Kryptographie und Komplexität

Einheit 4

Public Key Kryptographie mit RSA

1. Ver- und Entschlüsselung
2. Schlüsselerzeugung und Primzahltests
3. Angriffe auf das RSA Verfahren
4. Sicherheit von RSA

Probleme symmetrischer Verschlüsselung

- **Schlüsselverteilungsproblem**
  - Sender und Empfänger müssen den gleichen Schlüssel verwenden
  - Schlüssel muß vor der Kommunikation ausgetauscht werden
  - Zwischen beiden Teilnehmern muß ein sicherer Kanal existieren, was über größere Distanzen kaum möglich ist
  - Anwendungen verlangen spontanen Aufbau sicherer Verbindungen
- **Es sind zu viele Schlüssel erforderlich**
  - Wenn jeder mit jedem sicher kommunizieren will, braucht man bei n Teilnehmern $n(n-1)/2$ Schlüssel und zum Schlüsselaustausch eine gleichgroße Anzahl sicherer Kanäle
  - Bei 10^9 Internetnutzern braucht man 10^{18} Schlüssel/Kanäle
  - Organisatorisch nicht zu bewältigen
- **Kommunikation über sichere Zentrale?**
  - Führt zu Engpässen und Gefahr von Sicherheitslöchern in Zentralstelle

Public-Key Kryptographie ist flexibler

- **Schlüsselmanagement wird einfacher**
  - Ver- und Entschlüsselung benutzen verschiedene (inverse) Schlüssel
  - Empfänger erzeugt beide Schlüssel, legt Verschlüsselungsschlüssel offen
  - Jeder Teilnehmer kann einen sicheren Kanal zum Empfänger aufbauen
  - Pro Empfänger nur ein Schlüssel erforderlich
  - Schlüssel werden in öffentlichem Verzeichnis gelagert oder bei Bedarf vom Empfänger erzeugt
- **Wichtige Randbedingungen**
  - Privater Schlüssel des Empfängers darf nicht aus dem öffentlichen Schlüssel berechnet werden können
  - Es muß leicht sein, viele (gute) Schlüssel schnell zu erzeugen
  - Öffentlicher Schlüssel muß vor Fälschungen geschützt werden
- **Public-Key Verfahren sind langsamer**
  - AES ist etwa 1000 mal schneller als asymmetrische Verfahren
  - Praxis verwendet hybride Verfahren

Sicherheit von Public-Key Verfahren

- **Sicherheit des privaten Schlüssels**
  - Privater Schlüssel nicht aus öffentlicher Information zu berechnen
  - Sicherheitsbeweise sind Reduktionen auf schwierige mathematische Probleme, da Nachweis der "Unbrechbarkeit" nicht möglich
  - Berechnungsprobleme der Zahlentheorie liefern gute Verfahren
    Faktorisierung, Diskreter Logarithmus, Elliptische Kurven
- **Semantische Sicherheit**
  - Wahrscheinlichkeit, daß Angreifer Chiffrierung eines Klartextes von der eines beliebigen Strings unterscheiden kann, ist maximal 50%
  - Macht Public-Key Verfahren sicher gegen Ciphertext-only Angriffe
- **Adaptive-Chosen-Ciphertext Sicherheit**
  - Wahrscheinlichkeit, daß Angreifer einen gegebenen Chiffretext entschlüsseln kann, wenn er die Klartexte einer beliebigen Menge anderer Schlüsseltexte kennt, ist maximal 50%
  - Macht Verfahren sicher gegen Verfälschungen von Nachrichten

Kryptographie und Komplexität

Einheit 4.1

Das Public-Key Verfahren von Rivest, Shamir und Adleman (RSA)

