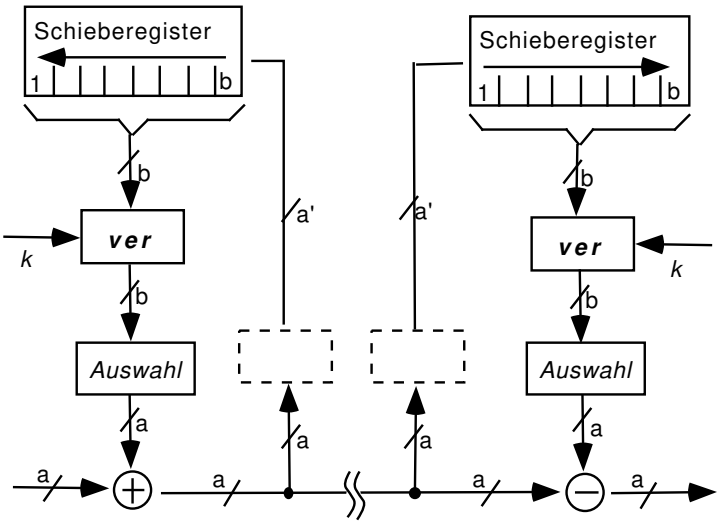


- Ein Ausgabeblock ganz zerstört
- Nächster hat noch 1 Bit Fehler wegen Speicher

E. „Cipher Feedback“ (CFB)

(„Schlüsseltextrückführung“)

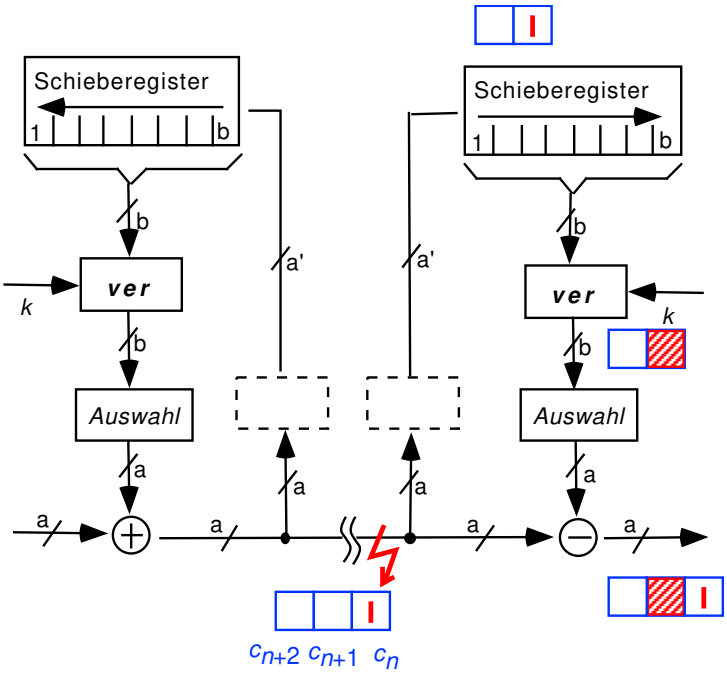
- V.a. für sehr kurze Dateneinheiten:
  - b Blocklänge
  - $a \le b$ Länge jeder Ausgabe



- Gestrichelte Kästchen zusätzl. Auswahl oder Ergänzung, nicht empfohlen
- Initialisierungsvektor nötig, d.h. Startinhalt für Schieberegister als Schlüsselteil.

Fehlerausbreitung: Selbstsynchronisierend:

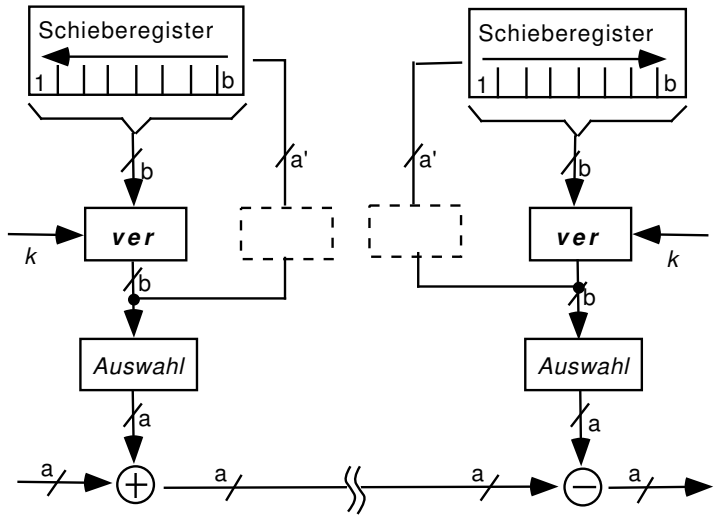
Solange gestört, wie Fehler in Schieberegister.



F. „Output Feedback“ (OFB)

(„Ergebnisrückführung“)

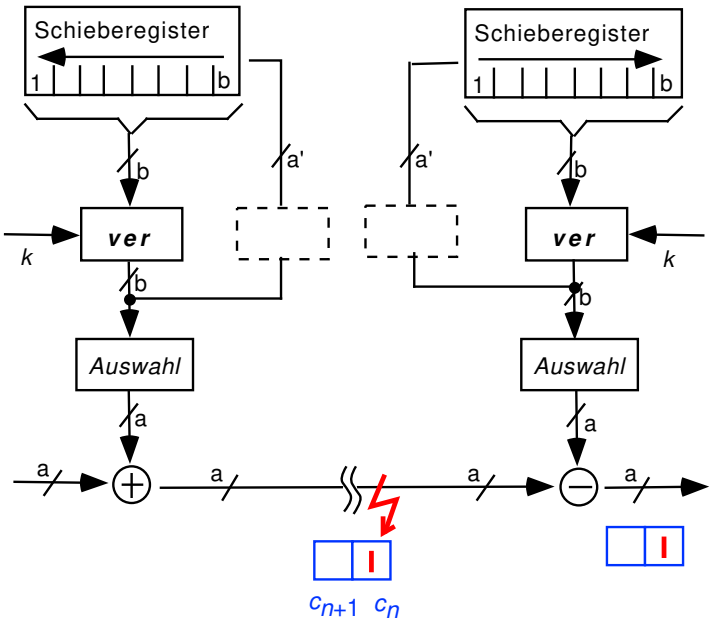
- Ein Pseudo-one-time-Pad (d.h. primär Pseudozufallsgenerator)



- Initialisierungsvektor (Startwert) nötig
- Ausgabelänge a beliebig

Fehlerausbreitung:

- Falsches Bit nur 1 Bit Auswirkung

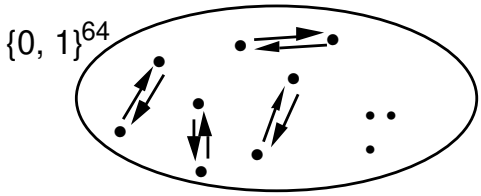


- Aber Synchronisation darf gar nicht verändert werden.

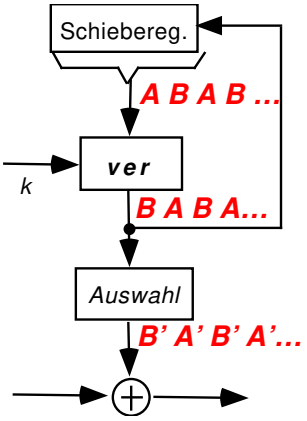
OFB: Pathologisches Gegenbeispiel

Zeigt, daß Betriebsarten nicht unbedingt aus sicheren Blockchiffren sichere Stromchiffren machen:

- **Geg.** Blockchiffre mit Ver- = Entschlüsselung (bei DES sowieso fast)



- **OFB** dann:



G. Sonstiges zu Betriebsarten

- Diese 4 standardisiert.
- Kompliziertere Kombinationen möglich; Sicherheit weniger untersucht.

  Dafür ergibt eine (PCBC) Verschlüsselung und Authentikation gemeinsam.
- **Weitere Unterscheidungskriterien** (wie Fehlerausbreitung):
  - Könnte Blockchiffre indeterministisch sein? (Ja bei Modi, wo Empfänger **ent**schlüsselt: ECB, CBC.)
  - Würde aus asymmetrischer Chiffre wieder asymmetrische? (Ja bei Modi, wo Empfänger **ent**schlüsselt.)

H. Literatur

Zitiert

BeKR_94 Mihir Bellare, Joe Kilian, Phillip Rogaway: The security of cipher block chaining; Crypto '94, LNCS 839, Springer-Verlag, Berlin 1994, 341-358.

Darstellung etwa aus

Pfit7_94 Andreas Pfitzmann: Sicherheit in Rechnernetzen; Skript, Dresden, Version 14.4.95.

Wieder fast jedes Buch.

6.1.4 Sonstige symmetrische Verschlüsselung

- **Historisch**: Substitutionen, Rotormaschinen
- **Weitere moderne Blockchiffren** (rundenbasiert, Konfusion-Diffusion), z.B.
  - IDEA, s.o.
  - (• FEAL: jap. System, schneller als DES, gegen differentielle Kryptoanalyse empfindlich.)
  - RC5 (von RSA-Firma, IDEA-ähnlich, Vorgänger RC2 geheim).
  - CAST (DES-ähnlich, in PGP5 u.a.).

  Siehe z.B. Schneier-Buch.
- **Pseudo-(One-time-Pad)**: Wie One-time-Pad, aber mit Pseudozufallszahlen.
  - Nur Startwert ausgetauscht
  - Für mehrere Nachrichten wieder Gedächtnis nötig: Zustand des PZG
  - Konkrete PZG in Kapitel 6.6.
- **Asymmetrische** Verfahren symmetrisch benutzbar oder anstelle symmetrischer.

6.2 Asymmetrische Verschlüsselung

6.2.1 RSA-Verschlüsselung

Bekanntestes System, von Ronald Rivest, Adi Shamir, Leonard Adleman [RSA_78].

Beruht auf Faktorisierungsannahme, aber nicht als so sicher wie diese bewiesen.

A. Kernidee: RSA als Trapdoor-Einwegpermutationen

(Vgl. Kap. 2.3.2)

Schlüsselgenerierung:

1. Wähle

   $$p, q$$

   prim, zufällig, unabhängig, $|p|_2 \approx |q|_2 = l$ und $p \neq q$..
   - Einzelheiten Kap. 5.4.3.
   - Evtl. sog. „safe primes“ (vgl. Kap. 5.4.2B), zusätzlich RSA-spezifisch: $p'-1$ hat großen Primfaktor p''.

2. $n := p \cdot q$.
3. Wähle

   $$e \in_R \mathbb{Z}_{\phi(n)}^*.$$

4. $d := e^{-1} \bmod \phi(n)$.

   Erinnerung: $\phi(n) = (p-1)(q-1)$;

   Berechnung mit erw. Euklidschen Algo.

Öffentlicher Schlüssel:

$$pk := (n, e)$$

Geheimer Schlüssel:

$$sk := (p, q, d)$$

Öffentliche Operation f:

- Nachrichtenraum $M_l = \mathbb{Z}_n$.
- Funktion: Exponentieren mit e mod n.

$$f(pk, m) := m^e \bmod n.$$

Geheime Operation f^{-1}:

Ziehen der e-ten Wurzel mod n.

Wirklich Trapdoor-Einwegpermutation?

Beh. 1: Ziehen der e-ten Wurzel ist gerade Exponentieren mit d mod n, d.h. $\forall m \in \mathbb{Z}_n$:

$$(m^d)^e \equiv m \equiv (m^e)^d.$$

Somit f^{-1} effizient berechenbar:

$$f^{-1}(sk, y) = y^d \bmod n.$$

Beh. 2: Exponentieren mit e mod n ist eine **Permutation**.

Beweis 2:

- Nach 1.: Jedes y hat e-te Wurzel, d.h. Surjektivität.
- Definitionsbereich = Wertebereich

  $\Rightarrow$ auch injektiv; d.h. jedes y hat **genau** 1 e-te Wurzel.

Anm. zu Bew. 1: Naheliegend: Verwende allgemeinen kleinen Fermatschen Satz

$$m^{\phi(n)} \equiv 1 \bmod n.$$

Gilt aber nur für $m \in \mathbb{Z}_n^*$, deshalb mod p und q einzeln rechnen.

Beweis 1: Mit kleinem Fermatschen Satz.

Klar: $(m^d)^e = m^{ed} = (m^d)^e$.

$$ed \equiv 1 \mod \phi(n)$$

$$\Rightarrow ed \equiv 1 \mod p\text{–}1$$

$$\Rightarrow ed = k(p\text{–}1) + 1 \text{ für ein } k \in \mathbb{Z}.$$

Also

$$\boldsymbol{m^{ed}} = m^{k(p-1)+1}$$

$$= \boldsymbol{(m^{p-1})^k \cdot m}.$$

1. Fall: $m \neq 0$, d.h. $m \in \mathbb{Z}_p^*$: Kleiner Fermat:

$$m^{ed} \equiv (1)^k \cdot m$$

$$\equiv m \mod p.$$

2. Fall: $m = 0$

$0^{ed} \equiv 0$.

Analog

$$m^{ed} \equiv m \mod q.$$

Mit Chin. Restsatz:

$$m^{ed} \equiv m \mod n.$$ □

Sicherheit

- Bisher **nicht** bewiesen: Ziehen von e-ten Wurzeln mit $e \in \mathbb{Z}_{\phi(n)}^*$ so schwer wie Faktorisierung.
- Also **RSA-Annahme(n)**: Ziehen von zufälligen e-ten Wurzeln ist schwer

  (d.h. genau die Sicherheit als Trapdoor-Einwegpermutation).
- Von folgenden Systemen **nicht** mal bewiesen, daß sicher unter RSA-Annahme.

B. Naiver und unsicherer Einsatz von RSA zur Verschlüsselung

Genau die Trapdoor-Einwegpermutation als Blockchiffre:

- Klartextblock ist Zahl m mit $0 \leq m < n$.

  (Meist nur Bitstring der Länge $|n|_2$-1 Bits.)
- $ver(pk, m) := m^e \mod n$.
- $ent(sk, y) := y^d \mod n$.

Passive Angriffe

- **Nur deterministisch.** Also der generelle Probierangriff aus Kap. 2.1.2:

  Ist $m_{Probe}^e \equiv$ *Chiffretext* ?
- **Partielle Information:** Konkret bekannt 1 bit, sog. Jacobi-Symbol (Details hier nicht).

Aktive Angriffe

Grundlage: RSA ist Homomorphismus:

$$(m_1 m_2)^e \equiv m_1^e m_2^e \mod n.$$

Sei c Schlüsseltext, den Angreifer sah.

1. Einfacher Angriff mit 2 gewählten Schlüsseltexten:

- Wähle $c_1 \in \mathbb{Z}_n$ beliebig.
- Setze $c_2 := c \cdot c_1^{-1} \mod n$.
- Lasse Angegriffenen diese beiden entschlüsseln, etwa zu m_1, m_2.
- Errechne $m := m_1 m_2$.

Korrekt:

$$m^e \equiv m_1^e m_2^e \equiv c_1 c_2 \equiv c.$$

2. Trickreicherer Angriff mit 1 gewählten Schlüsseltext:

- Wähle $m_1 \in \mathbb{Z}_n^*$ beliebig und $c_1 := m_1^e \mod n$.
- Sei $c_2 := c \cdot c_1^{-1} \mod n$.
- Lasse c_2 entschlüsseln, etwa zu m_2.
- Errechne $m := m_1 m_2$.

Korrekt: Spezialfall von obigem.

(Sog. **blinde** Entschlüsselung.)

C. Gegenmaßnahmen

2 Angriffsarten zu vermeiden:

- den auf deterministische Verschlüsselung
- aktive Angriffe mittels multiplikativer Struktur.

Zwei Hauptverfahren, zweites wichtiger.

α. Mit Zufallszahl und Redundanz

1. Mit jedem Klartextblock m neue unabhängige Zufallszahl z mitverschlüsseln.

   Z.B. Blocklänge für m als $|n|_2$–100 wählen; letzte 100 bits mit z auffüllen.

2. Verhinderung aktiver Angriffe: Jedem Klartextblock Redundanz hinzufügen. Bsp. mittels Hashfunktion h:

   Idee: Angegriffener beginnt c_2 zu entschlüsseln, aber *red* stimmt hoffentlich nicht
   $\Rightarrow$ keine Ausgabe m_2 an Angreifer.

Gesamtsystem **für kurze Nachrichten**:

- **Verschlüsselung** von Klartextblock m (Länge z.B. $|n|_2$–200).
  - Wähle z der Länge 100 zufällig.
  - Setze $m^* := (m, z, h(m, z))$.
  - $c := m^{*e} \bmod n$.
- **Entschlüsselung** von c:
  - Entschlüssele $m^* := c^d \bmod n$.
  - Zerlege m^* in 3 Teile (m, z, red).
  - Prüfe ob $red = h(m, z)$.

  Falls ok, Ausgabe m.

Sicherheit (nur heuristisch!):

- Um vom angegriffenen Empfänger Ausgabe m_2 zu erhalten, müßte Angreifer c_2 so bilden, daß gilt:
  - $c_1 c_2 = c$
  - c_2 ist von der Form $(m_2, z, h(m_2, z))^e$ für ein unbekanntes m_2.

  Hoffentlich schwierig.
- Hoffentlich keine neuen Angriffe auf kombiniertes System übersehen.

Für lange Nachrichten Betriebsarten möglich (die, wo Empfänger *ent* verwendet).

β. Hybride Verschlüsselung

Prinzip:

- Verschlüssele eigentliche Nachricht m mit **symmetrischem** Verschlüsselungssystem.
- Wähle dazu völlig neuen Schlüssel k.
- Verschlüssele nur k asymmetrisch.

Gesamtsystem dann:

- **Verschlüsselung** von Klartext m:
  - Generiere k fürs symmetrische System.
  - $c := (k^e \bmod n, ver_{sym}(k, m))$.
- **Entschlüsselung** von c:
  - Zerlege c in 2 Teile (c_{asym}, c_{sym}).
  - Entschlüssele c_{asym}:

    $$k := c_{asym}^d \bmod n.$$

  - Entschlüssele c_{sym}:

    $$m := ent_{sym}(k, c_{sym}).$$

Zusätzliche Vorteile hybrider Verschlüsselung:

- **Lange Nachrichten:** Wenn symmetrisches System Eingaben beliebiger Länge zuläßt:

  Obiges System erlaubt Verschlüsseln beliebig langer Texte in einem Stück.
- **Effizienz:** Meist symmetrisches System viel effizienter als RSA (oder andere asymmetrische):

  Hybrides System bei langen Nachrichten fast genauso effizient.

Sicherheit (nur heuristisch)

a) **Indeterminismus** durch k gegeben.

b) **Aktive Angriffe**:

Angreifer läßt wieder c_2 verschlüsseln, jetzt

$$c_{1,asym} \bullet c_{2,asym} = c_{asym}$$

und $c_{2,sym}$ noch wählbar.

(• Da keine Redundanz, gibt angegriffener Empfänger etwas aus.)

- Aber **nicht** $k_2 = c_{2,asym}{}^d$ selbst, sondern nur
$$ent(k_2, c_{2,sym}).$$
- Wenn symmetrisches System sicher gegen Angriff mit gewähltem Schlüsseltext, ist daraus k_2 schwer zu berechnen.
- Damit kann Angreifer nicht $k = k_1 k_2$ berechnen.

Heuristisch hofft man dabei z.B.:

- Kein erkennbarer Zusammenhang
$$ent(k_2, c_{2,sym}) \leftrightarrow ent(k, c_{sym}).$$
- Keine neuen Angriffe übersehen.

γ. Zusammenfassung

Kombination aus

- RSA-Trapdoor-Einwegpermutationen
- und
  - entweder Zufallszahl und Redundanz pro verschlüsseltem Block
  - oder hybride Verschlüsselung

ergibt vollständiges Verschlüsselungssystem.

Hoffentlich sicher:

- auch partielle Information geschützt
- unter beliebigen aktiven Angriffen.

Sicherheitsbetrachtungen nur Ausschluß bestimmter bekannter Angriffe, s.o.

D. Effizienzverbesserungen

Unterscheide:

- **Implementierungstricks** für genau dasselbe System
- **„Kleine" Änderungen** am System.

α. Rechnen modulo Faktoren *p, q*

(Nur Implementierungstrick.)

Geheime Operation bisher:

- Vorweg: $d := e^{-1} \bmod \phi(n)$.
- e-te Wurzel aus y: $y^d \bmod n$.

Beh.: Äquivalent und effizienter:

- Vorweg
$$d_p := e^{-1} \bmod p-1$$
$$d_q := e^{-1} \bmod q-1.$$
- e-te Wurzel aus y:

$$\boldsymbol{w := CRA(y^{d_p} \bmod p,\ y^{d_q} \bmod q).}$$

Korrektheit beweisbar wie vorn.

Effizienz: $l = |p|_2 = |q|_2$.

a) 1 Exponentiation der Länge $2l$:
$$\approx (2l)^3 = 8l^3$$

b) 2 Exponentiation der Länge l:
$$\approx 2l^3$$

Restliche Operationen vernachlässigbar.

Also Verbesserung um Faktor 4.

β. Kleiner öffentlicher Exponent

(Leicht geändertes System.)

- Z.B. $e = 3$ konstant für alle Teilnehmer.
- Einzelne Teilnehmer müssen in *gen* dann ggT(p–1, 3) = 1 = ggT(q–1, 3) sicherstellen.

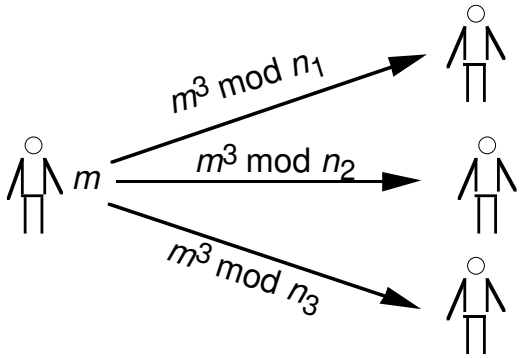
Effizienz:

- Verschlüsseln sehr schnell.
- Öffentliche Schlüssel kürzer.

Sicherheit:

Angriff auf naives System in **spezieller Situation**

Situation:



Angreifer

- hört ab (passiv)
- weiß um Multicast.

