Verschlüsselungsverfahren

Man unterscheidet:

- **Symmetrische Verfahren**

  $\rightarrow$ Sender und Empfänger verwenden den *gleichen* Schlüssel

  NACHTEIL: Der Schlüssel muss *vorher* über einen sicheren Kanal Ausgetauscht werden.

  VORTEIL: Effizient implementierbar.

  BEISPIEL: DES

- **Asymmetrische Verfahren**

  $\rightarrow$ Jeder Teilnehmer erhält einen *öffentlichen* Schlüssel und einem *privaten* Schlüssel. Zum Verschlüsseln ist nur der öffentliche Schlüssel des Empfängers nötig. Dieser entschlüsselt die Nachricht dann mit seinem privatem Schlüssel.

  VORTEIL: Ein (geheimer) Schlüsselaustausch ist nicht erforderlich.

  NACHTEIL: Rechenaufwendig.

  BEISPIEL: RSA

RSA (1)

Benannt nach seinen Erfindern: $\textsc{Rivest}$, $\textsc{Shamir}$, $\textsc{Adleman}$ (1978)

Initialisierung:

- **Empfänger** wählt zwei (grosse!) Primzahlen p und q,
  setzt $n = pq$ und $k = (p-1)(q-1)$,
  wählt eine beliebige Zahl $d \in \mathbb{Z}_k$ mit ggt$(d, k) = 1$,
  berechnet $e \in \mathbb{Z}_k$ mit $ed \equiv 1 (\text{mod } k)$ und
  veröffentlicht n und e.

Verschlüsselung: (ohne Einschränkung ist die zu verschlüsselnde Nachricht eine Zahl $m \in \mathbb{Z}_n$)

- **Sender** berechnet $c = m^e$ mod n und schickt c an Empfänger.

Entschlüsselung:

- **Empfänger** berechnet $m = c^d$ mod n.

RSA (2)

KORREKTHEIT:

müssen zeigen: $\forall m \in \mathbb{Z}_n$ gilt $(m^e)^d \equiv m \pmod{n}$ $\longrightarrow$ Satz von Fermat

IMPLEMENTATION:

- Bestimmung von p und q:

  $\longrightarrow$ Primzahltest, vgl. DSII

- Bestimmung von d:

  $\longrightarrow$ Durchprobieren, Algorithmus von Euklid

- Bestimmung von e:

  $\longrightarrow$ Erweiterter Algorithmus von Euklid

- Bestimmung von m^e mod n:

  $\longrightarrow$ siehe nächstes Slide

SICHERHEIT:

Beruht auf der (unbewiesenen!) Annahme, dass die Berechnung der Faktorisierung von n bzw. das Rekonstruieren der Nachricht m aus Kenntnis von $c = m^e$ mod n “schwierig” ist.

$\rightarrow$ Einwegfunktionen

Effiziente Berechnung von a^x mod n

IDEE: Abwechselndes Quadradrieren und Multiplizieren...

BEISPIEL: $a = 43,\ x = 50,\ n = 67$.

1.Schritt: Berechne Binärdarstellung von $x = 50\ =\ 110010_2$.

2.Schritt: Abwechselndes Quadradrieren und Multiplizieren. Dabei Multiplikation mit a bzw. 1, falls an der entsprechenden Stelle in der Binärdarstellung eine 1 bzw. eine 0 steht. Also

$$43^{50} = (((((43)^2 \cdot 43)^2 \cdot 1)^2 \cdot 1)^2 \cdot 43)^2 \cdot 1$$

Schieben wir nun noch nach jeder Operation eine Modulo Berechnung ein, erhalten wir:

$$(((((43^2 \bmod 67)^2 \cdot 43 \bmod 67)^2 \bmod 67)^2 \bmod 67)^2 \cdot 43 \bmod 67)^2 \bmod 67.$$

RESULTAT: Für die Berechnung von a^x mod n benötigt man lediglich $\mathcal{O}(\log x)$ Multiplikationen und Modulo Berechnungen für Zahlen mit höchstens $\mathcal{O}(\log n)$ Bits.

Einwegfunktionen

Eine Funktion f heisst **Einwegfunktion**, wenn gilt:

- $f(x)$ lässt sich für alle x effizient berechnen, aber
- $f^{-1}(y)$ ist für "fast alle" Werte y "schwer" zu berechnen.

BEISPIELE:

- Multiplikation zweier Primzahlen

  Umkehrfunktion: Zerlegung in Primfaktoren

- Exponentiation $x \to a^x \bmod n$

  Umkehrfunktion: diskreter Logarithmus

- Quadrieren $x \to x^2 \bmod n$

  Umkehrfunktion: Wurzelziehen modulo n

RSA (3)

Verwendung von RSA zum Authorisieren von Nachrichten:

Initialisierung:

- **Sender** wählt zwei (grosse!) Primzahlen p und q,
  setzt $n = pq$ und $k = (p-1)(q-1)$,
  wählt eine beliebige Zahl $d \in \mathbb{Z}_k$ mit $\text{ggt}(d, k) = 1$,
  berechnet $e \in \mathbb{Z}_k$ mit $ed \equiv 1 (\text{mod } k)$ und
  veröffentlicht n und e.

Verschlüsselung: (ohne Einschränkung ist die zu verschlüsselnde Nachricht eine Zahl $m \in \mathbb{Z}_n$)

- **Sender** berechnet $c = m^d \text{ mod } n$ und schickt c an Empfänger.