1. Verschlüsselungsverfahren
2. Zahlentheoretischer Hintergrund
3. Schnelle Ver-/Entschlüsselung

Public-Key Kryptographie mit dem RSA Verfahren

- **Ältestes und bedeutendstes Public-Key Verfahren**
  - Benannt nach Ron Rivest, Adi Shamir und Len Adleman (1977)
  - Sicherheit basiert auf Schwierigkeit des Faktorisierungsproblems
    Zerlegung großer Zahlen in Faktoren ist nicht in akzeptabler Zeit möglich
- **Verwendet weiterhin Modulararithmetik**
  - Multiplikation und Potenzierung sehr großer Zahlen ($>$ 100 Stellen)
- **Verwendet bekannte Gesetze der Zahlentheorie**
  - Ist $gcd(x,n) = 1$, so folgt $x^{\varphi(n)}$ mod $n = 1$ (Satz von Euler-Fermat)
    Konsequenz: Ist $e*d$ mod $\varphi(n) = 1$ und $x<n$, so ist $(x^e)^d$ mod $n = x$
  - Potenzierung mit e bzw. d liefert ein einfaches Public-Key Verfahren
  - Sicherheit: Um den privaten Schlüssel d aus e und n zu berechnen, muß man $\varphi(n)$ bestimmen, also n in Primfaktoren zerlegen können

Das RSA verfahren im Detail

- **Schlüsselerzeugung**
  - Generiere n als Produkt zweier großer Primzahlen p und q (z.B. 512 bit)
    In diesem Fall ist $\varphi(n) = (p{-}1)(q{-}1)$
  - Erzeuge zufälliges e mit $gcd(e, \varphi(n)){=}1$ und berechne $d = e^{-1} \bmod \varphi(n)$
  - Setze $n = p * q$ und lege n, e offen, halte d, p und q geheim
- **Verschlüsselungsverfahren**
  - Gesamtschlüssel ist $K := (n, p, q, d, e)$, wobei e, n öffentlich
  - Text wird in Blöcke der Länge $\log_2 n/8$ zerlegt (ein Byte pro Buchstabe)
    Jeder Textblock wird als Binärdarstellung einer Zahl x interpretiert
  - Verschlüsselung wird Potenzieren mit e modulo n: $\boldsymbol{e_K(x)} = x^e \bmod n$
  - Entschlüsselung wird Potenzieren mit d modulo n: $\boldsymbol{d_K(y)} = y^d \bmod n$
- **Benötigt schnelle Potenzierung großer Zahlen**
  - e-fache Multiplikation mit sich selbst ist indiskutabel
  - Laufzeit muß in Größenordnung der Anzahl der Stellen von e liegen

Das RSA Verfahren am Beispiel

- **Einfaches Zahlenbeispiel**
  - Wähle $p = 13$ und $q = 17$, also $n = 221$
  - Wähle $e = 5$. Dann muß $d = 77$ sein $(5 * 77 \bmod 192 = 1)$
  - Verschlüsselung der Zahlen 4 5 6 7 8 9 mit e ergibt 140 31 41 11 60 42
    Zeigt gute statistische Streuung der Schlüsseltexte
  - Entschlüsselung von 8 9 10 11 12 mit d ergibt 60 42 147 7 116
- **Realistische Blocklänge ist 256, 512 oder 1024 Bit**
  - Textblöcke von 32, 64 oder 128 Bytes werden als Zahlen interpretiert
  - Generierte Primzahlen p und q müssen je 38/77/155 Stellen haben
    Benötigt schnelle Primzahltests für sehr große Zahlen
  - Die Zahlen n, e und d haben jeweils 77/155/310 Stellen
  - Ver-/Entschlüsselung ist Potenzierung mit riesigen Zahlen
    Naive Algorithmen sind linear in e und d
  - Resultierende Zahl wird als Byte-Kette interpretiert / versendet

Beispiel für RSA Verfahren mit 256 Bit

- **Generiere Primzahlen p und q mit Zufallsgenerator**
  - Generiere $p = 200090614988854752464080544630480762821$
    und $q = 185943990722218737951811440044937739573$
  - Berechne $n = p{*}q$:

  $n =$ 37205647457090649636885952943039603307528960641855571381455020277436478815433
- **Wähle e und $d = e^{-1} \bmod \varphi(n)$**

  $e =$ 91225117934565043354694670473901491802882927843894977203888465620701016614749