1. Berechnet mit CRA

   $$m^3 \bmod n_1 \bullet n_2 \bullet n_3.$$

   Da $m^3 < n_1 n_2 n_3$, ist dies $= m^3$ in $\mathbb{Z}$!
2. Ziehe 3-te Wurzel aus m^3 **in $\mathbb{Z}$**
   (z.B. mit Verfahren in $\mathbb{R}$ oder logarithmischer Suche)

Gegenmaßnahmen:

- Bei probabilistischer Verschlüsselung kein Problem.

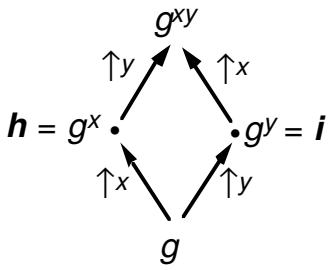
  (Voraussetzung: Unabhängige Zufallszahlen z_i in den 3 Verschlüsselungen.)
- Trotzdem oft $e = 2^{16}+1$ vorgeschlagen.

6.2.2 ElGamal-Verschlüsselung (und Diffie-Hellman-Schlüsselaustausch)

Überblick:

- Diffie-Hellman war erstes effizientes asymmetrisches System [DiHe_76].
- Bißchen schwächer als asymmetrische Verschlüsselung.
- ElGamal-Verschlüsselung ist einfache Erweiterung davon [ElGa_85]
- Beruht auf Standard-Diskreter-Log-Annahme, aber nicht als so sicher wie diese bewiesen.

A. Konstruktionsidee



Exponentieren kommutativ

- ***h*, *i* öffentlich**
  - vorweg bekannte Schlüssel
  - oder eigens zu dem Zweck verschickt.
- **Jeder Besitzer von *x* oder *y* kann g^{xy} ausrechnen.**
- Hoffnung: Sonst kann es niemand:

  Diffie-Hellman-Annahme.

$\Rightarrow g^{xy}$ gemeinsames Geheimnis von Besitzer von x und Besitzer von y.

Klar: Wenn Diskr.-Log.-Annahme gebrochen, dann auch Diffie-Hellman:

$$h \to x \to i^x = g^{xy}.$$

B. Diffie-Hellman-Schlüsselaustausch

(„Public key agreement“, „secret-key exchange“.)

α. Konstruktion grob

- g allgemein bekannt.
- $h, i \triangleq pk_1, pk_2$: öffentliche Schlüssel zweier Teilnehmer.
- $x, y \triangleq sk_1, sk_2$: zugehörige geheime Schlüssel.
- Zum Kommunizieren verwenden sie $k_{12} := g^{xy}$ als gemeinsamen geheimen Schlüssel (z.B. für DES).

β. Einsatz

- Viele solche Teilnehmer, etwa n.
- Jeder veröffentlicht 1 öffentlichen Schlüssel.
- Jetzt hat jedes der $\frac{n(n-1)}{2}$ Paare einen geheimen Schlüssel

γ. Nachteile gegen asymmetrische Verschlüsselung

- **Verschlüsselung auch von Sender-schlüsseln abhängig**
  a) unpraktisch
  b) nicht für anonyme Kommunikation geeignet
- **Gruppe (z.B. p für $\mathbb{Z}_p$) und g müssen für alle gleich sein.** Wer wählt sie?
  - Zentrale?
    - Man muß ihr vertrauen, oder
    - viel stärkere kryptographische Annahme als bisher nötig:
      Soll diskr. Log. nicht berechnen können, selbst wenn sie p, g irgendwie speziell wählt.
      Zur Zeit nicht gebrochen (bei kleinen zusätzlichen Tests), gilt aber als riskant.
  - Gemeinsamer Münzwurf mehrerer Zentralen.

  - Gemeinsamer Münzwurf aller Teilnehmer; großer Aufwand.
  - Alte Zufallszahltabellen (z.B. von RAND-Corporation).

δ. Sicherheit

Probleme (g^{xy} „ganz“ geheim?) siehe ElGamal-Verschlüsselung.

C. Naive und unsichere ElGamal-Verschlüsselung

Jetzt Senderhälfte des Rautendiagramms spontan zu jeder Nachricht gewählt.

Schlüsselgenerierung:

- Wähle Primzahl $\boldsymbol{p}$ mit l Bits.
- Wähle Generator $\boldsymbol{g}$ von $\mathbb{Z}_p^*$.
- Wähle Exponent $\boldsymbol{x} \in_R \mathbb{Z}_{p-1}$.
- Setze $h := g^x \bmod p$.

Öffentlicher Schlüssel:

$$pk := (p, g, h)$$

Geheimer Schlüssel:

$$sk := (p, g, \boldsymbol{x})$$

($\triangleq$ Standard-DL-Annahme; Einzelheiten der Generierung in Kap. 5.5.3A).

Verschlüsselung:

Für $ver(pk, m)$ mit $pk = (p, g, h)$:

- Wähle $y \in_R \mathbb{Z}_{p-1}$.
- Verwende $i = g^y$ als ersten Teil des Schlüsseltextes c.
- „Eigentliche Verschlüsselung“ mit $k := h^y$, ursprünglich

$$c^* := m \bullet k \bmod p.$$

Also $c = (i, c^*)$.

Entschlüsselung:

Für $ent(sk, c)$ mit $sk = (p, g, x)$ und $c = (i, c^*)$:

- Bilde $k := i^x$.
- „Eigentliche Entschlüsselung“ mit k, ursprünglich

$$m := c^* / k \bmod p.$$

In dieser Version Nachrichtenraum $\mathbb{Z}_p^*$.

D. Zur Sicherheit

1. Nicht bewiesen, daß Diffie-Hellman-Annahme so schwer zu brechen wie Diskreter-Log-Annahme.
2. Man erhält **partielle Information** über m.
3. Unsicher gegen **chosen-ciphertext-Angriff**

(4. Probabilistisch ist es aber schon, kein Probierangriff)

α. Partielle Information

Idee: Wenn $h = g^x$ oder $i = g^y$ in **Untergruppe** liegen, dann auch g^{xy}

$\Rightarrow$ Information über g^{xy}

$\Rightarrow$ Information über $m = c \bullet g^{-xy}$.

Beispiel: p–1 gerade, also gibt's Untergruppe

$$H := G_{(p-1)/2}$$

der Ordnung $(p–1)/2$. (Halbe Gruppe.)

Beh.: Angreifer erfährt, ob $m \in H$, d.h. ein Bit über m.

Beweis:

1. Beachte (für alle x):

$$g^x \in H \Leftrightarrow 2|x.$$

2. Effizienter Test, ob $h \in H$ (für alle h):

$$h^{(p-1)/2} = 1?$$

3. Angreifer kann testen, ob $\boldsymbol{g^{xy} \in H}$, denn

$$\begin{aligned} g^{xy} \in H &\Leftrightarrow 2|xy \\ &\Leftrightarrow 2|x \vee 2|y \qquad \text{(da 2 prim)} \\ &\Leftrightarrow h \in H \vee i \in H. \end{aligned}$$

Die letzte Zeile kann er mit Test aus 2. testen.

4. Damit kann er testen, ob $\boldsymbol{m \in H}$, mittels

$$(g^{xy} \in H \wedge c \in H) \vee (g^{xy} \notin H \wedge c \notin H).$$

Denn:

$$\begin{aligned} g^a \bullet g^b \in H &\Leftrightarrow 2|(a + b) \\ &\Leftrightarrow a, b \text{ beide gerade oder beide ungerade} \end{aligned}$$

$$\Leftrightarrow g^a, g^b \text{ beide in } H \text{ oder keines.}$$

b. Gegenmaßnahmen (heuristisch)

1. **Andere „eigentliche Verschlüsselung“** mit $k = g^{xy}$, z.B. mit DES:

$$c^* := \mathrm{DES}(g^{xy}, m)$$

bzw., weil g^{xy} lang:

$$c^* := \mathrm{DES}(hash(g^{xy}), m). \qquad (*)$$

Erreicht: Kenntnis, ob $k = g^{xy} \in H$ sagt hoffentlich nichts über m aus.

2. Falls Verschlüsselung in Gruppe sein soll ($\rightarrow$ höhere Protokolle): In **Untergruppe $\boldsymbol{G_q}$** primer Ordnung statt in $\mathbb{Z}_p^*$.
   - \+ Dort keine Untergruppen.
   - – Problem: Nachrichtenraum ist jetzt G_q, nicht einfach Blöcke.

γ. Chosen-ciphertext-Angriff

Angreifer will entschlüsseln:

$$c = (i, c^*).$$

- Er sendet an Empfänger

$$c_1 := (i, c_1^*)$$

mit beliebigem c_1^*.

- Empfänger entschlüsselt:

$$m_1 := c_1^* \ / \ i^x.$$

- Angreifer errechnet „eigentlichen Schlüssel“ $k = g^{xy} = i^x$ als

$$k = c_1^* \ / \ m_1.$$

- Er entschlüsselt damit c^* zu $m = c^* \ / \ k$.

δ. Gegenmaßnahmen (heuristisch)

1. **Wichtigste:** Selbe wie in (❋) scheint ok:

$$c^* := \mathrm{DES}(g^{xy}, m).$$

Erreicht: Angreifer kann aus (c_1^*, m_1) nicht auf $k = g^{xy}$ schließen (wenn symmetrisches System sicher gegen Chosen-ciphertext-Angriff), also nicht c^* entschlüsseln.

2. **Alternativ**: **Redundanz** in Nachricht einfügen**:** Verhindern, daß Empfänger „unechte Schlüsseltexte“ (i, c_1^*) entschlüsselt: m_1 hoffentlich nicht gültig, also nicht ausgegeben.

3. **Anm.:** Speicherung aller bisherigen Werte i und Verbot der Wiederholung reicht nicht:
   - Angreifer sendet

$$c_1 := (i^z, c_1^*)$$

für beliebiges z, c_1^*. (Sog. „**blinding**“: Unkenntlichmachen von i.)
   - Empfänger entschlüsselt:

$$m_1 := c_1^* \ / \ i^{zx}.$$

   - Angreifer errechnet „eigentlichen Schlüssel“ $k = g^{xy} = i^x$ als

$$k = (c_1^* \ / m_1)^{z^{-1}}.$$

D.h. z-te Wurzel mod p wie in RSA. Voraussetzung $\mathrm{ggT}(z, \phi(p)) = 1$.
   - Er entschlüsselt damit c^* zu $m = c^* \ / \ k$.

ε. Zusammenfassung

Benutzung von

- ElGamal-Verschlüsselung
- mit symmetrischer Verschlüsselung statt Multiplikation als „eigentlicher Verschlüsselung“

nach Formeln der Art (❋) ergibt vollständiges Verschlüsselungssystem.

Anm.: Sehr ähnlich wie hybride Verschlüsselung, nur daß k nicht mit Originalsystem asymmetrisch verschlüsselt wird, sondern Teil des geänderten Systems ist.

Hoffentlich sicher:

- auch partielle Information geschützt
- unter beliebigen aktiven Angriffen.

Sicherheitsbetrachtung nur Ausschluß bestimmter Angriffe, s.o. Insbesondere:

- Erhält man vielleicht mehr Bits über $k = g^{xy}$?
- Wird sym. Verschlüsselung teilweise unsicher, wenn man diese Bits von k weiß?

6.2.3 Weitere Ad-hoc-Systeme

(D.h. ohne Sicherheitsbeweise)

- Rucksacksysteme: Viele Varianten, fast alle gebrochen ⇒ viel zu riskant.
- RSA-ähnliche (Rabin, Lucas-Funktionen, ...).

  Lohnt nicht, wenn auch höchstens so sicher wie Faktorisierung. (Oft weniger sicher oder äquivalent.)
- Elliptische Kurven: ElGamal-artige auf anderen Gruppen
- McEliece: Mit fehlertolerierenden Codes; wegen langer Schlüssel nicht verwendet.

6.2.4 Skizze von Sicherheitsdefinitionen und beweisbar sicheren Systemen

A. Keine partielle Information

≈ Polynomialer Angreifer erfährt „nichts über m".

2 äquivalente Formulierungen:

a) **Ununterscheidbarkeit:** Angreifer kann keine 2 Nachrichten $\{m_1, m_2\}$ so wählen, daß er anhand eines Schlüsseltexts wesentlich besser als mit Wahrscheinlichkeit 1/2 die Nachricht unter diesen beiden raten kann.

b) **Semantische Sicherheit:** Es gibt keine (effizient berechenbare) Funktion von Nachrichten (z.B. gerade/ungerade, prim, Quersumme), die Angreifer anhand Schlüsseltext wesentlich besser bestimmen kann, als indem er den häufigsten Funktionswert rät.

Einzelheiten (Quantorenreihenfolge …) und Äquivalenz nicht trivial, bei Interesse siehe [MiRS_88].

Anm.: Bei nur **passiven** Angriffen gibt's in diesem Sinn beweisbar sichere Systeme, v.a.:

- Ähnlich effizient wie RSA unter Faktorisierungsannahme [BlGo_85].
- Im Prinzip aus beliebigen Trapdoor-Einwegpermutationen [Yao1_82]-

Nützt in Praxis nichts, da gegen aktive Angriffe unsicher.

B. Sicherheit gegen aktive Angriffe

- **Allgemeinste Definition:** Angreifer kann sich adaptiv beliebige Schlüsseltexte c_i entschlüsseln lassen, außer einem gegebenen c.

  Dann: Trotzdem keine partielle Information über m in c.

(• **Schwächer:** Er kann nur vorweg einiges entschlüsseln lassen, aber nicht mehr, nachdem er c gesehen hat.)

Konstruktionsidee (aus [BlFM_88]): Zu jedem Schlüsseltext c muß Sender beweisen, daß er den Inhalt selbst kennt.

Konstruktionen bekannt, aber sehr ineffizient, siehe v.a. [DoDN_91].

Definitiv ein unfertiges Forschungsgebiet!

6.2.5 Literatur

Zitiert:

BlFM_88 M. Blum, P. Feldman, S. Micali: Non-interactive zero-knowledge and its applications ; 20th Symposium on Theory of Computing (STOC), ACM, New York 1988, 103-112.

BlGo_85 M. Blum, S. Goldwasser: An Efficient Probabilistic Public-Key Encryption Scheme Which Hides All Partial Information; Crypto '84, LNCS 196, Springer-Verlag, Berlin 1985, 289-299.

DiHe_76 W. Diffie, M. E. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory 22/6 (1976) 644-654.

DoDN_91 D. Dolev, C. Dwork, M. Naor: Non-Malleable Cryptography; 23rd Symposium on Theory of Computing (STOC), ACM, New York 1991, 542-552.

MiRS_88 S. Micali, Ch. Rackoff, B. Sloan: The Notion of Security for Probabilistic Cryptosystems; SIAM Journal on Computing 17/2 (1988) 412-426.

RSA_78 R. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; Communications of the ACM 21/2 (1978) 120-126.

Yao1_82 A. Yao: Theory and Applications of Trapdoor Functions; 23rd Symp. on Foundations of Computer Science (FOCS), IEEE, 1982, 80-91.

Bücher:

RSA überall, ElGamal in neueren. Oft fehlen Angriffe. Konkret:

Schneier: *RSA mit Angriffen außer partieller Info, ohne Gegenmaßnahmen*

ElGamal ohne alles

Menezes et al.: *RSA mit allem außer partieller Info, ElGamal ohne alles.*

6.3 Symmetrische Authentikation

6.3.1 Informationstheoretisch sichere symmetrische Authentikation

Überblick:

- Möglich (genaue Def. siehe unten).
- Ziemlich lange in Praxis kaum beachtet.
- Deutlich praktikabler als info-th. sichere Verschlüsselung: Nur begrenzte Anzahl Schlüsselbits pro Nachricht.

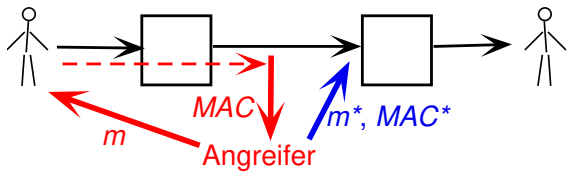
Anm. zur Geschichte:

- Eingeführt 1974 [GiMS_74], vermutlich vorher von Simmons, vgl. z.B. [Sim3_88].
- Erstes praktikables System [WeCa_79].

A. Definition

Angriffs- und Erfolgsart:

- Aktive Angriffe auf Sender und existentielle Fälschung betrachtet.
- Angriffe auf Empfänger denkbar, aber bei jedem Mal maximal ein Wert *MAC* ausgeschlossen $\Rightarrow$ meist vernachlässigt.
- Meist nur 1 Nachricht betrachtet; mehrere gehen aber effizienter als jede einzeln (s.u.)
- Aktiver Angriff bei 1 Nachricht:



Definition 1 (Sicherheit für 1 Nachricht):

$\forall$ prob. Funktionen A_1, A_2 (2 Angreiferschritte)

$P\big(test(k, m^*, MAC^*) = ok ::$ (W'keit v. Angreiferziel wenn…)

$k \leftarrow gen(l, \bullet);$ (Brave wählen)

$(m, loc) \leftarrow A_1(l);$ (Aktiver Angriff; *loc* = lokale Variable)

$MAC := auth(k, m);$ (Reaktion d. Senders)

$(m^*, MAC^*) \leftarrow A_2(l, m, loc, MAC)\big)$ (Angreifer rechnet)

$\leq 2^{-l}$ (Expo. klein)

Üblichere Definition 2:

- Ab jetzt Fall, daß *test*(*k*, *m*, *MAC*) geht, indem Empfänger *MAC* mit *auth*(*k*, *m*) vergleicht.
- Etwas schärfer: W'keit nur über letzten Teil, A_2; Rest Allquantoren.

$\forall l$

$\forall k \in [gen(l, \bullet)]$ (Alle Schlüssel)

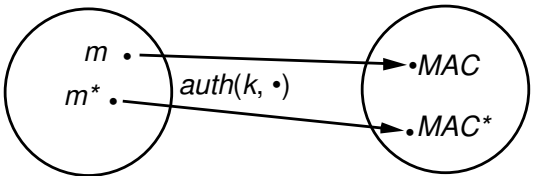
$\forall\ m \in M_l$ und $MAC := auth(k, m)$ (Nachricht mit echtem MAC)

$\forall\ m^*, MAC^*$ (alle denkbaren „Fälschungen)

$$P\big((MAC^* = auth(k, m^*) \mid MAC = auth(k, m)\big) \leq 2^{-l}$$

(W'raum über Wahl von *k* bzw. *z*)

Bildlich:



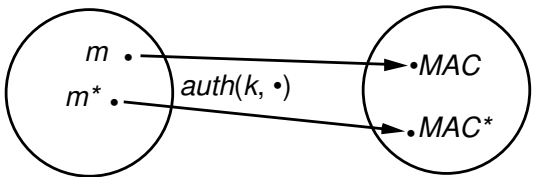
Nicht viele *k*'s mit dieser Eigenschaft

Lemma: Definition 2 („rückwärts") $\Rightarrow$ Definition 1 („vorwärts")

Rein technischer Beweis, weggelassen.

B. Hinreichendes Kriterium

Ziel war: jeweils wenig *k*'s so daß:



Jetzt: Für jedes Paar (*MAC*, *MAC**) gleich viele (und somit nie besonders viel).

Sei B_l Menge möglicher *MACs*.

Kriterium formal:

- Alle Schlüssel in K_l gleichwahrscheinlich.
- $|B_l| \geq 2^l$. (viele mögl. MACs)
- Es gibt Konstante *c* mit

  $\forall m \neq m^* \in M_l, \forall MAC^*, MAC \in B_l$:

  $$c = \big|\{k \mid auth(k, m) = MAC \wedge auth(k, m^*) = MAC^*\}\big|.$$

  Menge der Funktionen *auth*(*k*, •) heißt dann **strongly universal$_2$**.

Beweis gegen Def. 2:

$$P\big((MAC^* = auth(k, m^*) \mid MAC = auth(k, m)\big) = \frac{|\{k \mid MAC^* = \ldots \wedge MAC = \ldots\}|}{|\{k \mid MAC = \ldots\}|} = \frac{c}{|B_l| \bullet c} \leq 2^{-l}.$$

C. Bekannteste konkrete Funktionen: Wegman-Carter-Basissystem

(Aus [WeCa_79])

- $M_l = \{0, 1\}^l$. (Nachrichten fester Länge)
- Wähle Primzahl $p > 2^l$. Alle Rechnungen im Körper $\mathbb{Z}_p$.

  p kann allgemein bekannt und fest sein.
- *gen*: Wähle $a, b \in_R \mathbb{Z}_p$,

  $k = (a, b)$.
- *auth*: Bei Schlüssel $k = (a, b)$:

  $auth(k, m) = a + b \bullet m$ in $\mathbb{Z}_p$.

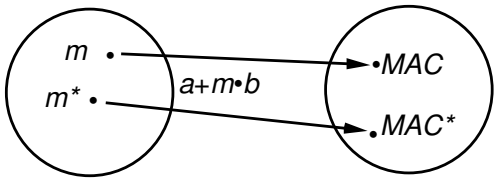
Anm.: Mit anderen Körpern geht's auch.

Beispiel: Sei $l = 3$, d.h. Fehlerw'keit 1/8: Wähle $p = 11$.

- Schlüssel z.B. $a = 6, b = 3$.
- Nachricht z.B. $m = 5$:

  $MAC := (6 + 3 \bullet 5) = 10 \bmod 11$.

Sicherheitsbeweis: Prüfe Kriterium für strongly universal$_2$:



Zähle diese Schlüssel $k = (a, b)$:

$(a + b \bullet m) = MAC \ \wedge \ (a + b \bullet m^*) = MAC^*$

2 lineare Gleichungen:

$$\begin{pmatrix} 1 & m \\ 1 & m^* \end{pmatrix} \rightarrow \begin{pmatrix} 1 & m \\ 0 & m^* - m \end{pmatrix}$$

Rang der Matrix ist 2, also immer genau 1 Lösung $\Rightarrow$ Kriterium gilt mit $c = 1$.

Effizienz:

- $|MAC|_2 = |p|_2 = l + 1$
- $|k|_2 = 2(l + 1)$.
- Zeitaufwand gering ($\approx$ 1 Multiplikation).

Gut für kurze Nachrichten. Für lange noch nicht!

(Man könnte l > Nachrichtenlänge wählen, wäre ähnlich One-time-Pad.)

D. Optimalität für kurze Nachrichten

Beh.: Wenn W'keit des Angreifererfolgs $\leq 2^{-l}$ gewünscht, gilt.

a) $|B_l| \geq 2^l$. (Maclänge $\geq l$)

b) $|K_l| \geq 2^{2l}$. (Schlüssellänge $\geq 2l$)

Beweisskizze:

a) Annahme: $< 2^l$ mögliche MACs.