Entschlüsselung:

- **Empfänger** berechnet $m = c^e \text{ mod } n$.

Zugangskontrollsysteme (1)

- “Klassische” Variante:

  Initialisierung:

  Benutzer bekommt Kennung & Passwort

  Rechner bekommt Kennung & (mit Einwegfunktion) verschlüsseltes Passwort

  Zugang:

  Benutzer schickt Kennung & Passwort an Rechner,

  Rechner verifiziert, daß Passwort gültig ist.

  NACHTEILE:

  — Kommunikation kann abgehört werden ($\rightarrow$ Abhilfe: slogin)

  — Rechner kennt danach das Passwort ($\rightarrow$ Abhilfe: Zero Knowledge Protokolle)

Zugangskontrollsysteme (2)

- Zero Knowledge Variante:

  Initialisierung:

  Benutzer bekommt 3-färbbaren Graphen & 3-Färbung

  Rechner bekommt *nur* den Graphen

  Zugang:

  Benutzer wählt sich eine beliebige Permutation $\pi : \{1, 2, 3\} \to \{1, 2, 3\}$ der Farben des Graphen und verschliesst die (neue) Farbe eines jeden Knotens in einer eigenen Kiste. Dann schickt er folgendes an den Rechner:

  den Graphen und

  für jeden Knoten die verschlossenen Kiste mit der (neuen) Farbe des Knotens.

  Rechner wählt sich zwei beliebige durch eine Kante verbundene Knoten und lässt sich vom Benutzer die Schlüssel für diese beiden Knoten (und nur für diese!) geben. Dann öffnet er die beiden Kisten und verifiziert, dass die beiden Knoten verschiedene Farben haben.

Digitales Verschliessen eines Bits

NOTWENDIG: sogenannte **Einwegfunktionen** mit **Falltür**

BEISPIEL: **Quadratische Reste**

Gegeben: $n = pq$ mit p, q Primzahlen und $x \in \mathbb{Z}_n$

Frage: Gibt es ein $z \in \mathbb{Z}_n$ mit $x = z^2$ mod n?

$\longrightarrow$ “schwierig”, aber einfach falls p und q bekannt sind

VERSCHLIESSEN:

Wähle (grosse) Primzahlen p, q und setze $n = pq$.

Falls zu verschliessendes Bit eine Eins ist, sende (n, x), wobei x quadratischer Rest.

Falls zu verschliessendes Bit eine Null ist, sende (n, y), wobei y *kein* quadratischer Rest.

AUFSCHLIESSEN:

Sende Primzahlen p und q.

Electronic Cash (1)

(WUNSCH-) ZIELE:

- Jede „Münze“ darf nur einmal verwendet werden,
- Kunde möchte anonym bleiben,
- Bank möchte wissen **wer** betrügt, falls eine Münze mehrfach verwendet wird.

HIER: Beispiel für eine grundsätzliche Realisierung ...

Electronic Cash (2)

BANK: Wählt p, q, n, e, d wie bei RSA, veröffentlicht n, e und eine Hash-Funktion f.

(Hash-Funktionen sind Einwegfunktionen, die “lange” Texte auf “kurze” abbilden.)

ABHEBEN:

$\forall i = 1, \ldots, 20$:

Kunde wählt a_i zufällig und berechnet $b_i := a_i\ XOR\ \langle$Kundennr., Geldscheinnr.$\rangle$.

Kunde sendet $m_i := \langle f(a_i) || f(b_i) \rangle$ an die Bank.

Bank sendet $c_i := (m_i)^d$ an Kunden.

Wichtig: Kennt die Bank nur a_i **oder** b_i, kann sie daraus den Kunden *nicht* identifizieren. — Sobald die Bank aber a_i **und** b_i kennt, kann sie den Kunden identifizieren!

Modifikation, um sicherzustellen, dass Kunde nicht mogelt:

Bank verlangt 40 Paare $\langle f(a_i) || f(b_i) \rangle$, wählt davon 20 zufällig aus, lässt sich bei diesen a_i und b_i vom Kunden geben und überprüft, dass der Kunde hier wirklich seine richtige Kundennummer verwendet hat.

Electronic Cash (3)

BEZAHLEN:

Kunde: Sendet $(c_1, \ldots, c_{20})$ an Händler.

Händler: $\forall i = 1, \ldots, 20$

wählt $t_i \in \{0, 1\}$ zufällig und sendet t_i an Kunden

Kunde: $\forall i = 1, \ldots, 20$

Falls $t_i = 0$: liefert b_i und $f(a_i)$,

falls $t_i = 1$: liefert a_i und $f(b_i)$.

Händler: $\forall i = 1, \ldots, 20$

Verifiziert, dass $(c_i)^e = \langle f(a_i) || f(b_i) \rangle$.

Doppeltes Bezahlen:

Mit Wahrscheinlichkeit $1 - 2^{-20} \approx 0.999999$ gibt es ein $1 \leq i_0 \leq 20$ für das bei den beiden Zahlvorgängen einmal $t_{i_0} = 1$ und einmal $t_{i_0} = 0$ verwendet wurde. Nach einem doppelten Bezahlen kennt die Bank daher a_{i_0} <u>und</u> b_{i_0} und damit auch die Identität des Kunden.