  $d =$ 10031874553482861914621375056580729784388315106392476555458055333137857251589
- **Transformiere Text in 256-Bit Zahl (77 Stellen)**
  - Ausgangstext `"  INFORMATIK BRAUCHT MATHEMATIK  "`

  $x =$ 14603531222158190878074976722480774803100880121174407015018204085926251667488
- **Verschlüssele x mit e zu $y = x^e \bmod n$**

  $y =$ 58295177558062215800130868212004345138747431877348356745826608826099396346890

DAS RSA VERFAHREN - WICHTIGE FRAGEN

- **Warum ist das Verfahren korrekt?**
  – Was sind die mathematischen Grundlagen für eine Ver- und Entschlüsselung mittels Potenzierung?
- **Wie bestimmt man effizient einen Schlüssel?**
  – Wie effzient können Primzahltests für zufällige Zahlen sein?
- **Wie schnell kann ver-/entschlüsselt werden?**
  – Schneller als wiederholtes Multiplizieren modulo n?
- **Wie sicher ist das Verfahren?**
  – Um privaten Schlüssel zu bestimmen, muß man n faktorisieren können
  – Wie effizient kann man eine sehr große Zahl faktorisieren?
  – Wieviel Bits sind erforderlich, damit RSA sicher ist gegen die besten Faktorisierungsalgorithmen und massive Parallelverarbeitung?

Mathematik: Zyklische Gruppen

- **RSA basiert auf Eigenschaften von Gruppen**
  - Potenzierung in $\mathbb{Z}_n$ ist eine iterierte Gruppenoperation: $x^e = \underbrace{x \cdot x \cdot \ldots \cdot x}_{e\ mal}$
  - Entschlüsselung benötigt eine Zahl d mit $x^{e \cdot d - 1} = 1$ in $\mathbb{Z}_n$
- **Ordnung eines Gruppenelements $g \in G$**
  - **order$_G(g)$**: kleinste Zahl e mit $g^e = 1$ in $(G, \cdot)$
  - In ($\mathbb{Z}_{27}$,+) ist order(2)=27, order(3)=9, order(4)=27, ...
  - In ($\mathbb{Z}^*_{27}$,·) ist order(2)=18 und order(4)=9 (3 gehört nicht zu $\mathbb{Z}^*_{27}$)

    Im Ring R ist R^* die Menge der bzgl. · invertierbaren Elemente
- **Von $g \in G$ erzeugte Untergruppe**
  - Menge $\langle g \rangle := \{g^k \mid k \leq \text{order}_G(g)\}$ zusammen mit der Verknüpfung $\circ_G$
  - G heißt **zyklisch**, wenn $G = \langle g \rangle$ für ein $g \in G$
  - ($\mathbb{Z}_{27}$,+) und ($\mathbb{Z}^*_{27}$,·) sind zyklisch mit Erzeuger 1 bzw. 2
  - ($\mathbb{Z}_n$,·) ist immer zyklisch, wenn n eine Primzahl ist — Beweis später

MATHEMATIK: ZYKLISCHE GRUPPEN (II)

- **$g^e{=}1$ genau dann, wenn $\text{order}_G(g)$ Teiler von e**

  $\Rightarrow$: Es sei $n{=}\text{order}_G(g)$, $g^e = 1$ und $e = q{\cdot}n + r$.

  Dann ist $g^r = g^e{\cdot}g^{-nq} = 1{\cdot}1 = 1$.

  Da n die kleinste Zahl mit $g^n = 1$ ist und $r{<}n$, muß $r = 0$ sein.

  $\Leftarrow$: Es sei $\text{order}_G(g)$ Teiler von e. Dann ist $e = k{\cdot}n$ für ein k,

  also $g^e = (g^n)^k = 1^k = 1$.

- **$g^x{=}g^y$ genau dann, wenn $x \equiv y \bmod \text{order}_G(g)$**
  - Folgt direkt aus obigem Satz mit $e = x - y$

- **Für $e = \text{order}_G(g)$ gilt $\text{order}_G(g^k) = e/gcd(e,k)$**
  - Es sei $k \in \mathbb{N}$ beliebig. Dann $(g^k)^{e/gcd(e,k)} = (g^e)^{k/gcd(e,k)} = 1$.
  - Damit ist $l{:=}\text{order}_G(g^k)$ Teiler von $e/gcd(e,k)$
  - Umgekehrt folgt aus $1 = (g^k)^l = g^{kl}$, daß $e/gcd(e,k)$ Teiler von l ist.