Angreifer nähme zu gegebenem m den wahrscheinlichsten.

$\Rightarrow$ Erfolgsw'keit $> 2^{-l}$.

b) Betrachte beliebige m, m^*.

- MAC zu m muß mindestens 2^l Werte annehmen können.
- Für gegebenes (m, MAC) muß MAC^* zu m^* noch 2^l Werte annehmen können.

$\Rightarrow$ Mindestens $2^l \bullet 2^l$ Kombinationen.

$\Rightarrow$ Mindestens 2^{2l} Schlüssel.

E. Mehrere Nachrichten

Geg. beliebiger Auth.code für 1 Nachricht.

- **Einfach:** Tausche N Schlüssel k_1, k_2, … aus; verwende einen pro Nachricht.
- **Meist effizienter:** Wenn Schlüssel länger als MAC:
  - $k_{neu} = (k_{alt}, z_1, z_2, \ldots)$ mit
    - k_{alt} nach Verfahren für 1 Nachricht
    - z_1, z_2, … One-time-Pads für MACs.
  - $auth_{neu}(k, m_i) = auth_{alt}(k_{alt}, m_i) \oplus z_i$.
- **Sicherheit; Skizze:**

  Verschlüsselung perfekt

  $\Rightarrow$ $MAC_{neu,i}$ gibt keine Information über MAC_{alt} und somit über k_{alt}.

  $\Rightarrow$ Man kann Sicherheit eines $MAC_{neu,j}$ betrachten, ohne andere $MAC_{neu,j}$ zu berücksichtigen

  $\Rightarrow$ Sicherheit wie bisher.

F. Lange Nachrichten

- [WeCa_79] enthielt **Baumkonstruktion** mit logarithmisch langem Schlüssel.
- Seit kurzem Forschung über **noch kürzere Schlüssel**, z.B. [Kraw1_94, Joha_97] und noch effizientere Berechnung.

Krawczyk's System für 1 Nachricht:

- Betrachte Bitstrings als Polynome über GF(2).
- k = irreduzibles Polynom vom Grad n.
- $auth(k, m) = m \bullet x^n \bmod k$.

D.h. m als Bitstring um n nach links geschoben, dann Polynomdivision mit Rest. Dies ist mod 2 sehr schnelle Operation.

Sicherheit:

- Nur im Zusammenhang mit One-time-Pad-Konstruktion aus Teil E.
- Erfolgsw'keit Angreifer $\leq (m+n)2^{-(n+1)}$.
- Beweis weggelassen. (Nicht sehr schwer, siehe [Kraw1_94].)

G. Variante mit stärkerem Ziel, Skizze

Authentikation mit Schiedsrichter („arbiter“.)

Art schwächere Signaturen:

- – Nur 1 spezieller Dritter kann Dispute entscheiden.
- – Nimmt an Schlüsselgenerierung aktiv teil.
- \+ Geht noch informationstheoretisch (und halbwegs effizient).

Untervarianten, wie sehr man Drittem vertrauen muß.

Bei Interesse siehe z.B. [Simm_88, BrSt_88, JoSm_95].

6.3.2 Beweisbar kryptographisch sichere Authentikation

- Verwende beliebigen informationstheoretisch sicheren Authentikationscode.
- Erzeuge Schlüssel pseudozufällig, tausche nur Startwert aus.

Idee für Sicherheitsbeweis: Wenn poly. Angreifer damit Authentikationssystem brechen könnte, was bei echten Zufallszahlen ja nicht geht, ist es ein Test, der echte und Pseudozufallszahlen unterscheidet.

(So formalisierbar, sobald Pseudozufälligkeit formalisiert.)

Effizienz: Gerade für lange Nachrichten *sehr* gut (teilweise besser als die folgenden heuristischen!)

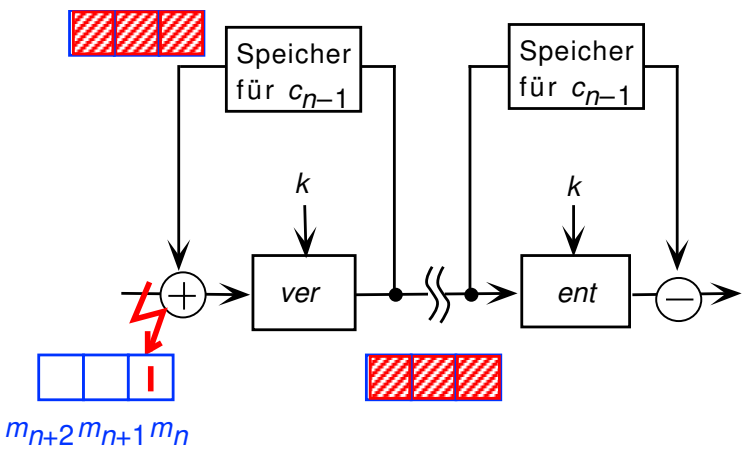
- Pseudozufallszahlen meist nur für die One-time-Pads gebraucht ⇒ ca. 100 bits und vorweg erzeugbar.
- Rest schnelle binäre Operationen.

6.3.3 Authentikationsbetriebsarten von Blockchiffren

Gegeben: Blockchiffre wie in Kap. 6.3.1, z.B. Tripel-DES oder IDEA.

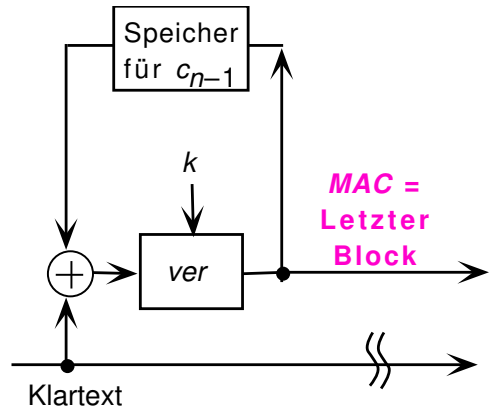
A. „CBC-Auth“

Betrachte Ausbreitung von **Klartextfehler** in CBC:



- Ab einem geänderten Bit aller Schlüsseltext hoffentlich anders.

- Ausnutzen zur Authentikation so: Verwende letzten Block als MAC.



- Empfänger bildet auch $MAC^* = auth_{CBC}(k, m^*)$ und vergleicht mit empfangenem *MAC*.
- In Standard nur linke 32 als *MAC*, ist aber sehr kurz.

B. „CFB-Auth“

Analog aus CFB mit

MAC = letzte Ausgabe von *ver*.

6.3.4 Sonstige symmetrische Authentikation

- Aus **schlüssellosen Hashfunktionen**.
  - Erstmals in [Tsud_92]:

    $auth(k, m) = hash(k_1, m, k_2)$.

    (d.h. Konkatenation von Nachricht mit 2 Schlüsselhälften, dann hashen).
  - Ähnlich HMAC [BeCK_96]:

    $auth(k, m) = hash(k_1, hash(k_2, m)$,

    wobei

    $k_1 = k \oplus opad$

    $k_2 = k \oplus ipad$

    für feste Werte *ipad*, *opad*. Hierzu gewisser Sicherheitsbeweis, aber Annahmen wenig überzeugend (in 6.3.2 viel normalere).
- Ganz **direkte Ad-hoc-Konstruktionen** — spielen keine große Rolle.
- **Signatursystem** liefert immer auch symmetrische Authentikation.

6.3.5 Literatur

Zitiert:

BeCK_96 M. Bellare, R. Canetti, H. Krawczyk: Keying Hash Functions for Message Authentication; Crypto '96, LNCS 1109, Springer-Verlag, Berlin 1996, 1-15.

BrSt_88 E. Brickell, D.. Stinson: Authentication Codes with Multiple Arbiters; Eurocrypt '88, LNCS 330, Springer-Verlag, Berlin 1988, 51-55.

GiMS_74 E. N. Gilbert, F. J. MacWilliams, N. J. A. Sloane: Codes which detect deception; The Bell System Technical Journal 53/3 (1974) 405-424.

Joha_97 T. Johansson: Bucket Hashing with a Small Key Size; Eurocrypt '97, LNCS 1233, Springer-Verlag, Berlin 1997, 149-162.

JoSm_95 Th. Johansson, B. Smeets: On A^2-codes including arbiter's attacks; Eurocrypt '94, LNCS 950, Springer-Verlag, Berlin 1995, 456-460.

Kraw1_94 H. Krawczyk: LFSR-based hashing and authentication; Crypto 94, LNCS 839, Springer-Verlag, Berlin 1994, 129-139.

Sim3_88 G. Simmons: A Survey of Information Authentication; Proceedings of the IEEE 76/5 (1988) 603-620.

Simm_88 G. Simmons: Message authentication with arbitration of transmitter/receiver disputes; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 151-165.

Tsud_92 G. Tsudik: Message Authentication with One-Way Hash Functions; Computer Communication Review 22/5 (1992) 29-38.

WeCa_79 M Wegman, J. L. Carter: New Classes and Applications of Hash Functions; 20th Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1979, 175-182.

Bücher:

- Informationstheoretisch sichere fehlen meist (z.B. in Schneier, Menezes et al.)
- Kommen vor in
  - Gustavus J. Simmons: A Survey of Information Authentication; Gustavus J. Simmons: Contemporary Cryptology – The Science of Information Integrity; IEEE Press, Hoes Lane 1992, 379-419.
  - Douglas R. Stinson: Cryptography - Theory and Practice; CRC Press, Boca Raton, 1995.

  aber primär Spielzeugsysteme und untere Schranken. Originalartikel viel netter zu lesen.
- 6.3.3, 6.3.4 fast überall (Betriebsarten u.ä.).

6.4 Signatursysteme

In vieler Hinsicht wichtigste Systemklasse.

6.4.1 RSA-Signaturen

Erinnerung: RSA von Natur aus Familie von Trapdoor-Einwegpermutationen (Kap. 6.2.1A).

A. Naiver und unsicherer Einsatz von RSA zum Signieren

Genau die Trapdoor-Einwegpermutationen f zum Testen und f^{-1} zum Signieren:

Öffentlicher Schlüssel:

$$pk := (n, e)$$

Geheimer Schlüssel:

$$sk := (p, q, d)$$

Signieren:

$$sign(sk, m) := m^d \bmod n.$$

Testen:

$$test(pk, m, sig) = ok$$

$$:\Leftrightarrow \; sig^e \equiv m \bmod n.$$

Passive Angriffe

Ergeben **existentielle** Fälschungen.

- **Prinzipielle Schwäche von Trapdoor-Einwegpermutation als Sig.sys.:**
  - **Wähle *sig*** beliebig aus Def.bereich, hier $\mathbb{Z}_n$ beliebig.
  - ***m* := *f*(*sig*)**, hier sig^e mod n.

  Dann ist *sig* gültige Signatur zu *m*.

  $\Rightarrow$ Mindestforderung an Anwendung: „sinnvolle“ Texte im Nachrichtenraum sehr selten.
- „**Multiplikativer Angriff“:** (Zur Hinführung auf folgende aktive.)
  - Geg. $sig_1 = m_1^d$ und $sig_2 = m_2^d$.
  - Setze $m := m_1 \bullet m_2$; $sig := sig_1 \bullet sig_2$.

  Homomorphismus $\Rightarrow$ es gilt $sig = m^d$.

Aktive Angriffe

Ergeben **selektive** Fälschung. Sei *m* Nachricht, zu der Angreifer Signatur möchte.

1. **Einfacher Angriff mit 2 gewählten Schlüsseltexten:**
   - Wähle $m_1 \in \mathbb{Z}_n^*$.
   - Sei $m_2 := m \bullet m_1^{-1}$.
   - Lasse m_1, m_2 unterschreiben.

   D.h. $m = m_1 \bullet m_2$ wie oben; $sig := sig_1 \bullet sig_2$.
2. **Trickreicherer Angriff mit 1 gewählten Schlüsseltext:**
   „Judy Moore’s Angriff“.
   - Wähle $sig_1 \in \mathbb{Z}_n^*$, setze $m_1 := sig_1^e$.
   - Sei $m_2 := m \bullet m_1^{-1}$.
   - Lasse m_2 unterschreiben.
   - Setze $sig := sig_1 \bullet sig_2$.

   Korrekt: Spezialfall von obigem.

   (Sog. **blinde** Signaturen; positive Anwendung in bargeldartigen Zahlungssystemen.)

B. Gegenmaßnahmen

α. Mit kollisionsfreien Hashfunktionen

Erinnerung: Einweg und Kollisionsfreiheit …

Prinzip:

- Mit „naiven“ System wird $m^* := hash(m)$ statt *m* signiert.

Gesamtsystem dann:

- **Signieren** von Text *m*:

  $$(hash(m))^d \bmod n.$$
- **Testen** einer Signatur *sig* zu *m*:

  $$sig^e \equiv hash(m)?$$
- **Schlüsselerzeugung:**
  - In Theorie braucht Hashfunktion auch Schlüssel *hk*.
  - In Praxis haben die meisten aber keinen.

  Ggf. *hk* zusätzlicher Teil von geheimem und von öffentlichem Schlüssel.

Zusätzliche Vorteile:

- **Lange Nachrichten:** Wenn Hashfunktion Eingaben beliebiger Länge zuläßt:

  Obiges System erlaubt Signieren beliebig langer Texte in einem Stück.

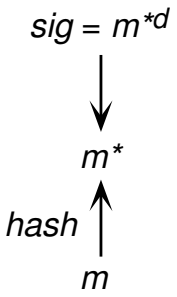
  (Andernfalls bräuchte man **sicheren** Mechanismus zur Blockverkettung.)
- **Effizienz:** Meist Hashfunktion viel effizienter als RSA (oder andere asymmetrische):

  Gesamtsystem bei langen Nachrichten fast genauso effizient.

Sicherheit (nur heuristisch)

a) **Passive Angriffe**:

- Angreifer kann wieder *sig* wählen und dazu $m^* = sig^e$ errechnen.
- Da einweg, findet er aber kein *m* mit $hash(m) = m^*$ dazu.



b) **Aktive Angriffe**:

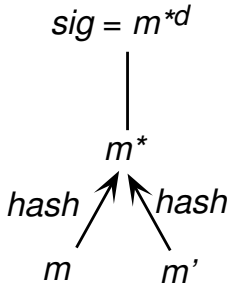
Angreifer möchte mindestens $m_2^* = m^* / m_1^*$ im Basissystem signieren lassen.

Signierer signiert aber nur im Gesamtsystem, also muß Angreifer ihm m_2 mit $hash(m_2) = m_2^*$ geben.

Da einweg, findet Angreifer das nicht.

c) **Angriff mit Kollisionen**: Wenn *hash* nur einweg, aber nicht kollisionsfrei wäre:

- Angreifer wählt *m*, *m'* mit $hash(m) = hash(m')$.
- Läßt *m* signieren.
- Signatur auch für *m'* gültig.



Heuristisch hofft man z.B., daß es keine direkte Umrechnungen von $hash(m_1)^d$ zu $hash(m)^d$ gibt.

Anm.: Bewiesen: Kombination von kollisionsfreier Hashfunktion mit sicherem Blocksignatursystem ist wieder sicher [Damg_88]. Hier nicht anwendbar, da Blocksignatursystem nicht sicher.

β. **Alternative Gegenmaßnahme**

(Nur für kurze Nachrichten.)

Redundanz zur Nachricht hinzufügen.

Gesamtsystem dann:

- **Signieren**: $(m, red(m))^d \bmod n$.
- **Testen** einer Signatur *sig* zu *m*:

$$sig^e \equiv (m, red(m)) \bmod n \,?$$

Sicherheit (nur heuristisch)

Hoffentlich Wahrscheinlichkeit klein, mit obigen Angriffen „gültige“ Nachricht zu erwischen.

Bsp.:

- Nachrichten mit ≥ 100 Nullen aufgefüllt; aber für algebraische Angriffe riskant [JoCh_86].
- Lieber chaotischere Redundanz, z.B. mit DES mit festem Schlüssel.

Hierzu ISO-Standard — man konnte sich wohl nicht auf Hashfunktion einigen [ISO 9796_91].

γ. **Zusammenfassung**

Kombination aus

- RSA-Basissignatursystem und
- Hashen oder Redundanz

ergibt vollständiges Signatursystem, hoffentlich sicher

- gegen existentielle Fälschung
- unter beliebigen aktiven Angriffen.

Sicherheitsbetrachtungen nur Ausschluß bestimmter bekannter Angriffe, s.o.

C. Effizienzverbesserungen

Genau wie für RSA-Verschlüsselungssysteme:

- Rechnen modulo Faktoren *p*, *q*.
- Kleiner öffentlicher Exponent.

  Im Signatursystemfall hier keine Angriffe bekannt.

6.4.2 ElGamal-artige Signatursysteme

Vorweg:

- Diskreter-Log-basiert, nicht beweisbar.
- Sonst kein Zusammenhang mit ElGamal-Verschlüsselung.
- Klasse ähnlicher Systeme, außer ElGamal [ElGa_85] v.a.
  - **Schnorr** [Schn5_91]
  - (sehr Schnorr-ähnlich):

    DSS („Digital Signature Standard“) = DSA („Digital Signature Algorithm“), amerikanischer Regierungsstandard [FIPS-186_94].

A. Grundidee

- Geheimer und öffentlicher Schlüssel immer entsprechend einer Diskr-Log-Annahme, z.B. für ElGamal Standard-DL-Annahme:

$$sk := (p, g, \boldsymbol{x})$$

$$pk := (p, g, h = g^x).$$

- **Signaturen** (r, s) mit
  - r zufällig, s dann errechnet.
  - Dabei wählt Signierer

    z zufällig;

    $r := g^z$.

    (Also $z \approx$ zusätzlicher geheimer Schlüssel speziell für eine Signatur.)
- **Testgleichungen** der Art

$$a^b = c^d e^f \mod p,$$

in denen r, s, m, g, h vorkommen. Alle können sowas testen, aber hoffentlich kann nur Signiererin sie lösen.

Sinnvollerweise g, h, r „unten“. (Allerdings r nachher zusätzlich einmal oben.)

B. Naives und unsicheres ElGamal-System

Testgleichung hier

$$\boldsymbol{g^m = r^s h^r}. \qquad (*)$$

Lösung mit geheimem Schlüssel:
Zuerst $r = g^z$, $h = g^x$ einsetzen:

$$(*) \Leftrightarrow \quad g^m = g^{zs}\, g^{xr} = g^{zs+xr}$$

$$\Leftrightarrow \quad m \equiv zs+xr \mod p\text{–}1$$

(Sätzchen 1 in Kap. 5.5.1B; und g ist Generator, also ord(g) = p–1.)

$$\Leftrightarrow \quad m - xr \equiv zs \mod p\text{–}1$$

$$\Leftrightarrow \quad s \equiv (m - xr)z^{-1} \mod p\text{–}1.$$

ElGamal-Basissystem also

- **Schlüsselerzeugung**: Wie Kap. 5.5.3A.
- **Signieren**:
  - Wähle $z \in_R \mathbb{Z}_{p-1}^*$.
  - $r := g^z \bmod p$;
  - $s \equiv (m - xr)z^{-1} \mod p\text{–}1$
- **Testen** von $sig = (r, s)$:

$$g^m = r^s h^r. \ ?$$

C. Sicherheit

1. Nicht bewiesen, daß Lösen der Testgleichung nach (r, s) so schwer wie DL-Annahme.
2. Lösen der Testgleichung nach (r, s) nicht die einzige Möglichkeit, Signaturen zu fälschen: existentielle Fälschung.
3. Mit aktivem Angriff gibt’s mehr Varianten, aber selektive Fälschung nicht bekannt. (Im Gegensatz zu RSA.)

Existentielle Fälschung mit passivem Angriff

Zu lösen: $g^m = r^s h^r$. (in $\mathbb{Z}_p^*$)

Idee: Wähle r von der Form

$$\boldsymbol{r = g^\alpha h^\beta}$$

und löse Gleichungen im Exponenten.

Genauer: Löse

$$g^m = (g^\alpha h^\beta)^s h^r$$

nach (m, s), d.h.

$$g^{m-\alpha s} = h^{\beta s+r}.$$

Diskreter Log. unbekannt, aber hinreichend: Beide Exponenten = 0, d.h.

- Für h: $\beta s + r \equiv 0 \bmod (p-1)$
  $\Leftrightarrow \quad s = -r\beta^{-1}$.
  Dazu anfangs β mit ggT$(\beta, p-1) = 1$ wählen.
- Für g: $m - \alpha s \equiv 0 \bmod (p-1)$
  $\Leftrightarrow \quad m = \alpha s$.

D. Gegenmaßnahmen

- **Mit kollisionsfreien Hashfunktionen**: Mit „naiven" System wird $m^* := hash(m)$ statt m signiert.
- **Mit Redundanz.**

Vorteile und Sicherheit gegen obigen Angriff wie bei RSA.

Sicherheit nur heuristisch: Paar Ansätze für Angriffe, die nicht funktionieren, z.B.

- ***m*, *r* zuerst wählen**, dann ist Lösen nach s diskreter Logarithmus:
  $$g^{hash(m)} = r^s h^r.$$
- ***m*, *s* zuerst wählen**: Schon nicht klar.
- **Aus mehreren Signaturen Schlüssel bestimmen?**
  Z.B. n lineare Gleichungen
  $$hash(m_i) \equiv z_i s_i + x r_i \bmod p-1.$$
  Unbekannt: $x, z_1, \ldots, z_n$: eins zuviel.
  Hierzu müssen z_i wirklich zufällig oder wenigstens pseudozufällig sein: Jede Abhängigkeit ergibt die fehlende Gleichung.

Schwer, nichts zu übersehen.

E. Varianten

α. Vorwegberechnung

(engl. *precomputation*)

g^z im Hintergrund ausrechnen, bevor m bekannt. (Außer evtl. bei Smartcard, weil keine Batterie.)

Dann Signieren sehr schnell: ohne Exponentiation. (Aber Test nicht. Bei RSA umgekehrt ⇒ Streit, was wichtiger ist.)

β. Testgleichung etwas ändern

$$g^m = \boldsymbol{r^r h^s} \qquad \text{(statt } r^s h^r\text{)}$$

$$\Leftrightarrow \quad g^m = g^{zr+xs}$$

$$\Leftrightarrow \quad m - zr \equiv xs \bmod p-1$$

$$\Leftrightarrow \quad s \equiv (m - zr)x^{-1} \bmod p-1.$$

Vorteil: Berechnung von x^{-1} einmal vorweg, statt jedesmal ein z^{-1} ⇒ 1 Invertierung pro Signieren gespart (Zeit ≈ 30 Multiplikationen).

γ. In Untergruppe

Jetzt $q \mid (p-1)$ prim; g Generator von $G_q \leq \mathbb{Z}_p^*$, siehe Kap. 5.5.3B.

Z.B. **Schnorr-Testgleichung** (Hashen integriert):

$$r = hash(g^s h^r, m).$$

Lösen: Hinreichend ist

$$r = hash(g^{z^*}, m) \ \wedge \ g^{z^*} = g^s h^r.$$

1. z^* frei wählen
2. r mit linker Gleichung ausrechnen.
3. Noch zu lösen:
   $$g^{z^*} = g^s h^r$$
   $$\Leftrightarrow \quad g^{z^*} = g^{s+xr}$$
   $$\Leftrightarrow \quad s \equiv z^* - xr \bmod q.$$

Vorteil: s, r kurz, d.h. nur mod q

⇒ alle Exponenten kurz

⇒ Exponentiationen schneller.

6.4.3 Weitere Ad-hoc-Systeme

- Mehr ElGamal-Varianten
  - Andere Gruppen wie immer.
  - Mehr Testgleichungen.
- Andere polynomiale Gleichungen mod $n = pq$ („Ong-Schnorr-Shamir“ u.ä.) oder Rucksacksysteme: Zu viele gebrochen!
- Fiat-Shamir, Guillou-Quisquater u.ä.: Heuristische Konstruktion aus beweisbar sicheren „Identifikationssystemen“. Beruhen auch auf Faktorisierung oder Diskr. Log.
- Selbes Prinzip mit ganz anderen Primitiven, z.B. [Ster_90, Poin_95]: Können Alternativen sein …

6.4.4 Skizze von Sicherheitsdefinitionen und beweisbar sicheren Systemen

A. Definition

Zu definieren: Sicherheit gegen existentielle Fälschung bei aktivem Angriff.

Erster Versuch:

- Bei aktivem Angriff mit nur x gewählten Nachrichten

  Anm.: Vergleiche Def. von Authentikationscodes mit 1 Nachricht:)

$\forall$ prob. poly. Algo $A_1, \ldots, A_{x+1}$ (Angreiferschritte)

P($test(pk, m^*, sig^*) = ok$ (W'keit daß
$\wedge\ m^* \notin \{m_1, \ldots, m_x\}$ :: Signatur ok und Nachricht neu, wenn)

$(sk, pk) \leftarrow$ ***gen***$(l, \bullet)$; (Signierer wählt)

$(m_1, loc_1) \leftarrow \boldsymbol{A_1}(l, pk)$; (Aktiver Angriff; loc = lok. Variable)

$sig_1 \leftarrow$ ***sign***(sk, m_1); (Reaktion d. Signierers)

$(m_2, loc_2) \leftarrow \boldsymbol{A_2}(loc_1, sig_1)$; (• • • immer abwechselnd • • •

• • •

$(m_x, loc_x) \leftarrow \boldsymbol{A_x}(loc_{x-1}, sig_{x-1})$;

$sig_x \leftarrow$ ***sign***(sk, m_x); (Reaktion d. Signierers)

$(m^*, sig^*) \leftarrow \boldsymbol{A_{x+1}}(loc_x, sig_x)$**)**

(Angreifer fälscht)

$\leq$ **1/*poly*(*l*)** (Vernachlässigbar)

Was fehlt?

- Angreifer selbst wählen lassen, wieviel Nachrichten $x_1, \ldots, x_n$ er signieren läßt.
- Signierer mit Gedächtnis ausstatten, weil beweisbar sichere Systeme das oft brauchen.

$\Rightarrow$ Formaler Begriff interagierender Maschinen nötig statt der Einzelschritte. Z.B. 2 interagierende Turingmaschinen. Vgl. [GoMR_89] für Details.

Grob:

- **Definiere allgemein**
  - *Traffic* (Folge der ausgetauschten Nachrichten) aus Interaktion zweier solcher Maschinen.
  - *Hin* (Teilfolge der Nachrichten von 1. an 2. Maschine)
  - $Ergebnis_1$ (lokale Schlußausgabe der ersten Maschinen)

- **Definiere *Sign**** als interaktive Version des Signieralgorithmus, d.h.
  - Hält intern Zustandsvariable loc_S.
  - Gibt auf Eingabe m immer $sign(sk, loc_S, m)$ aus.

Dann Sicherheitsdefinition:

$\forall$ prob. poly. interaktive Algo A^* (Angreifer)

$\mathbf{P}($ $test(pk, m^*, sig^*) = ok$

$\wedge\ m^* \notin \{m_1, \ldots, m_x\}$ ::

$(sk, pk) \leftarrow \mathbf{gen}(l, \bullet)$;

$hist \leftarrow Traffic(A^*(l, pk), Sign^*(sk))$;

(Aktiver Angriff)

$(m_1, \ldots, m_x) := Hin(hist)$;

$(m^*, sig^*) := Ergebnis_1(hist)\ )$

(Ergebnis des Angreifers sollte Fälschung sein)

$\leq \mathbf{1/poly(}l\mathbf{)}$ (Vernachlässigbar)

B. Beweisbar sichere Systeme

Überblick:

- Gibt's.
- Auf Faktorisierung bzw. Diskretem Log beruhend: Nicht viel ineffizienter als pures RSA oder ElGamal.
- Theoretisch (weniger effizient) aus jeder Einwegfunktion. (Keine Trapdoor nötig!)
- Beweisbar kollisionsfreie Hashfunktionen nicht so effizient wie unbeweisbare $\Rightarrow$ wenn man bei langen Nachrichten so schnell sein will, muß man Hashannahme zusätzlich machen.

In Praxis bisher kaum verwendet, ich würde aber.

Konkrete Systeme:

- **Einmal-Signaturen**: Sehr ineffizient, aber Basis für vieles andere ([Lamp_79], schon in [DiHe_76] dargestellt).

  Verbesserungen [Merk_90, Merk_88, Vaud_92].

- **GMR**, unter Faktorisierungsannahme [GoMR_88, FoPf_91].
- **Effizienter unter RSA-Annahme**: [DwNa_94, CrDa_96].
- **Diskreter Log**: [CrDa_95] (vorher schon Fail-stop-Signaturen, siehe 6.4.5)
- Aus beliebiger **Einwegfunktion**: [Romp_90].

6.4.5 Signatursysteme mit Zusatzeigenschaften

- **Fail-stop:** Falls Annahme gebrochen, wird das bemerkt und ist beweisbar, siehe [PePf_97].
- **Informationstheoretisch sicher** (mit anderen Abschwächungen), siehe [ChRo_91, PfWa_92].
- **Unsichtbar** („undeniable signatures“, schlechter Name). Gewisse Datenschutzaspekte, siehe [ChAn_90]
- **Blind**: Wie aktiver Angriff auf RSA: Signierer sieht Nachricht nicht. Ziel: Angreifer erhält nicht mehr Signaturen also Signierer leistet. Verwendung als „Münzen“ [Cha8_85]
- **Gruppensignaturen**: Mehrere Signierer … (viele Varianten).

Allgemeine Definition [Pfit4_93, Pfit8_96].

6.4.6 Literatur

Zitiert:

Cha8_85 D. Chaum: Security without Identification: Transaction Systems to make Big Brother Obsolete; Communications of the ACM 28/10 (1985) 1030-1044.

ChAn_90 D. Chaum, H. van Antwerpen: Undeniable signatures; Crypto '89, LNCS 435, Springer-Verlag, Berlin 1990, 212-216.

ChRo_91 D. Chaum, S. Roijakkers: Unconditionally Secure Digital Signatures; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 206-214.

CrDa_95 R. Cramer, I. Damgard: Secure Signature Schemes based on Interactive Protocols; Crypto '95, LNCS 963, Springer-Verlag, Berlin 1995, 297-310.

CrDa_96 R. Cramer, I. Damgård: New Generation of Secure and Practical RSA-Based Signatures; Crypto '96, LNCS 1109, Springer-Verlag, Berlin 1996, 173-185.

Damg_88 I. Damgård: Collision free hash functions and public key signature schemes; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 203-216.

DiHe_76 W. Diffie, M. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory 22/6 (1976) 644-654.

DwNa_94 C. Dwork, M. Naor: An efficient existentially unforgeable signature scheme and its applications; Crypto 94, LNCS 839, Springer-Verlag, Berlin 1994, 234-246.

ElGa_85 T. ElGamal: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms; IEEE Transactions on Information Theory 31/4 (1985) 469-472.

FIPS-186_94: Digital Signature Standard; Federal Information Processing Standards Publication 186, U.S. Department of Commerce/ N.I.S.T., National Technical Information Services, Springfield 1994.

FoPf_91 D. Fox, B. Pfitzmann: Effiziente Software-Implementierung des GMR-Signatursystems; Proc. Verläßliche Informationssysteme (VIS), Informatik-Fachberichte 271, Springer-Verlag, Berlin 1991, 329-345.

GoMR_88 S. Goldwasser, S. Micali, R. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM Journal on Computing 17/2 (1988) 281-308.

GoMR_89 S. Goldwasser, S. Micali, Ch. Rackoff: The Knowledge Complexity of Interactive Proof Systems; SIAM Journal on Computing 18/1 (1989) 186-207.

JoCh_86 W. de Jonge, D. Chaum: Attacks on Some RSA Signatures; Crypto '85, LNCS 218, Springer-Verlag, Berlin 1986, 18-27.

Lamp_79 L. Lamport: Constructing Digital Signatures from a One-Way Function; SRI Intl. Report CSL-98, Menlo Park, Oct. 1979.

Merk_88 R. Merkle: A digital signature based on a conventional encryption function; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 369-378.

Merk_90 R. Merkle: A certified digital signature (That antique paper from 1979); Crypto '89, LNCS 435, Springer-Verlag, Berlin 1990, 218-238.

PePf_97 T. Pedersen, B. Pfitzmann: Fail-stop Signatures; SIAM Journal on Computing 26/2 (1997) 291-330.

Pfit4_93 B. Pfitzmann: Sorting Out Signature Schemes; 1st ACM Conference on Computer and Communications Security, acm press 1993, 74-85.

Pfit8_96 B. Pfitzmann: Digital Signature Schemes — General Framework and Fail-Stop Signatures; LNCS 1100, Springer-Verlag, Berlin 1996.

PfWa_92 B. Pfitzmann, M.Waidner: Unconditional Byzantine Agreement for any Number of Faulty Processors ; STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, LNCS 577, Springer-Verlag, Berlin 1992, 339-350.

Poin_95 D. Pointcheval: A New Identification Scheme Based on the Perceptrons Problem; Eurocrypt '95, LNCS 921, Springer-Verlag, Berlin 1995, 319-328.

Romp_90 J. Rompel: One-Way Functions are Necessary and Sufficient for Secure Signatures; 22nd Symposium on Theory of Computing (STOC), ACM, New York 1990, 387-394.

Schn5_91 C. Schnorr: Efficient Signature Generation by Smart Cards; Journal of Cryptology 4/3 (1991) 161-174.

Ster_90 J. Stern: An alternative to the Fiat-Shamir protocol; Eurocrypt '89, LNCS 434, Springer-Verlag, Berlin 1990, 173-180.

Vaud_92 S. Vaudenay: One-time identification with low memory; Eurocode 92, CISM Courses and Lectures Nr. 339, Springer-Verlag, Wien 1992, 217-228.

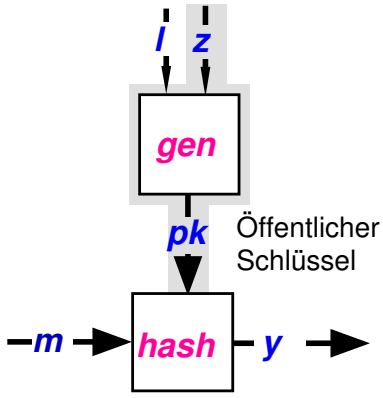
Bücher:

- Menezes et al.;
- (• bei Schneier nur einfache Verfahren.)
- Längerer Überblick u. mehr Lit. zu exotischeren Verfahren [Pfit8_96].

6.5 Hashfunktionen

Erinnerung

- Ziele: Einweg und kollisionsfrei.
- Wenn Eingaben länger als Ausgaben, geht kollisionsfrei nicht informationstheoretisch sicher.
- In Praxis oft ohne Schlüssel, aber Kollisionsfreiheit dann nicht definierbar, für Theorie also:



Sei M_{pk} Nachrichtenraum.

(Allgemeinster Fall, besser M_{pk} immer $\{0,1\}^*$ oder wenigstens nur von l abhängig.)

Genauer:

- Sei $\mathcal{M} = (M_{pk})_{pk \in [gen(\bullet,\bullet)]}$ eine Familie von Nachrichtenräumen.
- Seien effiziente Algorithmen für „$\in_R M_{pk}$“ und Test „$m \in M_{pk}$?“ bekannt.

  D.h.
  - *wahl* mit Eingabe (pk, z') für Zufallszahl z' und
  - *ist_in* mit Eingabe (pk, m).

6.5.1 Definitionen und allgemeine Beziehungen

Formale Def. Kollisionsfreiheit:

$\forall$ prob. poly. Algo. A_{koll} (Angreifer)

$P\big(m, m^* \in M_{pk} \wedge m \neq m^* \wedge$ (Angreifer-
$hash(pk, m) = hash(pk, m^*) ::$ ziel)

$pk \leftarrow gen(l, \bullet);$ (Brave wählen)

$m, m^* \leftarrow A_{koll}(l, pk)\big)$ (Angreifer rechnet)

$\leq 1/\mathrm{poly}(l)$ (Vernachlässigbar)

Einweg in diesem Fall:

$\forall$ prob. poly. Algo. A_{urbild} (Angreifer)

$P\big(m^* \in M_{pk} \wedge$ (Angreifer-
$hash(pk, m^*) = y ::$ ziel)

$pk \leftarrow gen(l, \bullet);$ (Brave wählen)

$m \in_R M_{pk};$ (")

$y := hash(pk, m);$ (")

$m^* \leftarrow A_{urbild}(l, pk, y)\big)$ (Angreifer…)

$\leq 1/\mathrm{poly}(l)$ (Vernachlässigbar)

Anm.: Mittelding „universal one-way“: A_{urbild} bekommt noch m als Eingabe, muß $m^* \neq m$ finden.

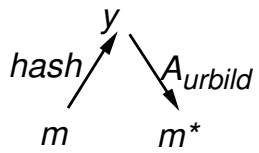
Satz: Kollisionsfrei impliziert einweg wenn „gleichmäßig verkürzend“, konkret z.B. wenn

$$\forall pk \in [gen(\bullet, \bullet)]\ \forall y \in hash(pk, \bullet)\quad |hash_{pk}^{-1}(y)| \geq 2.$$

(Mit $hash_{pk}^{-1}(y) := \{m \mid hash(pk, m) = y\}$.

Folge: Für konkrete Funktionen ggf. nur „kollisionsfrei“ zu betrachten.

Beweisskizze:



Problem: Ist wirklich $m \neq m^*$?

- Immer mindestens 2 Urbilder.
- Angreiferalgo. wählt m^* unabhängig davon, welches Urbild schon m ist

$\Rightarrow$ im Mittel über Wahl von m (!) jedes zweite Mal $m \neq m^*$.

Beweis: Sei die Familie $(gen, hash, \mathcal{M})$ kollisionsfrei und gleichmäßig verkürzend.

Annahme: $\boldsymbol{A_{urbild}}$ sei ein Algo., der Beh. widerspricht.

Sei nun $\boldsymbol{A_{koll}}$ folgender Algorithmus:
Auf Eingabe (I, pk):

$m \in_R M_{pk}$;
$y := hash(pk, m)$;
$m^* \leftarrow A_{urbild}(pk, y)$;
Output (m, m^*).

Offensichtlich probabilistisch polynomial.

Hilfssatz: Erfolgsw'keit von A_{koll} für bestimmtes I ist höchstens um Faktor 2 kleiner als von A_{urbild} für selbes I.

Seien $\boldsymbol{P_{koll}(I)}$ bzw. $\boldsymbol{P_{urbild}(I)}$ diese W'keiten, d.h. genau die aus den Definitionen.

Beweis von Satz aus Hilfssatz:

Idee: Erfolgsw'keit damit immer noch > 1/poly

Genauer heißt „$P_{urbild}(I)$ nicht $\leq 1/\mathrm{poly}(I)$“:

$\exists$Polynom $pol\ \forall I_0\ \exists I \geq I_0 : P_{urbild}(I) > 1/pol(I)$

(d.h. für unendlich viele I …).

Dann gilt für $1/2P_{urbild}(I)$ dasselbe mit Polynom $2pol$.

Beweis des Hilfssatzes:

$P_{koll} = \mathrm{P}\big(m, m^* \in M_{pk} \wedge m \neq m^* \wedge hash(pk, m) = hash(pk, m^*) ::$
$pk \leftarrow gen(I, \bullet);$
$m \in_R M_{pk};$ (ab hier A_{koll} eingesetzt)
$y := hash(pk, m);$
$m^* \leftarrow A_{urbild}(pk, y).\big)$

$= \mathrm{P}\big(\sout{\boldsymbol{m,}}\ m^* \in M_{pk} \wedge m \neq m^* \wedge hash(pk, m^*) \boldsymbol{= y}::$
$pk \leftarrow gen(I, \bullet);$
$m \in_R M_{pk};$
$y := hash(pk, m);$
$m^* \leftarrow A_{urbild}(pk, y)\big).$

- Dies ist $P_{urbild}(I)$, außer noch „$m \neq m^*$“ verlangt.
- Laut Beweisidee wollen wir, um dies rauszutrennen, über m mitteln, aber es wird in der Mitte gewählt

  $\Rightarrow$ Regeln zur Reihenfolgevertauschung in solchen W'keiten nötig.
- Dazu auf elementarere Definition gehen:

  Allgemein sei:

$$\mathrm{P}(pred(x, y) :: x \leftarrow alg_1(c); y \leftarrow alg_2(c, x)) = \sum_{x,y} \mathrm{P}_{alg_1(c)}(x)\ \mathrm{P}_{alg_2(c,x)}(y)\ 1_{pred(x,y)}$$

$$= \sum_{x,y:pred(x,y)} \mathrm{P}_{alg_1(c)}(x)\ \mathrm{P}_{alg_2(c,x)}(y),$$

  wobei

$$\mathrm{P}_{alg(in)}(out) = \mathrm{P}(out^* = out :: out^* \leftarrow alg(in).$$

  (≈ Kompositionstheorem/-definition für probabilistische Algorithmen)

Also

$$P_{koll} = \sum_{pk,m,y,m^*} P_{gen(l)}(pk)\ \frac{1}{|M_{pk}|}\ 1_{y=hash(pk,m)}\ P_{A_{urbild}(pk,y)}(m^*)\ 1_{m^*\in M_{pk}}\ 1_{m^*\neq m}\ 1_{y=hash(pk,m^*)}.$$

(P_{urbild} : Dasselbe ohne Term $1_{m^*\neq m}$)

Jetzt Wahl von m absondern (formal: alle Terme mit m soweit möglich nach hinten)

$$P_{koll} = \sum_{pk,y,m^*} P_{gen(l)}(pk)\ \frac{1}{|M_{pk}|}\ P_{A_{urbild}(pk,y)}(m^*)\ 1_{y=hash(pk,m^*)}\ 1_{m^*\in M_{pk}}\ \Big(\sum_{m} 1_{y=hash(pk,m)}\ 1_{m^*\neq m}\Big)$$

$$= \sum_{pk,y,m^*} P_{gen(l)}(pk)\ \frac{1}{|M_{pk}|}\ P_{A_{urbild}(pk,y)}(m^*)\ 1_{y=hash(pk,m^*)}\ 1_{m^*\in M_{pk}}\ \big(|hash_{pk}^{-1}(y)| - 1\big)$$

$$\geq \sum_{pk,y,m^*} P_{gen(l)}(pk)\ \frac{1}{|M_{pk}|}\ P_{A_{urbild}(pk,y)}(m^*)\ 1_{y=hash(pk,m^*)}\ 1_{m^*\in M_{pk}}\ \big(\mathbf{1/2}|hash_{pk}^{-1}(y)|\big)$$

$$= \mathbf{1/2} \sum_{pk,y,m^*} P_{gen(l)}(pk)\ \frac{1}{|M_{pk}|}\ P_{A_{urbild}(pk,y)}(m^*)\ 1_{y=hash(pk,m^*)}\ 1_{m^*\in M_{pk}}\ |hash_{pk}^{-1}(y)|.$$

Wenn man P_{urbild} analog umformt, bekommt man diesen Term ohne „1/2“. Also

$$P_{koll}(l) \geq 1/2 P_{urbild}(l).$$

Dies beweist den Hilfssatz und somit den Satz. □

Anm.: Man kann A_{koll} wiederholen und so fast sicher Kollision finden.

6.5.2 Beweisbar kollisionsfreie Hashfunktionen

(Nach [ChHP_92] und [Damg_88].)

So sicher wie Diskreter-Log-Annahme in Untergruppe primer Ordnung.

A. Für relativ kurze Nachrichten

- **Schlüsselgenerierung**:

  $$pk = (p, q, g, h)$$

  - wobei $q \mid (p-1)$ und g Generator von $G_q \leq \mathbb{Z}_p^*$ (wie in Kap. 5.5.3B).
  - q nicht viel kleiner als p
  - h zweiter zufälliger Generator von G_q.
- **Nachrichtenraum:**

  $$M_{pk} = \mathbb{Z}_q^2$$

  d.h. $m = (x, y)$ und x, y sind mögliche Exponenten.
- **Hashen** von $m = (x, y)$

  $$hash(pk, m) = \mathbf{g^x h^y} \bmod p.$$

Verkürzend falls $|p|_2 < 2|q|_2$.

Gleichmäßig verkürzend: Zähle Urbilder eines $i \in G_q$:

Da g Generator, existieren α, β mit

$$\boldsymbol{h = g^\alpha,\ i = g^\beta.}$$

Also

$$g^x h^y \equiv i \bmod p$$

$$\Leftrightarrow \quad g^x g^{\alpha y} \equiv g^\beta \bmod p$$

$$\Leftrightarrow \quad x + \alpha y \equiv \beta \bmod q.$$

Genau q Lösungen: Wähle erst y, dann x.