Mathematik: Zyklische Gruppen (III)

- **Jede endliche zyklische Gruppe G hat genau $\varphi(|G|)$ Erzeuger und jeder hat die Ordnung $|G|$**
  – Ist $e = \text{order}_G(g)$ für ein $g \in G$, so gilt $|\langle g\rangle| = e$.
  – Damit sind die Elemente g der Ordnung $|G|$ genau die Erzeuger von G und jeder Erzeuger g' von G beschreibbar als $g' = g^k$ für ein $k \leq |G|$
  – Wegen $|G| = \text{order}_G(g^k) = \text{order}_G(g)/gcd(|G|, k)$ folgt $gcd(|G|, k) = 1$.
  – Damit entsprechen die Erzeuger den zu $|G|$ teilerfremden Zahlen.

- **Ist U Untergruppe von G so ist $|U|$ Teiler von $|G|$**

  Satz von Lagrange

  – Definiere $a \equiv b$ g.d.w. $a \circ b^{-1} \in U$. Dann ist $\equiv$ eine Äquivalenzrelation.
  – Für die zugehörige Äquivalenzklassen $[a] = \{b \mid a \equiv b\} = \{u \circ a \mid u \in U\}$ gilt $|[a]| = |U|$, weil $f : U \rightarrow [a]$ mit $f(u) = u \circ a$ bijektiv ist.
  – Da G disjunkte Vereinigung aller Äquivalenzklassen ist, muß $|U|$ Teiler von $|G|$ sein.

Der kleine Satz von Fermat

Ist $gcd(x,n) = 1$, so folgt $x^{\varphi(n)} \bmod n = 1$

- **Für jedes $g \in G$ ist $\mathrm{order}_G(g)$ Teiler von $|G|$**
  - $\langle g \rangle$ ist Untergruppe von G der Ordnung $\mathrm{order}_G(g)$
  - Nach dem Satz von Lagrange ist somit $\mathrm{order}_G(g)$ Teiler von $|G|$
- **Für jedes $g \in G$ ist $g^{|G|} = 1$**
  - Folgt aus der Tatsache, daß $\mathrm{order}_G(g)$ Teiler von $|G|$ ist
- **Ist $gcd(x,n) = 1$, so folgt $x^{\varphi(n)} = 1$ in $\mathbb{Z}_n^*$**
  - Gilt $gcd(x,n) = 1$ so ist $x \in \mathbb{Z}_n^*$. Die Ordnung von $(\mathbb{Z}_n^*, \cdot)$ ist $\varphi(n)$.
- **Korollar: Ist $e{*}d \bmod \varphi(n) = 1$ und $x<n$, so gilt $(x^e)^d \bmod n = x$**
  - Es sei $e{*}d = k * \varphi(n) + 1$. Dann gilt

$$(x^e)^d \bmod n = x^{e*d} \bmod n = x^{k*\varphi(n)+1} \bmod n = x * x^{k*\varphi(n)} \bmod n$$
$$= x * (x^{\varphi(n)})^k \bmod n = x * 1^k \bmod n = x$$

Mathematik: Schnelle Potenzierung

- **Naive Potenzierung x^e mod n ist in $\mathcal{O}(n\cdot\|n\|^2)$**
  - Undurchführbar für große n
- **Quadrieren und Multiplizieren**
  - Ist $e = \sum_{i=0}^{k} e_i 2^i$ die Binärentwicklung von e so ist $x^e = \prod_{i=0}^{k} x^{e_i 2^i}$
  - Weil die e_i nur 0 oder 1 sein können, folgt $x^e = \prod_{i\leq k, e_i=1} x^{2^i}$
  - Wegen $x^{2^{i+1}} = (x^{2^i})^2$ ist x^e durch sukzessives Quadrieren zu berechnen

    z.B. $4^{11} \bmod 27 = (4^5)^2_{27}\cdot_{27}4 = ((4^2_{27})^2_{27}\cdot_{27}4)^2_{27}\cdot_{27}4$