Kollisionsfreiheit:

Hauptschritt: Kollision → diskreter Logarithmus:

$$g^x h^y = g^{x^*} h^{y^*}$$

$$\Leftrightarrow \quad g^{x-x^*} = h^{y^*-y}$$

$$\Leftrightarrow \underline{?} \quad h = g^{(x-x^*)(y^*-y)^{-1}}.$$

Beide Seiten mit $(y^*-y)^{-1}$ (mod q) potenziert. Da $\mathbb{Z}_q$ Körper, geht das, **außer wenn $y^* = y$.**

Hilfsbeh.: Wenn Kollision, dann $y^* \neq y$.

Gegeben: $(x, y) = m \neq m^* = (x^*, y^*)$.

Wenn dabei $y = y^*$ wäre, dann

$$g^x h^y = g^{x^*} h^y \text{ (in } G_q\text{)},$$

also

$$g^x = g^{x^*} \text{ (in } G_q\text{)},$$

also $x \equiv x^* \bmod q$. Widerspruch zu $m \neq m^*$.

Also Algorithmus $A_{diskrLog}$

Auf Eingabe (p, q, g, h):

$(m, m^*) \leftarrow A_{koll}(p, q, g, h)$;

Zerlege m in (x, y) und m^* in (x^*, y^*) (wenn möglich);

Falls $y \not\equiv y^* \bmod q$, setze

$$z := (x - x^*)(y^* - y)^{-1}.$$

Offensichtlich prob. poly, und Erfolgsw'keit = der von A_{koll}. (nach obigem)

D.h. wenn *hash* nicht kollisionsfrei wäre, wäre Diskr-Log-Annahme gebrochen. □

Anm.: Es geht auch mit Funktionen

$hash(x_1, \ldots, x_n) = g_1^{x_1} \ldots g_n^{x_n}$.

Beweis etwas komplizierter [ChHP_92] oder [Pfit8_96].

B. Für lange Nachrichten

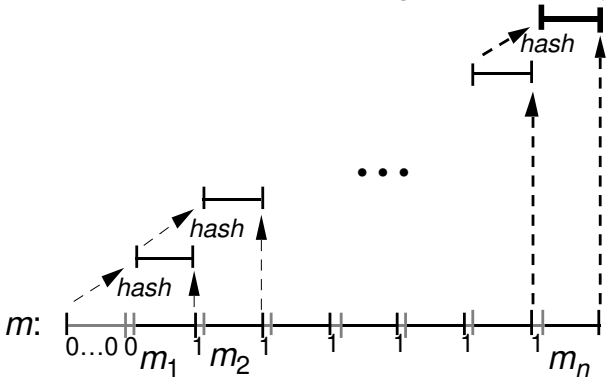
Gegeben

- Familie von Hashfunktionen (*gen*, *hash*, $\mathcal{M}$)
- Funktionen *len_in*, *len_out* (polynomial) so daß $\forall pk \in [gen(l, \bullet)]$:
  - $M_{pk} \supseteq \{0, 1\}^{len_in(l)}$
  - $hash(M_{pk}) \subseteq \{0, 1\}^{len_out(l)}$
  - und $len_out(l) < len_in(l) - 1$.

Konstruktion einer Familie für beliebig lange Nachrichten:

- ***gen*** bleibt
- $\boldsymbol{M_{pk}} = (\{0, 1\}^{len_in(l) - len_out(l) - 1})^*$ für alle *pk*.
- ***hash**** wird so konstruiert:

Ergebnis: $hash^*(m)$



D.h.:

0. Zerlege m in Blöcke der Länge $len_in(l) - len_out(l) - 1$.
   (Nach Voraussetzung > 0).
   Sei $\boldsymbol{n}$ die Blockanzahl, $\boldsymbol{(m_1, \ldots, m_n)}$ die Blöcke.
1. $\boldsymbol{y_1} := hash(pk, 0^{len_out(l)+1} \,||\, m_1)$
   (|| für Konkatenation).
2. Bilde für $i = 2, \ldots, n$:
   $\boldsymbol{y_i} := hash(pk, y_{i-1} \,||\, 1 \,||\, m_i)$
3. **Gib $\boldsymbol{y_n}$ aus**.

Offensichtlich polynomial.

Kollisionsfrei: Annahme: Es gäbe $\boldsymbol{A^*_{koll}}$, das Kollisionen für *hash** zu oft findet.

Konstruiere A_{koll} so: Auf Eingabe *l*, *pk*

$(m, m^*) \leftarrow A^*_{koll}(l, pk)$;

Prüfe, ob Kollision;

Wenn ja, suche *hash*-Kollision daraus:

- Seien $(m_1, \ldots, m_n)$ und $(m^*_1, \ldots, m^*_{n^*})$ die Blöcke und $(y_1, \ldots, y_n)$ und $(y^*_1, \ldots, y^*_{n^*})$ die Zwischenergebnisse.
  Suche nun von oben (= hinten) den ersten Unterschied:
- Es gilt $\boldsymbol{y_n = y^*_{n^*}}$.
  Falls $n, n^* > 1$ und
  $(y_{n-1} \,||\, 1 \,||\, m_n) \neq (y^*_{n^*-1} \,||\, 1 \,||\, m_{n^*})$,
  Kollision gefunden.
- Sonst $\boldsymbol{y_{n-1} = y^*_{n^*-1}}$.
  Genauso …
- Wiederhole, bis Kollision oder ein Nachrichtenanfang erreicht.
- Falls jetzt $n = n^*$ und
  $(0^{len_out(l)+1} \,||\, m_1) = (0^{len_out(l)+1} \,||\, m^*_1)$,

wäre $m = m^*$. Widerspruch, daß (m, m^*) Kollision.

- Also oBdA $n < n^*$. Dann

  $(0^{len_out(l)} \,||\, \mathbf{0} \,||\, m_1) \neq (y^*_{n^*-n} \,||\, \mathbf{1} \,||\, m_{n^*-n})$

  wegen mittlerem Bit.

Also hat A_{koll} **selbe Erfolgsw'keit** wie A^*_{koll}.

Widerspruch zur Kollisionsfreiheit von *hash*. □

Anm.: Einschieben von Nullen und Einsen dient „**Postfixfreiheit**", d.h. : Andernfalls könnte Angreifer Anfang von Nachricht unentdeckt wegwerfen.

Einweg ??

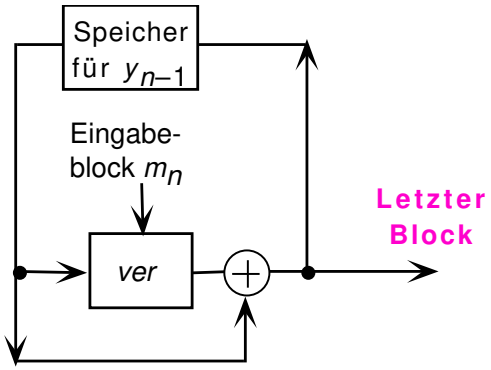
- Jedes Urbild m^* von y bzgl. *hash** liefert insbesondere ein direktes Urbild von *hash*, nämlich $(y_{n-1} \,||\, 1 \,||\, m_n)$.
- Aber W'verteilung von y in Annahme und Behauptung verschieden …

(Jetzt reicht's ...)

6.5.3 Hashfunktionen aus Blockchiffren

Jetzt kein Schlüssel für Blockchiffre vorhanden (oder man müßte den symmetrischen veröffentlichen).

Also Nachrichtenblöcke als Schlüssel, z.B.



- Speicher mit festem Initialisierungswert $y_0 = IV$ vorbelegt.
- $y_i := ver(m_i, y_{i-1}) \oplus y_{i-1}$.

Sicherheit gegen einfache Angriffe?

\+ **Postfixangriff?** D.h. kann man vordere Blöcke einfach weglassen?

Nur wenn zufällig ein $y_i = IV$ für $i \neq 0$. (Dann $hash(m_{i+1}, \ldots) = hash(m_1, \ldots)$.

(Wäre nur Problem, wenn *IV* frei gewählt und mit Nachricht mitgeschickt.)

\+ **Rückwärts rechnen?** Wenn unterster Pfeil und ⊕ nicht wären, dann so:

Wähle y beliebig und m, m^* beliebig. Entschlüssele mit m_n zu y_n bzw. m_{n^*} zu y_{n^*}. usw. (Scheitert aber auch an festem IV.)

- **Geburtstagsangriff!**

  Wähle viele zufällige Nachrichten m.

  Errechne Hashwerte und speichere Paare $(hash(m), m)$ in sortierter Liste.

  Warte, bis erste Kollision gefunden.

  Bei Ausgabelänge b im Mittel nach ca. $\mathbf{2^{b/2}}$ Versuchen. Bei DES nur 2^{32}!

Nachteil solcher Konstruktionen: Effizienz in Software z.B. bei DES wesentlich geringer als zur Verschlüsselung, da Tabellenerzeugung schlüsselabhängig.

Genereller Name: Iterierte Hashfunktionen für dies und 6.5.2B:

- $y_0 = IV$.
- Schritte der Form

  $$y_i := hash(y_{i-1}, m_i).$$

- Schema zur Fixierung der Nachrichtenlänge.

6.5.4 Direkt konstruierte Hashfunktion

Schlüssellos.

Vor allem MD4-Varianten [Rive2_91]

- MD4 selbst aber **gebrochen!** [Dobb_96]
- MD5 auch nicht mehr verwenden.

Übrig:

- SHA-1 (Update von SHA = Secure Hash Algorithm); [FIPS 180-1_97]
  (vor allem zum DSS-Signatursystem)
- RIPEMD-160 [DoBP_96].

Beide mit 160-bit Ausgabe, d.h. Geburtstagsangriff bräuchte 2^{80} Versuche.

Sehr chaotisch, vgl. Menezes o. Schneier.

6.5.5 Literatur

Zitiert:

ChHP_92 D. Chaum, E. van Heijst, B. Pfitzmann: Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 470-484.

Damg_88 I. B. Damgård: Collision free hash functions and public key signature schemes; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 203-216.

Dobb_96 H. Dobbertin: Cryptanalysis of MD4; Fast Software Encryption, LNCS 1039, Springer-Verlag, Berlin 1996, 53-69.

DoBP_96 H. Dobbertin, A. Bosselaers, B. Preneel: RIPEMD-160: A Strengthened Version of RIPEMD; Fast Software Encryption, LNCS 1039, Springer-Verlag, Berlin 1996, 71-82.

Pfit8_96 B. Pfitzmann: Digital Signature Schemes — General Framework and Fail-Stop Signatures; LNCS 1100, Springer-Verlag, Berlin 1996.

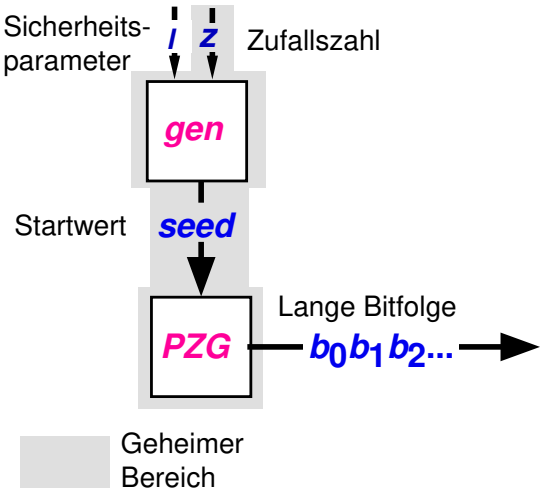
Rive2_91 R. L. Rivest: The MD4 Message Digest Algorithm; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 303-311.

Bücher:

- Theorieteil (Kap. 6.5.1., 6.5.2): keines.
- Praxisteil:
  - Menezes et al. 9.4 (+ Teile von 9.2, 9.3., aber z.T. mit Authentikationscodes vermischt)
  - Schneier: Teile von 18.11, 18.4-9.

6.6 Pseudozufallsgeneratoren

Erinnerung:

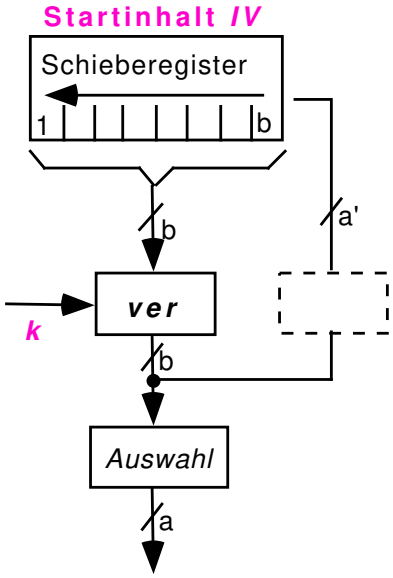


- Manchmal hat *seed* Teil, der öffentlich sein kann.

6.6.1 Konstruktionen aus Blockchiffren

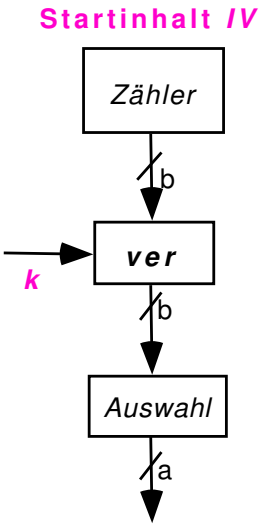
A. OFB

Vgl. Kap. 6.1.3F: OFB war primär Pseudozufallsgenerator:



- ***IV***: Initialisierungsvektor.
- ***seed*** = (*k*, *IV*)
  - *k* geheim,
  - *IV* darf öffentlich sein,
  - *k* wiederverwendbar, wenn *IV* wechselt.

B. Anderes Beispiel

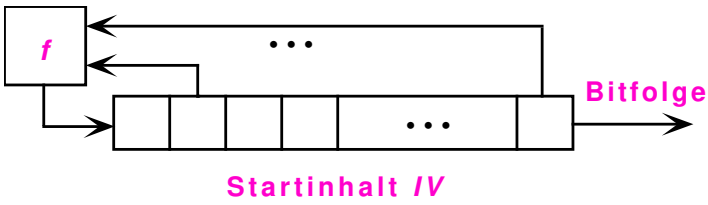


Heuristisches Argument: Schlüsseltexte aufeinanderfolgender Zahlen unabhängig voneinander.

6.6.2 Direkte Konstruktionen

Vor allem rückgekoppelte Schieberegister.

- Englisch LFSR (linear feedback shift register)
- Z.B. so



Startinhalt *IV*

- *seed*: Teile von *f* (und evtl. *IV*)
- Wenn Rückkopplungsfunktion *f* linear (d.h. Exor mehrerer Bits), leicht zu brechen.
- Wenn *f* nichtlinear, oder mehrere solcher Folgen nichtlinear kombiniert, könnte es sichere geben.
- Viel mathematisch nette Theorie zu Periodenlänge u.ä.

Aber: viele Versionen gebrochen!

(Dafür sehr schnell.)

6.6.3 BBS oder x^2-mod-n: Beweisbar sicherer Pseudozufallsgenerator (hier ohne Beweis)

Überblick:

- Aus [BlBS_86] („Blum-Blum-Shub-Generator“)
- So sicher wie Faktorisierungsannahme
- Nicht so effizient wie obige Systeme, aber praktikabel: 1 modulare Multiplikation pro 1-10 Bits.

Für hochsichere Anwendungen zu empfehlen, v.a.

- Generierung von Sitzungsschlüsseln
- oder Zufallszahlen innerhalb Signatursystemen.

Dort ist Geschwindigkeit auch unkritisch.

Schlüsselgenerierung:

1. Wähle

$$p, q$$

prim, zufällig, unabhängig, $|p|_2 \approx |q|_2 = l$ und $p \neq q$ und

$$p \equiv q \equiv 3 \bmod 4.$$

(Einzelheiten Kap. 5.4.3.)

2. $$n := p \cdot q.$$

3. Wähle $s_0 \in_R \mathbb{Z}_n^*$

Biterzeugung:

Für *i*-tes Bit:

$$s_i := s_{i-1}^2 \bmod n,$$

$$b_i := s_i \bmod 2$$

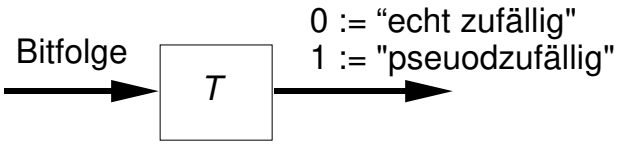
(d.h. letztes Bit).

Variante: Mehr als 1 Bit pro Quadrierung nehmen:

- Bis $\log \log(n)$ noch beweisbar (ca. 10).

6.6.4 Skizze der Sicherheitsdefinition

Betrachte alle polynomiale (Test-)Algorithmen T:



- Man kann nicht verlangen, daß T nie richtig entscheidet. (Raten unter 2 Möglichkeiten!)

$\Rightarrow$ Statt dessen: Bei pseudozufälligen im Mittel *fast genauso* entschieden wie bei echten, d.h.

$$\text{Unterschied} < 1/poly(l).$$

☞ Pseudozufälligkeit ist Eigenschaft einer **Menge von Folgen**, nicht einer einzelnen Folge.

Anm.: Pseudozufallsgeneratoren aus beliebiger Einwegfunktion konstruierbar [ImLL_89].

6.6.5 Literatur

Zitiert:

BIBS_86 L. Blum, M. Blum, M. Shub: A Simple Unpredictable Pseudo-Random Number Generator; SIAM Journal on Computing 15/2 (1986) 364-383.

ImLL_89 R. Impagliazzo, L. A. Levin, M. Luby: Pseudo-random Generation from One-way Functions; 21st Symposium on Theory of Computing (STOC), ACM, New York 1989, 12-24.

Zitiert:

Menezes et al.: Teile von 7.2.2 und 6.3., Kap. 5.5.2.

Schneier: 9.8-9.9, Teile von 16.2-16.4., 17.9.

6.7 Literaturverweise auf weitere Klassen

$\rightarrow$ Spezialvorlesung Sommersemester, z.B.

Weitere Bausteine

- Zero-knowledge Beweise
- Commitments (Festlegschemata)

Kleine nützliche Systeme

- Secret sharing
- Münzwurf (mehrseitig vertraute Zufallszahlen)

Große nützliche Systeme

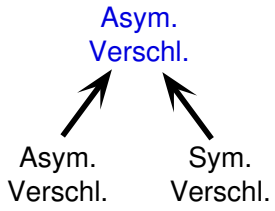
- Wahlen
- Zahlungssysteme

Generelle Übersetzungstechniken

7 Wichtige Kombinationen und Anwenderschnittstelle

Größtenteils Wiederholung, aber bisher auf einzelne Systeme verstreut.

7.1 Hybride Verschlüsselung



- **Schlüsselerzeugung:**

$$gen_{neu} = gen_{asym}$$

- **Verschlüsselung:** Eingabe pk, m:
  - $k \leftarrow gen_{sym}(l)$
  - $c := (ver_{asym}(pk, k), ver_{sym}(k, m))$
- **Entschlüsselung:** Eingabe sk, c:
  - Zerlege c in 2 Teile (c_{asym}, c_{sym}).
  - $k := ent_{asym}(sk, c_{asym})$.
  - $m := ent_{sym}(k, c_{sym})$.

Effizienzverbesserungen

1. **Ersten Block auffüllen:** Meist k viel kürzer als 1 Block m_{asym}, z.B. 128 Bits gegen 1024.

   $\Rightarrow$ Setze $c_{asym} = (ver_{asym}(pk, (k, m^*))$

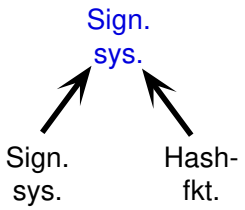
   wo z.B. m^* erste 896 Bits der Nachricht.

2. **Für Multicast:** (Dann nicht 1.!): Verwende für alle Empfänger selbes k und c_{sym}. Nur c_{asym} verschieden, bzw. alle Teile davorgehängt.

Sicherheit

Vermutlich beweisbar, wenn beide Komponenten sicher im strengen Sinn, aber meines Wissens kein Beweis bekannt.

7.2 Hashen vor Signieren



- **Schlüsselerzeugung:**
  - $hk \leftarrow gen_{hash}(l)$; (fällt oft weg)
  - $(sk, pk) \leftarrow gen_{sig}(l)$;
  - $sk_{neu} = (sk, hk)$; $pk_{neu} = (pk, hk)$.
- **Signieren:** Eingabe sk_{neu}, m:

  $m^* := hash(hk, m)$;

  $sig := sign(sk, m^*)$.
- **Testen:** Eingabe pk_{neu}, sig zu m:

  $m^* := hash(hk, m)$;

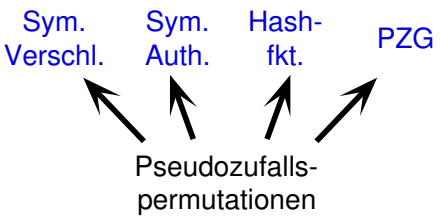
  $test(sk, m^*, sig)$.

Sicherheit

Bewiesen, wenn beide Komponenten sicher im strengen Sinn [Damg_88].

7.3 Betriebsarten von Blockchiffren

Eigentlich



- Pseudozufallspermutationen(familien): Ähnlich definierbar wie PZG: Poly. Angreifer kann, geg. einige Paare $(x, f(x))$, nicht unterscheiden, ob f echt zufällig war oder eine Permutation der Familie.
- Wenn überhaupt beweisbar, dann damit.
- In Praxis aber Blockchiffren, die eher nur für Verschlüsselung entworfen wurden.
- Kryptographisch beweisbare PZPerm. gibt's [GoGM_86], aber für sym. Systeme zu ineffizient.

7.4 Einsatz von Pseudozufallsgeneratoren

Überall statt echter Zufallszahlen, v.a.:

- Als Pseudo-one-time-Pad.
- In Authentikationscodes.
- Zufallszahlerzeugung für probabilistische Verschlüsselung (z' in Kap. 2.1.2).
- Zufallszahlerzeugung für indeterministische Signaturen, z.B. z bei ElGamal.
- Erzeugung vieler symmetrischer Schlüssel für hybride Verschlüsselung.

Sicherheit

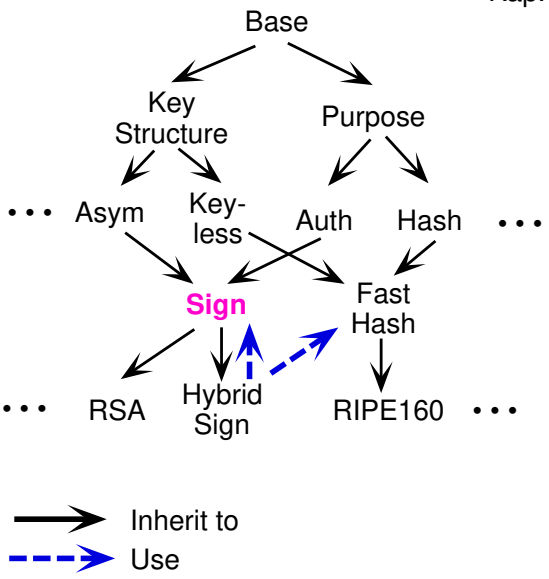
- Aus informationstheoretischer Sicherheit wird kryptographische.
- Kryptographische bleibt erhalten.

(Für jedes Einzelsystem beweisbar, als Gesamtbeweis nicht bekannt.)

7.5 Anwenderschnittstelle

- All diese Kombinationen sollten für Anwender **unsichtbar** sein.

  (Z.B. Erwähnung von Hashfunktionen bei Unterschriften für Electronic Commerce unnötig.)
- Objektorientierte Kryptobibliothek z.B. so aufgebaut (Bsp. aus CryptoManager++ [BaBl_95])
  - Teilbaum für Signaturen mit Hashfunktion



- Anwender verwendet Objekt vom Typ SIGN-Objekt (≈ Schlüssel)
- Aus Key-structure geerbt: *gen*, *load_key*, *store_key* (persistent speichern)
- Aus Auth geerbt: *auth*, *test*
- Z.B. aus Voreinstellungen: Default-SIGN-Generierung ergibt Hybrid-Sign mit RSA und RIPE160.

7.6 Literatur: Vergessen → Kap. 8

TEIL II Kryptographie in der Praxis

8 Nötig: Zufallszahlen, Schlüsselverteilung

Vgl. Bilder in Kap. 2: Fast alle kryptographischen Systeme brauchen:

- **Zufallszahl** (selbst wenn meist PZG verwendet: wenigstens für *seed*-Generierung).
- **Verschieden sichere Kanäle** zum Schlüsselaustausch.

(• Geräte, die die krypt. Algorithmen sicher genau nach Benutzerwunsch ausführen → Kap. 12.)

8.1 Erzeugung echter Zufallszahlen

Beachte 1: Muß für asym. Systeme immer beim Besitzer des geheimen Schlüssels erfolgen, sonst kennt noch jemand geheimen Schlüssel!

Beachte 2: In vielen Produkten schlecht gelöst! (Z.B. frühe Version von SSL (in Netscape) so gebrochen.)

Im folgenden:

1. Mögliche Quellen
2. Umgang: Kaum eine einzeln und unverändert zu nutzen ...

(3. Fast alles so langsam, daß man es wirklich nur für PZG-Startwert verwendet.)

8.1.1 Mögliche Quellen

- **Physikalische Phänomene**

  Z.B.
  - Rauschdioden
  - Turbulenzen im Lüfter

  [FaMC_85, Agne_88, GlVi_88, DaIF_94].

  \+ Benutzer nicht gestört

  \+ Manche wohl beweisbar zufällig (Quantenprozesse)

  – Spezialhardware nötig (auf PCs und Smartcards nicht üblich)

  – Hardware kann unbemerkt kaputt gehen

  ◦ Keine von Angreifer beobachtbare Quelle nehmen.

  Wenig Literatur über Qualität.

- **Direkt vom Benutzer**
  - Würfeln lassen.
    \+ Wohl am besten von allem!
    – Manche werden es nicht tun
  - "Zufällig" tippen lassen.
    \+ Einfacher zu tun
    – Zufälligkeit geringer
- **Indirekt vom Benutzer**
  - Meist niederwertige Bits von Zeitabständen zwischen Tastendrücken.
    \+ Benutzer nicht gestört
    – Zufälligkeit unklar
    – Je nach Betriebssystem Abtastung nur in festen Zeitabständen!

¿ **Aus Rechner selbst?**

– Systemparameter wie aktuelle Uhrzeit, Prozessornummer nicht zufällig!

◦ Höchstens indirekt aus Last abgeleitete

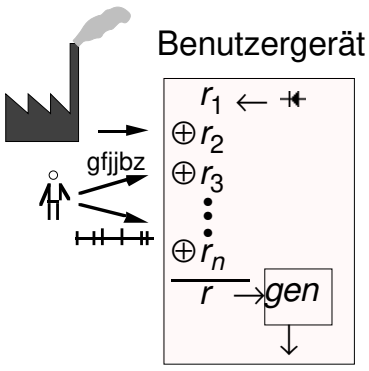
⇒ Eigentlich wieder "indirekt vom Benutzer".

◦ Jetzt von mehreren Benutzern; aufpassen, daß nicht vom Angreifer beobachtbar.

8.1.2 Verwendung der Quellen

1. **Einzelne Quellen komprimieren**. Selbst echt zufällige Quellen liefern meist nicht gleichverteilte Bits. Komprimierung sichert das weitestgehend.
2. **EXOR möglichst vieler Teile**

   Z.B.



Satz: Solange ein r_i zufällig und unabhängig von den anderen ist, ist r zufällig.

Beweis: Vgl. One-time-Pad: Selbst wenn Angreifer alle anderen r_j kennt, ist jedes r mit gleicher W'keit möglich.

(Sei etwa $r^* = \bigoplus_{j \neq i} r_j$, dann ergibt sich r genau wenn

$$r_i = r \oplus r^*,$$

und die r_i sind gleichverteilt.)

8.2 Schlüsselverteilung

Erinnerung: Nötig ist

- Für symmetrische Verschlüsselung und Authentikation:

  Geheimer und integrer Austausch von *k*.
- Für asymmetrische Verschlüsselung:

  Integrer Austausch von *pk*.
- Für Signatursysteme:

  Integrer und konsistenter Austausch von *pk*.

Weiterer Unterschied zwischen Verschlüsselung und Authentikation:

- Wenn Signatursystem primär wegen Disputmöglichkeit (z.B. elektronische Bestellungen), werden Probleme durch Inkonsistenz auf jeden Fall entdeckt.
- In dem Fall genügt es auch, wenn jemand haftet.

(Bei Datenschutzverletzungen nicht.)

In Praxis derzeit trotzdem meist gleich behandelt.

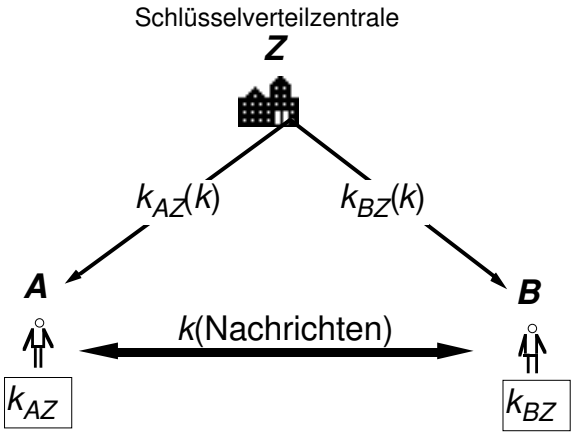
8.2.1 Symmetrische Schlüssel

Möglichkeiten für Austausch:

- **Persönlich** (sinnvoll in kleinen Gruppen für informationstheoretisch sichere Systeme).
- **Per Bote** (früher üblich in Militär, Banken).
- **Mit „Masterschlüssel“**:
  - Diesen 1x persönlich oder per Bote austauschen
  - und nur für Austausch von „Sitzungsschlüsseln“ verwenden.

  Vor allem, wenn man
  - Sicherheit des Verfahrens gegen Angriffe mit vielen Klartext-Schlüsseltext-Paaren
  - oder Aufbewahrung der „Sitzungsschlüssel“ nicht ganz vertraut.

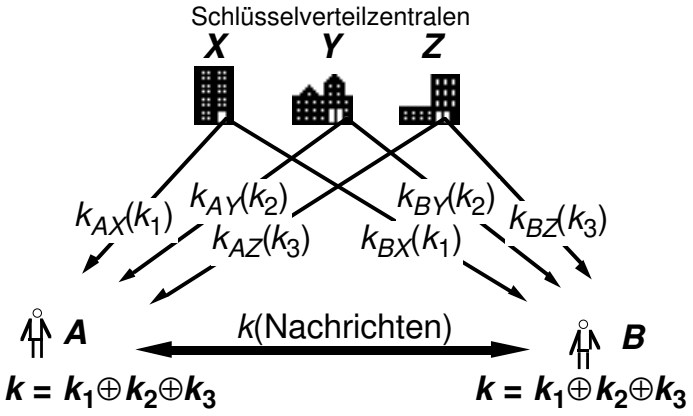
- **Über Zentralen**
  - **Eine Zentrale:**



(wobei $k(m)$ Abkürzung für $ver(k, m)$)

- \+ Jeder Teilnehmer braucht vorweg nur ein k_{AZ}
- – Riskant: *Z* kann alles lesen bzw. authentisieren!
- ⇒ Nicht für offene Systeme, aber z.B. für interne Kommunikation kleiner Firma.

  - **Mehrere Zentralen:**



  - Schlüssel ist zumindest geheim, solange wenigstens eine Zentrale korrekt arbeitet.

    (Evtl. können *A* und *B* nicht kommunizieren.)
- **Mit asymmetrischem System:**
  - Z.B. nachrichtenweise mittels hybrider Verschlüsselung.
  - Oder pro „Sitzung“ auf irgendeiner Ebene.

  Heutzutage am häufigsten.

8.2.2 Asymmetrische Schlüssel

Begriff auch **„Public-key infrastructure“** (PKI). Derzeit viel diskutiert, da Fehlen wesentliches Hindernis für Electronic Commerce.

Erinnerung: Für Verschlüsselung und Signaturen eigentlich recht verschiedene Anforderungen. Im folgenden bei einzelnen Maßnahmen skizziert.

A. Möglichkeiten zur Verteilung grob

- **Persönlich.**
- **Papierverzeichnisse**, z.B. Telefonbuch (schwer selektiv fälschbar!)
- **Über Zentralen digital**
- **Über Ketten von Bekannten digital**

Technisch hat alles viele ähnliche Aspekte:

B. Überblick Aspekte der Verteilung

1. **Prüfung der Zuordnung Person-Schlüssel** außerhalb des Systems.

   („Identitätsprüfung“, „Registrierung“.)

2. Evtl. **Bestätigung** der Zuordnung für Dritte.

   („Schlüsselzertifikate“.)

3. Ggf. **Verteilung** solcher Zertifikate.

   (Oft „directory“, „key server“.)

4. **Prüfung durch Empfänger**, ob ihm Zuordnung genügend geprüft / bestätigt erscheint.

   („Vertrauensmodell“, „Trust model“.)

C. Externe Prüfung der Zuordnung

Sei *B* prüfende Person, *pk* der Schlüssel, *A* die behauptete Identität des Besitzers.

α. Möglichkeiten:

1. **Überreichen** von *pk* auf Papier oder auf Diskette: Reicht wenn
   - Person *B* Person *A* persönlich kennt und
   - *pk* nur für
     - Verschlüsselung oder
     - (vom Ziel her symmetrische) Authentikation von *A* an *B* verwendet wird.

   Dann beste Möglichkeit!

2. **Eigenhändige Unterschrift** (auf Papier).

   Als Ergänzung zu 1., wenn
   - *pk* Schlüssel von Signatursystem und *B* ihn evtl. vor Gericht verwenden will.

     Anm.: Müßte in freier Beweiswürdigung anerkannt werden, auch wenn nicht spezifisch in Gesetz, sofern *A* frei und informiert entscheiden konnte.

   - oder *B* Zertifikat über diesen Schlüssel ausstellen will: Vorsorge für Dispute, d.h.
     - *A* sagt: *pk* ist nicht mein Schlüssel.
     - *B* sagt: *A* hat ihn aber überreicht.

     Wenn *B* keine Unterschrift von *A* hat, sollte *A* recht bekommen.

   Genauer Text: ähnlich Zertifikaten.

3. **Ausweis prüfen:** Als Ergänzung zu 1. (und meist 2.), wenn *B* Person *A* nicht persönlich kennt. V.a. als „Zertifikat“ vom Staat, wie *A*'s Unterschrift aussieht.

4. **Schwächere Formen:**

   a) **Telefonisch:**
   - Nicht schlecht statt 1., wenn man Stimme kennt.

   Andernfalls:
   - Ohne Rückruf sinnlos.
   - Mit Rückruf kann es jeder aus *A*'s Haushalt bzw. Büro sein.
   - Auch fehlt *B* Unterschrift von *A* für Dispute.

(b) **Email, Fax, WWW: Abzuraten.** Auf keinen Fall rechtskräftig, und selbst bei „Email-Rückruf“ relativ leicht zu fälschen. Deswegen will man ja Signaturen …**)**

5. **Mit digitaler Unterschrift von *A*:**
   - Natürlich nicht beim 1. Mal.
   - Für Austausch zusätzlicher Schlüsseln (z.B. für Verschlüsselung oder Benutzung auf weniger sicherem Rechner) ok.
6. **Sonderfälle:** Wenn *A* sehr bekannt (z.B. Zeitschrift c't, „die Bundesregierung“): Verbreitung über passende Medien.

β. Effizienzverbesserung

pk beliebig austauschen und nur ***hash*(*pk*) wie oben behandeln**.

- D.h. *hash*(*pk*) auf Papier unterschreiben, telefonisch prüfen usw.
- Welche Hashfunktion muß klar sein bzw. mit unterschrieben werden.
- Nötige Eigenschaft: „universal one-way“, vgl. Kap. 6.5.1. Einweg und kollisionsfrei ist dazu hinreichend (müßte man im Prinzip für alle Anwendungen nachweisen).

In PGP heißt *hash*(*pk*) „**Fingerprint**“. (Aber sehr mehrdeutiges Wort)

γ. Was, wenn *A* einen Schlüssel *pk* von *C* verteilt?

- **Für bisherige Zwecke egal!**
  - *A* kann nicht bzgl. *pk* signieren bzw. entschlüsseln.
  - *A* wird haftbar für *C*'s Signaturen.
- Wenn man es für andere Zwecke doch **vermeiden** will: Unterschrift mit *pk* unter Text: „Mein Name ist *A*“.

  (In Praxis oft verlangt.)

Abstrakter:

- String „*A*“ und „*pk*“ sind letztlich 2 Namen oder Pseudonyme. Bisher signiert Besitzer von „*A*“: Ich will auch Namen „*pk*“ haben.

  Davon liegt in seiner Zuständigkeit aber nur
  - Geheime Post an mich ist unter *pk* ok bzw.
  - Ich bin für Signaturen mit *pk* verantwortlich.

  Deswegen stört Falschaussage nicht.

δ. Inhalt der „Zuordnung“

Die Aussage von *A* (nach jeder der obigen Möglichkeiten) sollte folgendes enthalten:

- **Genaues System**, zu dem *pk* gehört, inkl. Nachrichtenformat.

  (Keine Sicherheitsaussage für ein System gilt, wenn *pk* evtl. noch anders verwendet, z.B. Kollision von Nachricht aus System 1 mit Nachricht aus System 2 … .)
- **Gültigkeitsdauer**, falls *A* Schlüssel nicht beliebig lange vertraut.
- **Sonstige Beschränkungen**, z.B.
  - Keine Haftung für Signaturen.

    V.a. in Systemen, wo Schlüssel für Verschlüsselung *und* Signaturen verwendet, oder wenn Authentikation primär symmetrisch gedacht.
  - Schlüssel unsicher aufbewahrt, schickt bitte nichts allzu Geheimes

- Haftung nur in bestimmten Anwendungen
- Haftung bis Obergrenze (sofern Empfänger das verifizieren können)

Vor allem als **Schutz für *A* gegen Schlüsseldiebstahl** — heutzutage kaum auszuschließen.

⇒ Sollte von Empfängern und Gerichten respektiert werden, derzeit meist unklar.

D. (Schlüssel-)Zertifikate

Begriff „**Zertifikat**“ uneinheitlich verwendet:

- Hier gemeint: Digitale Bestätigung einer Zuordnung Person-Schlüssel.

  („**Schlüsselzertifikat**“, „key certificate“ klarer.)

- **Nicht verwechseln** mit Beglaubigungen, Empfehlungsschreiben, die Zuordnung Person-Eigenschaft oder Person-Recht enthalten.
  - „**Attributzertifikat**“ nennen („attribute certificate“, auch „credential“)
  - Es gibt auch Mischung: Aussage über Person-Schlüssel-Recht. Ungeschickt
    a) **Semantik unklar**, z.B.: Haben Person *A* und Schlüssel *pk* allein das Recht oder nur zusammen? Haftet *A* für *pk* überhaupt / auch in anderen Zusammenhängen?
    b) **Datenschutz**: Zumindest Hauptschlüssel sollte benutzbar sein, ohne Attribute vorzuzeigen.

Zertifikat genauer: Person *C* unterschreibt:

- „*A* sagte, daß sie *pk* verwenden will, und zwar in folgendem Umfang (System, Datum, Beschränkungen).“
- Falls *C* nicht 100% sicher ist, daß es *A* war, sollte *C* das klarstellen:

  „Eine Person, die einen Personalausweis auf *A* dabei hatte und entsprechend signieren konnte, sagte …“.

Dabei:

- Oft **Text** nicht so explizit, sondern „**Policy**“ (**Geschäftsbedingung**) von *C*. (Manchmal steht nicht mal die fest :-()

  Der Text des einzelnen Zertifikats ist nur noch „*A*, *pk*, *Datum*, …“

  Ungeschickt, andere rechtliche Dokumente sind auch ausgeschrieben. (Und das würde viel Chaos mit Attributzertifikaten vermeiden.)
- Das „***A* sagte, daß** …“ würden viele Autoren weglassen, aber nur *A* kann entscheiden, wofür sie haften will, also gehört es eigentlich dazu.

- **Haftung von *C***. Wenn *C* keine Einschränkung macht, verläßt sich Empfänger *B* des Zertifikats darauf. Falls *A* dann nicht haftet (z.B. weil *C* keine Unterschrift von *A* hatte), müßte *C* haften.

  Juristisch noch nicht klar.

Anm.: Falls man in großem Stil zertifiziert (Zentrale), ist Geheimhaltung des eigenen Signierschlüssels äußerst kritisch!

Kann derzeit sicher nicht absolut garantiert werden

⇒ auch Zertifikate von sog. „trusted third parties“ kann man nicht ganz vertrauen, selbst *wenn* man der Organisation an sich vertrauen würde.

Begriff: CA (certification authority).

E. Verteilung der Zertifikate

Möglichkeiten:

- **Via *A***: *C* gibt Zertifikat *A*, und *A* gibt es Kommunikationspartnern *B*.

  Vor allem
  - bei Signaturen (Nachricht geht sowieso von *A* an Partner)
  - wenn *A* wenig Zertifikate hat, also Auswahl nicht von *B* abhängt.

  Sonst auch Aushandlung *A* — *B*.
- **Digitale Verzeichnisse** („Directories“, „key server“).
- ***B* kann lokal sammeln**.
- **Papierverzeichnisse (dann meist ohne digitale Signatur):**

  Vorteil: Schwer selektiv zu fälschen. *A* kann z.B. prüfen, ob es falsche Zertifikate zu ihr gibt.

Anm.: Sicherheit von Directories **unkritisch**. Empfänger vertraut den Zertifikaten bestimmter Leute, und Directory kann diese nicht fälschen.

Da hier Online-Zugriff, sollten Rechner von Zertifizierungsstellen getrennt sein.

F. Lokale Prüfung bei *B*

Letztlich muß *B* entscheiden, ob

1. er der **Integrität** von *pk* sicher ist, d.h. daß *A* diesen Schlüssel möchte. (Bei Verschlüsselung und sym. gedachter Authentikation.)
2. **bzw.** er der **Konsistenz** sicher ist bzw. wenigstens der Haftung. (Bei Signaturen.)

   Genauer: ein Gericht wird auch glauben, daß *A* diesen Schlüssel wollte, bzw. jemand anders haften für ein falsches Zertifikat haften lassen.

Ersteres ist Frage von Vertrauen, letzteres von ausspezifizierter Haftung, also

- bei 1. Zertifikate **persönlicher Bekannter** günstig (sofern man gemeinsame hat)
- bei 2. eher von **staatlichen Stellen** (sofern sie haften).

Verbesserung des Vertrauens:

- Verlangen mehrerer Zertifikate (unter selben Schlüssel; viel einfacher als bei symmetrischem System.

Zertifikatsketten: Falls *A* und *B* nicht direkt gemeinsame Bekannte haben, oder nicht mit der gleichen Zentrale arbeiten, mehrere Schritte von Zertifikaten:

Z.B:

- *B* kennt Schlüssel von C_B,
- C_B kennt Schlüssel von C_A
- C_A kennt Schlüssel von *A*.

C_B könnte nun Schlüssel von C_A zertifizieren; C_A den von *A*.

Reicht das *B*?

- Sowohl C_B als auch C_A könnten ihn betrügen, d.h. er muß beiden vertrauen bzw. beide müssen haften (und Geld haben …)
- Falls er C_A nicht kennt, müßte ihm jemand (z.B. C_B) sagen, daß er C_A vertrauen kann.

Bzgl. Vertrauen recht allgemein so darstellbar (nach [Maur1_96] — mit Haftung müßte es ähnlich gehen, gibt's aber noch nicht):

- $\boldsymbol{Aut_{B,X}}$ heiße, daß *B* einen (Signier-) Schlüssel von *X* für authentisch hält.
- $\boldsymbol{Cert_{X,Y}}$ heiße, daß *B* ein Zertifikat von *X* über Schlüssel von *Y* hat.
- $\boldsymbol{Trust_{B,X,1}}$ heiße, daß *B* vertraut, daß *X* vor Zertifizierung Identität ordentlich prüft.
- $\boldsymbol{Rec_{X,Y,1}}$ („recommendation": Empfehlung) heiße, daß *B* ein **Attributzertifikat** von *X* hat, indem *X* sagt, daß er *Y* vertraut, daß *Y* ordentlich zertifiziert.
- Analog induktiv definiert
  - $\boldsymbol{Trust_{B,X,i}}$: *B* vertraut, daß *X* Empfehlungen vom Typ *i*–1 ordentlich ausstellt.
  - $\boldsymbol{Rec_{X,Y,i}}$: Entsprechendes Zertifikat.

Anm. 1: Man kann *Aut* als Spezialfall von *Cert* und *Trust* als Spezialfall von *Rec* sehen, und technisch noch *Cert* als $Rec_{...0}$ darstellen.

Anm. 2:

- $Trust_{...1}$, $Rec_{...1}$: Vertrauen in Sorgfalt.
- $Trust_{...2}$, $Rec_{...2}$: Vertrauen in Menschenkenntnis (bzgl. Sorgfalt)
- Vermutlich versteht kaum jemand im realen Leben noch $Trust_{...3}$, $Rec_{...3}$

  $\Rightarrow$ weglassen oder durch ein absolutes $Rec_{...all}$ ersetzen?

Auswertung nach folgenden Regeln:

- Nehme als initiale $Aut_{B,X}$ die, wo *B* die Zuordnung selbst geprüft hat.
- Nehme als initiale $Trust_{B,X,1}$ die, die *B* von sich aus denkt.

Leite weitere ab:

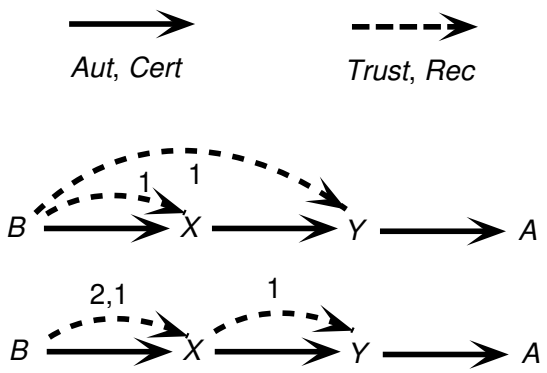
- $Aut_{B,X}$, $Trust_{B,X,1}$, $Cert_{X,Y}$ $\vdash$ $Aut_{B,Y}$
- $Aut_{B,X}$, $Trust_{B,X,i+1}$, $Rec_{X,Y,i}$ $\vdash$ $Trust_{B,Y,i}$.

und sonst nichts.

- Wenn so $Aut_{B,A}$ ableitbar, „vertraut" *B* einem Schlüssel von *A*.

(Art logischer Kalkül ohne Semantik, aber man kann Regeln als Definition von *Trust* deuten.)

Bsp.:



Beidemale läßt sich $Aut_{A,B}$ ableiten.

Erweiterung auf nicht ganz sichere Aussagen: Deutung von Unsicherheiten als W'keiten, z.B. 80%-Trust, 90%-Cert. Hat keinen wirklichen Hintergrund (d.h. Deutung über relative Häufigkeiten), aber erlaubt halbwegs zur Intuition passende Auswertung. (Einzelheiten siehe [Maur1_96].)

In Wirklichkeit nicht ganz allgemein, z.B. manche *Rec* in Realität drücken nur teilweises Vertrauen aus, z.B. „kann in Ort *O* zertifizieren".

Policy: Feste Strategie, wie das Anfangsvertrauen verteilt sein soll. (Z.B. nur in bestimmte Zentralen.) Z.T. mit Strategien für die anderen Schritte (Prüfung der Zuordnung, …) verbunden.

G. Rückruf (Revocation)

- **Nötig, wenn Schlüssel evtl. gestohlen.** *A* stößt an, andere widerrufen Zertifikate.
- **Ziel eher zu viel Rückruf** als zu wenig, da es nur Verfügbarkeit, nicht Integrität und Vertraulichkeit verletzt.
- **Problem wieder Haftung**: Was, wenn *B* noch Schlüssel bzw. Zertifikat benutzt, obwohl Aussteller es zurückgezogen hat?
  - Entweder darf *B* keinen lokalen Cache für Zertifikate haben, dann haben nur noch Verzeichnisse das Problem.
  - Oder Rückrufe müssen an bestimmten Stellen sein, und in bestimmten Abständen muß *B* dort schauen.

Zeitpunkte von Signaturen sowieso nicht feststellbar: Wenn *B* schon lange Schuldschein von *A* hat, und *A*'s Schlüssel wird zurückgezogen, ist nicht nachweisbar, daß der Schuldschein alt ist — *B* könnte auch der Schlüsseldieb sein und rückdatiert haben.

H. Wichtige konkrete System

PGP [Zimm_95]

- Keine feste Strategie zur Prüfung der Zuordnung. (Empfehlungen ähnlich oben.)
- Zertifikate ohne festgelegte Felder für Beschränkungen (man kann aber das Namensfeld verwenden; nicht maschinell interpretiert).
- Verteilung der Zertifikate beliebig, aber einige feste Key Server bekannt.
- Typischerweise für Verteilung über persönliche Bekannte verwendet (web of trust), aber nicht zwingend.
- Keine Attributzertifikate über Vertrauen, d.h. aus obigem Modell nur *Aut*, *Cert*, und $Trust_{...,1}$.

  Dazu 2 Grade (teilweise bzw. ganz vertraut).

War trotzdem Start für Betrachtung von Vertrauensmodellen in diesem Zusammenhang.

X.509 (vor allem Version 3)

ISO-Standard 9594-8 mit Ergänzungen [ISO 9594-8_91, ISO_9594-8_95, ISO_95]

Vor allem Formatdefinitionen für Zertifikaten. In X.509v3 recht allgemein:

- Beabsichtigte Verwendung und sonstige **Beschränkungen möglich**.
- Können als essentiell oder nicht deklariert werden, d.h. Empfänger muß sie berücksichtigen oder nicht.
- Policy-Feld, wo wohl hinkann, wie gut Zuordnung geprüft wurde.

Formatdefinition in ASN.1 (Abstract Syntax Notation) — ein Standard für Datentypen und ihre Darstellung (sehr explizit: Felder mit Typname, oft Längenfeld, Daten).

Scheint manchen Leuten umständlich, aber für recht allgemeine Datentypen gar nicht schlecht.

Primär für hierarchische Zertifizierung (d.h. feste Policy)

Prinzip:

- 1 Weltzentrale zertifiziert Länderzentralen
- Länderzentralen zertifizieren Regionszentralen

  …

D.h. jeder beginnt mit Schlüssel der Weltzentrale … ($Auth_{B,Welt}$ und $Trust_{B,Welt,\text{all}}$).

- Zertifikate unterscheiden nach "wieder CA" oder "normaler Benutzer"; ersteres enthält implizit Empfehlung, wieder beliebig.
- Start kann auch von unten gehen: *B* beginnt mit Schlüssel von lokaler Zentrale, diese zertifiziert nächsthöhere … Ab niedrigster gemeinsamer Zentrale über *A* und *B* wieder nach unten.

Formate kann man aber beliebig verwenden, auch PGP verwendet es ungefähr.

Spezifische Namen verwendet. (Generelles Problem, an was für Namen man Schlüssel bindet.) Hier „distinguished names“ analog hierarchischen Email-Adressen. Etwas unpraktisch, wenn man Ort wechselt.

Deutsches Signaturgesetz

[SigG_97, SigV_97]
(Zum großen Teil ein Zertifizierungsgesetz)

- Kein Ausschließlichkeitsanspruch, d.h. anders weitergegebene Schlüssel sind erlaubt, nur „hinreichende“ Maßnahmen
- 2-stufige Hierarchie von Zertifizierungsstellen (innerhalb Deutschland)
- Registrierung sorgfältig geregelt: Persönlich und mit Ausweisvorlage. Vermutlich auch Unterschrift verlangt (etwas unklar „nachprüfbar … zu dokumentieren“).
- Beschränkungen im Zertifikat möglich.
- Haftung für falsche Zertifikate noch unklar.

c't und DFN bilden etwa das nach [ct_97, DFN_97].

„Einfachere“ Internet-Vorschläge

(SPKI [Elli_96], SDSI [RiLa_96]):

- Eher Attributzertifikate, d.h. für andere Zwecke.
- Z.T. auch nur X.509v3 in anderer Syntax, ohne das zu merken (z.B. hierarchische Namen in SDSI).

Einfacher, weil Zuordnung zu natürlicher Person nicht vorkommt und Rückruf.

8.2.3 Sonderfälle

Nicht jeder Schlüssel ist Person zugeordnet.

a) **Dienst, Rechner, Server u.ä.**: Als Name den des Objekts verwenden. Zuordnung sollte von deren Besitzer ausgehen. Falls Dienst Schaden anrichten kann, d.h. Signatur gewünscht, klären, ob Betreiber oder Hersteller haftet.

b) **Person anonym:**
   - **Vorweg bekannte Pseudonyme** behandelbar wie Identitäten, wenn Besitzer sich irgendwie unter ihnen „ausweisen“ kann. (Z.B. Nummernkontobesitzer bei Bank.)
   - Gerade Verschlüsselungsschlüssel kann man **einfach so** verteilen (z.B. für Antwort auf Chiffreanzeige)
   - Oft mit **Attributzertifikaten** verbunden (z.B. „unter diesem Pseudonym wurde Geld hinterlegt“; Bsp. Telefonkarte.)

8.3 Literatur

Zu Kap. 7:

Agne_88 G. Agnew: Random sequences for cryptographic systems; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 77-81.

BaBl_95 Th. Baldin, G. Bleumer: CryptoManager++ -- An object oriented software library for cryptographic mechanisms; 12th IFIP International Conference on Information Security (IFIP/Sec '96), Chapman & Hall, London 1996, 489-491.

8-seitige Version unter http://www.semper.org/sirene/publ/BaBl_95CryptoMan++.ps.gz

DaIF_94 D. Davis, R. Ihaka, Ph. Fenstermacher: Cryptographic randomness from air turbulence in disk drives; Crypto 94, LNCS 839, Springer-Verlag, Berlin 1994, 114-120.

Damg_88 I. Damgård: Collision free hash functions and public key signature schemes; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 203-216.

FaMC_85 R. C. Fairfield, R. L. Mortenson, K. B. Coulthart: An LSI Random Number Generator (RNG); Crypto '84, LNCS 196, Springer-Verlag, Berlin 1985, 203-230.

GlVi_88 C. Glowacz, H. Vierhaus: Der Las-Vegas-Chip; Der GMD-Spiegel 18/2-3 (1988) 6-8.

GoGM_86 O. Goldreich, S. Goldwasser, S. Micali: How to construct random functions; Journal of the ACM 33/4 (1986) 792-807.

Zitiert:

c't_97 http://www.heise.de/ct/pgpCA/default.shtml

DFN_97 http://www.cert.dfn.de/dfnpca/certify/

Elli_96 C. Ellison: Generalized Certificates; Webseite, September 1996, Hintergrundtext zu Internet Draft "SPKI (Simple Public Key Infrastructure)", http://www.clark.net/pub/cme/spki.txt.

ISO_95 ISO/IEC JTC 1/SC 21 N9214: Open Systems Interconnection, Data Management and Open Distributed Processing; Draft Amendment ISO/IEC 9594-2/4, 9594-6-2, 9594-7/1, 9594-8/1, July1995.

ISO_9594-8 91 ISO/IEC 9594-8: Information technology - Open Systems Interconnection - Specification - The Directory - Part 8: Authentication framework; ISO International Standard, First edition 15.12.1990, with Technical Corrigendum 1, 15.12.1991.

ISO_9594-8 95 ISO/IEC 9594-8: Technical Corrigendum 2 to ISO/IEC 9594-8: 1990 & 1995; ISO/IEC JTC 1/SC 21/WG 4 and ITU-T Q15/7 Collaborative Editing Meeting on the Directory, Ottawa, Canada, July 1995.

Maur1_96 U. Maurer: Modelling a Public-Key Infrastructure; ESORICS '96 (4th European Symposium on Research in Computer Security), Rome, LNCS 1146, Springer-Verlag, Berlin 1996, 325-350.

RiLa_96 Ronald L. Rivest, Butler Lampson: SDSI — A Simple Distributed Security Infrastructure; Manuscript, April 30, 1996, http://theory.lcs.mit.edu/~rivest/sdsi10.ps.

SigG_97 Artikel 3 des IuKDG (Informations- und Kommunikationsdienste-Gesetz), Dez. 1996, erhältlich von http://www.iid.de/rahmen/iukdgbt.html

SigV_97 Verordnung zur digitalen Signatur (Signaturverordung - SigV), 1997; erhältlich unter <http://www.iid.de/rahmen/sigv.html>.

Zimm_95 Philip R. Zimmermann: The Official PGP User's Guide; MIT Press, Cambridge, England 1995.

Bücher:

- Keins ist ganz vollständig oder ganz systematisch.
- Ford/Baum (siehe Kap. 0.C) aber ganz gut.

9 Techniken für einfache Protokolle, v.a. Authentikationsprotokolle

- **„Einfache“ Protokolle** = kurze interaktive Sequenzen aus den Bausteinen aus Kap. 2.
- **Am häufigsten:**
  - „Gegenseitige Authentikation“.
  - Symmetrischer Schlüsselaustausch
  - Einfache Zahlungssysteme, d.h.
    - ohne (viel) Datenschutz
    - primär Ersetzen handschriftlicher Unterschriften aus Papiersystemen durch digitale.

      (Überweisung, Scheck, Abhebung, Kreditkartenautorisierung.)
- **„Techniken“: Robuster Protokollentwurf:**
  - Klarer Entwurf
  - Vermeidung typischer Fehler
  - Vereinfachung von Sicherheitsbetrachtungen (egal ob informell oder Beweis)

9.1 Freshness-Maßnahmen

9.1.1 Ziele

Grob: Nachricht soll „frisch“ sein, d.h. nicht veraltet.

Detailziel und Grund anwendungsabhängig:

- **Replayvermeidung** als Zusatzziel für Authentikation:

  Empfänger akzeptiert *m* nur so oft, wie Sender *m* authentisiert hat.
  - Für Unterschriften unter normale Texte nicht nötig.
  - Aber z.B. für Überweisungen praktisch, wenn Replayvermeidung automatisiert ist: Dann

    $m = (Empfänger, Betrag)$

    möglich.

- **Aktualität** („timeliness“) von Authentikation:

  Empfänger akzeptiert *m* nur, wenn vor kurzer Zeit authentisiert.

  - Für normale Texte wieder egal.
  - Z.B. bei Türzugangskontrolle: Dann

    m = *„will jetzt rein“*

    möglich.
  - Z.B. bei Austausch von symmetrischem Schlüssel mit Masterschlüssel: Angreifer nicht zuviel Zeit zum Brechen des Sitzungsschlüssels geben.

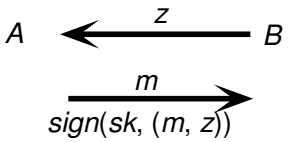
9.1.2 Verfahren

- **Zeitstempel** in Nachrichten.
  - Geht gut, wenn es globale Zeit gibt, z.B. globale Runden.
  - Evtl. Anfangs- und End-Gültigkeitszeit einzeln.
  - Wenn globale Zeit ein Netzservice, nur so sicher wie dieser Service.
- **Sequenznummern**:

  Jeweils für festes Paar Sender-Empfänger:
  - Sender und Empfänger führen Zähler.
  - Nur Nachricht mit neuester Sequenznummer wird akzeptiert.
  - Evtl. speichert Empfänger mehr und akzeptiert alle neuen Nummern.

  (Z.B. Überweisungsnummern.)

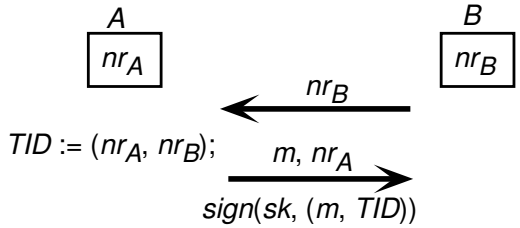
- „**Nonces**“ oder „**Challenge-Response“**: Interaktiv:
  - Empfängerkomponente wählt Zufallszahl
  - läßt sie von Senderkomponente mitsamt Nachricht authentisieren.



  Warum?
  - Korrekt, weil W’keit sehr gering, daß Empfänger *z* wählt, zu der es schon authentisierte Nachricht gibt.
  - Geht ohne globale Zeit und für viele Kommunikationspartner ohne viel Synchronisation.

- **Für beide eindeutige Transaktions-ID**

  Mittelding aus Sequenznummer und Nonce, wenn viele Paare Sender-Empfänger:
  - Einigen auf Transaktionsnummer (*TID* = transaction identifier), die für beide eindeutig ist.
  - Z.B.: Jeder *X* hat lokalen Zähler nr_X.



9.2 Explizitheit

Hauptproblem

Die meisten Protokoll werden gebrochen, weil eine Nachricht anstelle einer anderen verwendet werden kann, z.B.:

- Einfacher Replay.
- Kollisionen bei Nachrichtenformat, z.B. ein *sign*(*m*, *TID*) und ein *sign*(*m*, *Adresse*).

Maßnahmen, um das auszuschließen:

- **Schlüssel nur für 1 Systemtyp verwenden.**

  Vor allem: Nicht mit selbem RSA-Schlüssel signieren und entschlüsseln.

  Grund:
  - Sonst ist aktiver Angriff auf Signatursystem auch einer auf Verschlüsselungssystem und umgekehrt.
  - $\Rightarrow$ Sicherheitsbetrachtungen werden sehr komplex.

- Geg. einen Schlüssel:
  Festes Format für Protokolltyp.

  (Sonst kann man evtl. signierte Nachricht aus Schlüsselaustauschprotokoll *X* in Zahlungssystem *Y* interpretieren.)

- Geg. Schlüssel und Protokolltyp:
  Transaktionsnummern und evtl. **ID der Beteiligten.**

  (Damit Nachrichten nicht von einem Protokoll*ablauf* in anderen verschoben werden können.)

- Geg. Schlüssel, Protokolltyp, Transaktion:
  Nachrichtennr. oder -name innerhalb Protokoll.

  (Nr. bei einfachem linearen Ablauf; komplizierter Namen sonst.)

- **Keine verschlüsselten Nachrichten unterschreiben.**

  (Sonst hängt Inhalt von korrektem Entschlüsselungsschlüssel u.ä. ab.)

  D.h.

  $$\boldsymbol{ver}(\boldsymbol{k}, \boldsymbol{sign}(\boldsymbol{sk}, \boldsymbol{m})),$$

  nicht *sign*(*sk*, *ver*(*k*, *m*)).

Effizienz?

- All das macht Nachrichten nicht wesentlich länger.
- Minimierung nur in Extremfällen nützlich (ähnlich Assemblerprogrammierung).

Weitere Explizitheitsmaßnahmen

- **Nachrichtensemantik klarstellen** (z.B. von Zertifikaten):

  Bedeutung von Nachrichtenfeldern immer klarmachen, evtl. als Text direkt in Nachricht.

- **Forderungen an Primitive klarstellen:** Z.B. nicht von symmetrischer Verschlüsselung oder Hashfunktion implizit auch Authentikation verlangen.

  (Ist bei dieser Community oft schlecht gemacht!)

- **Präzise Nachrichtenformate**: z.B. Darstellung von *n*-Tupeln, Padding muß eindeutig sein.

  Oft in
  - Sprache ASN.1 [ISO/IEC 8824] (zum Ausdrücken der Nachrichten).
  - mit Encoding rules (Bitdarstellung) [ISO/IEC 8825].

- **Tests ausspezifizieren**: Z.B. oben bei Nonce und Transaktions-ID, was *B* prüft.

  (Wird sonst bei Implementierung evtl. vergessen.)

  Anm.: Eine wirklich genaue Protokollspezifikation
  - gibt die beiden lokalen Programme für jeden Teilnehmer an, z.B. mit Ausnahmebehandlung,
  - nicht nur den korrekten Nachrichtenfluß wie oben.

  Beispielsprache SDL [DiCh_83, CCITT XR26E].

- **Fehlertoleranz**: Vor allem Maßnahmen bei Protokollabbruch des Partners oder Leitungsunterbrechung. Evtl. auch gegen lokalen Absturz.

  Meist nochmal genauso lang wie Protokoll ohne, und wesentlich schwieriger. (Viel mehr Fälle, etwa wie ein „goback“)

9.3 Dispute mitbetrachten

Viele Sicherheitseigenschaften eines Protokolls sind nur in Zusammenhang mit späterem Disput (vor Gericht und Öffentlichkeit) formulierbar. Dann

- **Speicherung von Beweismitteln**
- **Disput selbst**

mit spezifizieren.

a) Sonst bei Implementierung evtl. vergessen oder falsch.

b) Jede Sicherheitsbetrachtung muß den Disput sowieso enthalten.

Bsp.: Abheben von Geld von Konto auf eine „electronic purse“ (Hardware oder Software): Der nötige Disput ist:
- Eine Abhebung erscheint auf Kontoauszug.
- Kunde sagt, sei falsch.
- Bank sollte Gericht Beweismittel zeigen:
  - Unterschrift des Kunden.
  - Auf Replayvermeidung achten.

Version 1: Abhebungen mit **Sequenznr.**, diese auch auf Kontoauszug. Dann
- Digitaler Disput über Abhebung zu dieser Nummer,
- nichtdigitaler, ob Bank zu diese Nummer zweimal Geld abgebucht hat.

Version 2: Genauso mit **Nonce**? Problematischer: Bank könnte beliebig alte Überweisung nochmal vorzeigen.
- Kunde müßte alles speichern (solange sie Schlüssel nicht wechselt)
- ihre Kontoauszüge durchsuchen,
- und Doppeltauftreten eines Nonce gilt als Fehler der Bank.

9.4 Identifikation meist zu wenig

Es gibt in Literatur und Praxis relativ viele „Identifikationsprotokolle“. Für die meisten Anwendungen aber zu schwach:

- **Fast immer eigentlich Authentikation von Nachrichten nötig!**

Ziel von Identifikation (grob): Prüfen, daß bestimmte Person „jetzt“ „am anderen Ende“ ist.

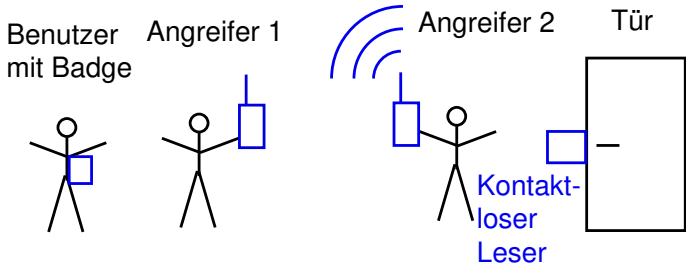
Entspricht Authentikation einer festen Nachricht mit Freshness.

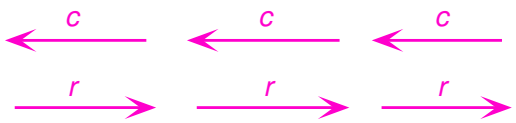
Hauptnutzen: Türzugang mit Karten oder Badges (weil Nachricht da eindeutig: „ich will hier rein“).

„Hier“ ist aber nicht trivial.

Bsp. wenn Badge automatisch auf Tür-Challenge antwortet:

Ahnungsloser Benutzer weit weg von Tür:





Tür öffnet sich!

Hier c = challenge, r = response.

Geht aber generell: Angreifer leiten alle Nachrichten zwischen Badge und Leser weiter.

Eine Version eines „middle-person“ oder „man-in-the-middle“ attack, auch „mafia fraud“ (das aber für falsche Anwendung auf Nachrichtenauthentikation).

Mögliche Gegenmaßnahmen

- **Nachrichtenauthentikation** (m = welche Tür). Benötigt Display auf Badge und Benutzer-OK.
- **Realzeitmessungen durch Leser.**
- (• **Faradaykäfig** und Beobachtung auf Kabel hin im Eingangsbereich.)

Bsp. von Identifikationssystemen:

- **Symmetrisch:** TAN (d.h. Liste vorweg ausgetauschter Zufallszahlen)

  Erinnerung: Obwohl von Banken oft verwendet, authentisiert es eigentlich keine bestimmte Nachricht!

- **Asymmetrisch:**
  - Zero-Knowledge-Beweise: Beweis, daß man bestimmten geheimen Schlüssel kennt, z.B. [FiSh_87].
  - Benutzer gibt vorweg gewählte Urbilder unter Einwegfunktion bekannt. Z.B.

    $$pk = f^n(sk),$$

    sukzessive Identifikation mit

    $$f^{n-1}(sk),\ f^{n-2}(sk),\ \ldots,\ sk.$$

9.5 Beispielprotokoll: Sitzungsschlüsselaustausch

(Z.B. für TCP-Verbindungen)

Ziel: Selbst wenn Angreifer manche Sitzungsschlüssel erhalten kann, erhält er keine Vorteile für andere Sitzungen.

Hier vorausgesetzt:

- Asymmetrische Verfahren zum Schlüsselaustausch. (Letzter Punkt in Kap. 8.2.1.)
- Teilnehmer A und B kennen zuverlässig öffentliche Schlüssel des jeweils anderen (mit Maßnahmen aus Kap. 8.2.1).

Verfahren:

a) **Mit Zeitstempeln:**

- **Voraussetzung:** Globale Zeit mit maximaler Abweichung Δ, Nachrichtenlaufzeiten meist $\leq d$:
- **1-Runden-Protokoll** möglich:

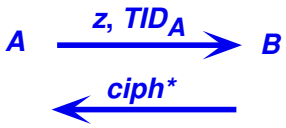
***A*:**
- Generiere Sitzungsschlüssel $\boldsymbol{k_1, \ldots, k_4}$ (für sym. Verschlüsselung bzw. Auth. und Nachrichten von A bzw. B);
- $\boldsymbol{t_A}$:= lokale Zeit von A;
- $\boldsymbol{TID_A}$:= für A derzeit lokal eindeutige Sitzungsnummer.
- ***msg*** := („Protocol X Version Y“, „session start“, t_A, A, B, TID_A, $k_1, \ldots, k_4$);
- ***sig*** := $sign(sk_{A,\,sig}, msg)$;
- ***ciph*** := $ver_{asym}(pk_{B,ver}, (msg, sig))$;

***B*:**
- Entschlüssele *ciph*.
- Prüfe Protokoll und Versionsnr.
- Prüfe daß t_A nicht älter ist als $d + \Delta$ Minuten (d, δ Konstanten bei B).
- Prüfe Namen B.
- Suche $pk_{A,\,sig}$ zu Namen A und teste damit Signatur.

- Suche unter gespeicherten Nachrichten mit noch gültigem Zeitstempel ob Replay.

  Wenn nicht, speichere *msg* dort.
- Initialisiere Sitzungskontext mit Werten (A, TID_A, k_1, …, k_4), und Zähler *hin* := 0, *rück* := 0.

b) **Mit Nonces:**



Hierbei *ciph** wie oben *ciph*, nur

- Rollen von A, B vertauscht
- B verwendet aber TID_A
- Nonce z statt Zeitstempel t_A.

A prüft z und muß keine alten Sitzungsstartnachrichten speichern.

9.6 Techniken zum Fehlerfinden

Die wenigsten „einfachen Protokolle“ werden bewiesen. (Es fehlen auch schon Definitionen, z.B. von Zahlungssystemen inkl. Disputen.)

Dafür gibt es einige voll formale Kalküle zum Fehlerfinden, insbesondere „BAN-Logik“ [BuAN_90]. Unter Praktikern (!) sehr beliebt.

Beachte:

– Ein BAN-Logik (o.ä.) Beweis eines Protokolls beweist nicht Korrektheit, sondern nur Abwesenheit bestimmter Fehler!

– Viele Leute wissen das nicht, Aussagen „ein bewiesenes Protokoll“ also vorsichtig prüfen.

– Die Logiken reichen nur für Authentikationsprotokolle; jede Erweiterung ändert die Logik.

\+ Dafür sind Beweise „**formal**“ im Sinne der Logik, d.h. maschinell zumindest verifizierbar (Anwendung von Regeln aus festem Kalkül).

Alles kryptographisch „formale“ ist in dieser Terminologie nur „**rigoros**“, wie Mathematik.

Persönliche Meinung: 1/10 soviel Arbeiten in diesem Bereich würden genügen, dafür mehr Anstrengungen bei echten Definitionen und Beweisen.

Grundprinzip: Algebraische Abstraktion von Kryptographie. (Nach [DoYa_83].)

Kryptographische Systeme als abstrakte Datentypen, in denen **nur** bestimmte Kürzungsregeln gelten, z.B.

$$ent(k, ver(k, m)) = m$$

Nicht immer richtig, z.B. könnte

$$odd(ver(k, m))$$

immer gelten, oder $ver(k, ent(k, m)) = m$ in der Hälfte der Fälle.

Anm. 1: Paßt besser auf Pseudozufallspermutationen als auf Verschlüsselung, aber allgemeiner formaler Zusammenhang bisher nicht bewiesen.

Anm. 2: Ist vielen BAN-Logik-Benutzern nicht bewußt (man sieht diese Annahme dort nicht). Wenn man aber BAN-Logik mit formaler Semantik versieht [AbTu_91], dann ist sie so.

Richtung 1:

Direkte Angriffssimulation in diesen algebraischen Abstraktionen, z.B. [DoYa_83, Mill_84, Mead1_94].

- Nachrichten sind „Wörter“ aus Symbolen *ver*, *ent* usw.
- Angreifer kann Nachrichten an ehrliche Teilnehmer schicken und bekommt ggf. Antwort.
- Auf bisher gesammelte Wörter kann er Kürzungsregeln anwenden.
- System prüft, ob so je z.B. geheimer Schlüssel rauskommt.

Problem: Suchraum primär unendlich, d.h. nur Semientscheidungsalgorithmus. (Außer bei sehr eingeschränkten Protokollklassen.)

Richtung 2:

Sog. Glaubenslogiken (logic of belief).

\+ Für viele Leute intuitiver als obiges.

\+ Ziel: Sicherheitsbeweise. (Statt „Abbruch ohne erfolgreichen Angriff“.)

– Die Beweise beweisen nicht, was man erwartet, nicht mal in algebraischer Abstraktion.

Syntax:

- Hier nur für symmetrische Kryptographie
- in verbesserter Version aus [AbTu_91]:

1. **Mehrere Mengen von Konstanten:** Für Teilnehmer, Schlüssel, elementare Aussagen, Nonces, Zeitstempel.
2. **Nachrichten und Formeln**. Ziemlich gemischt, deshalb gemeinsame rekursive Definition:

   a) **Nachrichten**:
   - Jede Konstante; jede Formel
   - Tupel von Nachrichten.

   - $\{X^P\}_K$ wenn
     - X Nachricht
     - P Teilnehmer. (Absender)
     - K Schlüssel. (X damit verschlüsselt)
   - $<X>_Y$ wenn X, Y Nachrichten. („Kombination“; etwas komisch relativ zu Tupel.)
   - ‘X’ wenn X Nachricht. („nur weitergeleitet“).

   b) **Formeln**:
   - Elementare Aussagen.
   - Kombination mit $\wedge$, $\neg$.
   - *fresh*(X) für Nachricht X.
   - Wenn P, Q Teilnehmer und K Schlüssel:
     - $P \overset{K}{\leftrightarrow} Q$: (gemeinsamer Schlüssel)
     - P *has* K.
   - Wenn P, Q Teilnehmer und X Nachricht:
     - P *sees* X (erhält es in Nachricht)
     - P *says* X
     - P *said* X

     - $P \overset{X}{\leftrightarrow} Q$ (gemeinsames Geheimnis)
   - Wenn P; Q Teilnehmer und ϕ Formel:
     - P *believes* ϕ.
     - P *controls* ϕ. (darf es entscheiden)

3. **Spezifikation von Ausgangssituation** („initial belief“), z.B.

   A *believes* $(A \overset{K_{AS}}{\leftrightarrow} S) \wedge B$ *believes* $(B \overset{K_{BS}}{\leftrightarrow} S)$.

4. **Spezifikation von Ziel** z.B.

   A *believes* $(A \overset{K_{AB}}{\leftrightarrow} B) \wedge B$ *believes* $(A \overset{K_{AB}}{\leftrightarrow} B)$.

   (Nachteil: Protokollspezifische Spezifikation, K_{AB} ist nur durch Protokoll definiert!)

5. **Kalkül: ca. 30 Regeln („Axiome“):**

   (Ziemlich viel für kleinen Anwendungsbereich!)
   - **Normale logische**
   - **Glauben**, z.B.

     P *believes* $\phi \wedge P$ *believes* $\psi \Rightarrow$
     P *believes*$(\phi \wedge \psi)$.

     (Spannender: Glaubt man alle richtigen Sachen?)

- **Verschlüsselung** (eher Authentikation!)
  $P \overset{K}{\leftrightarrow} Q$: $\wedge$ P *sees* $\{X^S\}_K$ mit $S \neq P$
  $\Rightarrow Q$ *said* X.

  (D.h. aus Sicht von bravem P, das nie Nachrichten mit falschem Absender verschlüsselt.

  Nachteil: Wenn Protokoll nun doch Verschlüsselung mit falschem Absender enthält …)
- **Sehen**: V.a. mit Nachricht auch Teile, sofern man sie entschlüsseln kann.
- ***said*** so ähnlich.
- **Kontrolle**: P *controls* $\phi \wedge P$ *says* $\phi \Rightarrow \phi$.

  (Z.B. P Schlüsselserver P, ϕ Geheimheit von neuem Schlüssel.)
- **Freshness**: Mit Teilnachrichten auch ganze Nachricht.
- ***says***:

  fresh(X) $\wedge$ P *said* $X \Rightarrow$ P *says* X.

6. **Grundstruktur von Beweisen**:
   - Anfangs glaubt man an gewisse Schlüssel.
   - Dann wählt jemand Sitzungsschlüssel und sagt, die seien gut.
   - Man sieht das in Nachricht.
   - Mit Freshness und Control glaubt man es.

Problem:

- Die Logik ist **monoton**.

  (D.h. wie bei den meisten Logiken fügt man weitere gültige Formeln nur hinzu.)
- Nur deshalb leicht benutzbar. (Sobald man Zielformel abgeleitet hat, kann man sie glauben.)
- Wenn nun
  - am Protokollanfang ein Schlüssel gut ist,
  - man deshalb $P \overset{K}{\leftrightarrow} Q$ voraussetzt
  - aber **Fehler innerhalb des Protokolls** macht, so daß K bekannt wird,

  bleibt die jetzt falsche Formel $P \overset{K}{\leftrightarrow} Q$ erhalten.

Anm.: Wieso kann eine so „falsche“ Logik eine korrekte formale Semantik haben?

- Intuitiv wirkt es falsch, weil man die Ableitung temporal auffaßt, insbesondere die „initial beliefs“ als „zu Protokollbeginn“.
- Formal werden sie aber über das ganze Protokoll definiert, d.h. formal gilt $P \overset{K}{\leftrightarrow} Q$ nur, wenn es über das Protokoll erhalten bleibt.
- Damit verliert man aber den Hauptsinns des Beweises: Wie soll man das vorweg prüfen? Wieder eine komplette Protokollanalyse …

9.7 Literatur

Zitiert:

AbTu_91 M. Abadi, M. Tuttle: A Semantics for a Logic of Authentication; 10th Symposium on Principles of Distributed Computing (PODC), ACM, New York 1991, 201-216.

BuAN_90 M.Burrows, M. Abadi, R. Needham: A Logic of Authentication; ACM Transactions on Computer Systems 8/1 (1990) 18-36.

CCITT_XR26E International Telegraph and Telephone Consultative Committee (CCITT): Recommendation Z.100: CCITT Specification and Description Language (SDL); CCITT, Study Group X, Report R 26, July 1992.

DiCh_83 G. Dickson, P. de Chazal: Status of CCITT Description Techniques and Application to Protocol Specification; Proceedings of the IEEE 71/12 (1983) 1346-1355.

DoYa_83 D. Dolev, A.. Yao: On the Security of Public Key Protocols; IEEE Transactions on Information Theory 29/2 (1983) 198-208.

FiSh_87 A. Fiat, A. Shamir: How to Prove Yourself: Practical Solutions to Identification and Signature Problems; Crypto '86, LNCS 263, Springer-Verlag, Berlin 1987, 186-194.

ISO_8824-1_95 Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation; ISO International Standard 8824-1, 1st Edition, 15.10.1995.

ISO_8825-1_95 Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER); ISO International Standard 8825-1, 1st Edition, 15.10.1995.

Mead1_94 C. Meadows: The NRL Protocol Analyzer: An Overview; Journal of Logic Programming 19&20 (1994) 1-19.

Mill_84 J. Millen: The Interrogator: A Tool for Cryptographic Protocol Security; Proceedings of the 1984 Symposium on Security and Privacy, IEEE, April 29 - May 2 1984, Oakland, California, 134-141.

Bücher u.ä. (nicht sehr ähnlich)

- In Schneier, vgl. Kap. 3.3, 3.4
- In Menezes et al. Kap. 10.3, 12.5
- Robustheit als Prinzip: [AnNe1_95]:

  R. Anderson, R. Needham: Robustness Principles for Public Key Protocols; Crypto '95, LNCS 963, Springer-Verlag, Berlin 1995, 236-247.

10 Juristische Aspekte

10.1 Signaturen

- **Generell im deutschen Recht:**
  - **Freie Beweiswürdigung** (d.h. *alles* kann ein Beweismittel sein.)
  - **Spezielle Rollen der eigenhändigen Unterschrift:**
    - Willenserklärungen, die Schriftform brauchen (BGB = Bürgerliches Gesetzbuch §126).
    - Im Beweis durch Privaturkunde. (ZPO = Zivilprozeßordnung §§ 416, 439, 440).

    Beachte: Urkunde ist Beweis für Erklärung des Inhalts. Aber Beweislast für Echtheit der Urkunde bei Empfänger!

- **Speziell für digitale Signaturen:** [SigG_97, SigV_97], vor allem zu
  - Zertifizierung (siehe Kap. 8.2.2H),
  - Gerätesicherheit (Verlangt mehr, als im Moment üblich, und Zertifizierungsstellen müssen das prüfen). Vgl. Kap. 12.

  Bisher keine Anbindung an obige Gesetze, aber geplant

  Erinnerung: Verwendung anderer Verfahren freigestellt.

10.2 Verschlüsselung

Lage zur Zeit

- **Einsatz, Import, Verbreitung in Deutschland** uneingeschränkt legal.

  (Auch in meisten „wichtigen“ anderen Ländern. Nicht Frankreich, nahöstl. Staaten, …)
- **Export** unterliegt Außenwirtschaftsgesetz.
  - Verschenken wohl problemlos. (Schriftlich Veröffentlichen auf jeden Fall.)
- **Export aus USA** dort wie Kriegswaffen
  - Aus USA legal exportierte Verschlüsselung hier nicht sinnvoll.
  - Strafbar macht sich nur Exporteur, nicht Benutzer im Ausland.
  - In Papierform erlaubt, und Verkauf von Trademark (nicht Lizenz).

Diskussion Kryptogesetze

Diskussionen über Teilverbote zwecks Abhörbarkeit für Strafverfolgung oder Nachrichtendienste.

Beachten:

- **Superencryption:** Entschlossene Benutzer schicken

  $$ver_{erlaubt}(k_1, (ver_{verboten}(k_2, m)).$$

  Nur verhinderbar, wenn *alles* entschlüsselt und gelesen würde!
- **Steganographie:** Selbst wenn alles mitgelesen: Verstecken von Daten in anderen möglich. (In jeder Art von Rauschen oder Zufallskomponente.)

  (D.h. man hindert nur Ehrliche am Verschlüsseln.)
- Speichern von Schlüsseln an zentralen Stellen gibt **Angriffspunkte** (technisch und bzgl. Bestechung).

  (Selbst bei Secret Sharing und Beschränkung des Abhörzeitraums.)

- Sog. „**Key Recovery**“ für Einzelbenutzer bei Schlüsselverlust anders lösen
  a) Sowieso nicht für kommunizierte Daten, sondern für gespeicherte.
  b) Lokal organisieren, wie normale Backups.
- Technische Beobachtungsmöglichkeiten nehmen auch in anderen Bereichen zu, d.h. direkte **Einzelüberwachung** bei Bedarf wird eher einfacher.

Umkehrung?

- Verschlüsselung müßte anhand genereller Datenschutzgesetze für viele Zwecke eigentlich vorgeschrieben sein. Kaum umgesetzt.

10.3 Literatur

Kann sich alles schnell ändern.

- Deutsche Gesetze derzeit auf http://www.iid.de/rahmen/ gesammelt.
- Sonstige juristische Kryptographieaspekte oft am schnellsten im Web. (Unsere Startpointer: http://www.semper.org/sirene/outsideworld/security.html#politics
- Analysen z.B. in Zeitschrift DuD <http://www.dud.de>

11 Produkte, Standards

Unterscheide:

- Für Endanwender primär mit Einbindung in irgendwelche Anwendungen nötig (wenigstens GUI = graphical user interface und File-Behandlung).
- API-Standards (Programmierschnittstelle); auch „Frameworks“.
- Protokollstandards (Nachrichtenformat und Austausch).
- Bibliotheken von Algorithmen.

11.1 Kerberos

- Bekanntestes System für rein symmetrische Schlüsselverteilung über Zentralen.
- Protokollstandard und Library.
- „Ticket“ als Name für Servernachrichten, die den Teilnehmern Sitzungsschlüssel geben.

Aus MIT-Projekt „Athena“ für sichere verteilte Systeme, ab etwa 1987; jetzt etwas veraltet.

11.2 PGP („Pretty Good Privacy“)

Bekanntestes Programm für privaten Bereich, dort kostenlos. Evaluation willkommen.

Einsatzmöglichkeiten
Relativ umfassende Anwendung. (Demnächst auch Developer Tools.)

- Email
- Direkte Dateiverschlüsselung
- Transparente Dateiverschlüsselung (PGP-Disk).

Algorithmen

- Asymmetrische Verschlüsselung (Version 2 RSA, Version 5 ElGamal).
- Signaturen (Version 2 RSA, Version 5 DSS)
- Darunter hybrid mit guter symmetrischer Kryptographie.
- Flexible Schlüsselverteilung, s. Kap. 8.2.2H.

Details siehe Übungen.

11.3 Sonstiges für Email

- Z.T. demnächst direkt eingebaut, aber bei US-Produkten nicht genug.
- **S/MIME:** Wird wohl Protokollstandard außer PGP, wegen Industrieunterstützung.
- **PEM:** (privacy enhanced mail), älter als S/MIME, nicht durchgesetzt.

Alles im **Grobkonzept** ähnlich:

- öffentliche Schlüssel,
- intern hybride Systeme,
- Nachrichtenformate spezifiziert,
- Behandlung von Zertifikaten (nicht immer sinnvoll, z.B. scheint S/MIME automatische Zertifizierung auf Email-Anfragen hin vorzuschreiben).
- Nicht volle Anwendungen wie PGP, dazu dann spezielle Produkte.

11.4 SSL („Secure Socket Layer“)

- Von Netscape. Beabsichtige Anwendung direkt auf TCP-Verbindungen; Spezifikation unabhängig davon.
- **Protokollstandard**, andere Implementierungen erlaubt (z.B. SSLeay frei verfügbar)
- Primär
  - **asymmetrischer Austausch** von symmetrischen Sitzungsschlüsseln (SSL-Handshake)
  - und Formate für Nachrichten in Sitzung (SSL-Record-Protocol).
- Algorithmen größtenteils frei wählbar, außer HMAC für sym. Authentikation.
- Zertifikate X.509v3; wohl nur für hierarchische Struktur.
- **Keine Signaturen** auf Nutznachrichten, d.h. keine rechtliche Bindung.

11.5 IPSEC (für IPv6)

- **Schlüsselaustausch nicht** mit spezifiziert, oberhalb IP geplant.
- „**Security Association**“ = Sitzung im Sinn von Sitzungsschlüssel, simplex. Pro Empfänger eindeutig.
- **Extension Headers** (d.h. zusätzliche Header für IP-Pakete).
  - **Authentication Header**: Enthält Identifikator der Sitzung und MAC. (Mit Tsudik-MAC oder HMAC, s. Kap. 6.3.4.)
    Authentikation auch über normale Header, aber Teile ändern sich unterwegs (Hop-Zähler); diese durch 0 ersetzt.
  - **IP Encapsulating Security Payload** Header: Schlüsselidentifikator, dann verschlüsselte Daten.
    - Transport mode: Nur Nutzdaten verschlüsselt.
    - Tunnel mode: Ganzes Paket, außen neuer Header.

11.6 SSH („Secure Shell“)

- Kryptographisch wieder v.a. Sitzungsschlüsselaustausch und Paketformate innerhalb Sitzung.
- Anstatt rlogin u.ä., Betriebssystemeinbindung später.

11.7 Literatur

Kerberos: Vgl. Schneier-Buch.

PGP: http://www.pgp.com/

und PGP-Buch (s. Kap. 0.C)

Sonstige Email-Standards: Start z.B. bei: http://www.semper.org/sirene/outsideworld/security.html#prot.

SSL: http://home.netscape.com/newsref/std/SSL.html

Sicherheitsanalyse in [WaSc_96]: David Wagner, Bruce Schneier: Analysis of the SSL 3.0 Protocol; 2nd USENIX Workshop on Electronic Commerce, 1996, 29-40.

IPSEC: http://www.ietf.org/html.charters/ipsec-charter.html

Netter in [Stal_96]: William Stallings: IPv6: The New Internet Protocol; IEEE Communications Magazine 34/7 (1996) 96-108.

SSH: http://www.cs.hut.fi/ssh/