    $= (16^2_{27}\cdot_{27}4)^2_{27}\cdot_{27}4 = (13\cdot_{27}4)^2_{27}\cdot_{27}4 = 25^2_{27}\cdot_{27}4 = 4\cdot_{27}4 = 16$
- **Funktionale Implementierung**

```
let rec pow x e n
= if e = 0 then 1
  else let r = pow (x*x mod n) (e/2) n
       in if e mod 2 = 0 then r else r*x mod n;;
```

  - Laufzeit ist $\mathcal{O}(\|n\|^3)$, da nur $k = \|e\|$ Multiplikationen/Quadrierungen

Der Chinesische Restsatz

Die simultane Kongruenz $\forall i{\leq}k.\ x\equiv a_i \bmod m_i$
hat eine eindeutige Lösung modulo $m{=}\prod_{i=1}^k m_i$,
wenn alle m_i paarweise teilerfremd sind

Konstruktion: Sei $M_i = \prod_{j\neq i} m_j$. Dann gilt $gcd(m_i, M_i) = 1$ und
$y_i = M_i^{-1} \bmod m_i$ kann mit `egcd` berechnet werden.
Setze $x = (\sum_{i=1}^k a_i y_i M_i) \bmod m$.

Korrektheit: Wegen $a_i y_i M_i \bmod m_i = a_i$ und $a_i y_i M_i \bmod M_i = 0$ folgt
$x \equiv (a_i y_i M_i + \sum_{j\neq i} a_j y_j M_j) \bmod m_i = a_i \bmod m_i$ für alle i

Eindeutigkeit: Ist x' eine Lösung der simultanen Kongruenz so gilt
$\forall i{\leq}k. x \equiv x' \bmod m_i$ und somit $x \equiv x' \bmod m$.

Laufzeit: Berechnung von m kostet Zeit $\mathcal{O}(\|m\|\cdot \sum_{i=1}^k \|m_i\|) = \mathcal{O}(\|m\|^2)$.
Berechnung eines M_i aus m und eines y_i liegt in $\mathcal{O}(\|m\|\cdot\|m_i\|)$.
Berechnung von x benötigt $\mathcal{O}(\|m\|\cdot \sum_{i=1}^k \|m_i\|) = \mathcal{O}(\|m\|^2)$.
Gesamtlaufzeit ist $\mathcal{O}(\|m\|^2)$ bei Platzbedarf $\mathcal{O}(\|m\|)$.

Liefert schnelle Lösung simultaner Kongruenzen

Schnelle Entschlüsselung

- **Reduziere absolute Schrittzahl um 75%**
  - Absolute Rechenzeit kritisch für Chipkarten und ähnliche Hardware
  - 512 Quadrierungen + 256 Multiplikationen für 512 Bit ist zu viel
- **Verwende den Chinesischen Restsatz**
  - Wegen $n = p{\cdot}q$ berechne $x_p = y^d \bmod p$ und $x_q = y^d \bmod q$
  - Laufzeit ist jeweils ein Achtel der Berechnungszeit für $x = y^d \bmod n$
  - Löse simultane Kongruenz $x \equiv x_p \bmod p \ \wedge \ x \equiv x_q \bmod q$

    Dann gilt $x \equiv y^d \bmod p$ und $x \equiv y^d \bmod q$ also $x \equiv y^d \bmod n$
  - Lösung der Kongruenz erfordert nur zwei Multiplikationen, weil $y_1 = p^{-1} \bmod q$ und $y_2 = q^{-1} \bmod p$ statisch berechnet werden können
  - Gesamtlaufzeit ist somit viermal schneller als Potenzierung modulo n

Korrektheit und Komplexität von RSA

- **Korrektheit: Ver-/Entschlüsselung sind invers**
  - Weil e, d so gewählt werden, daß $e{\cdot}d \bmod \varphi(n) = 1$ ist, gilt
    $d_K(e_K(x))$ = $(x^e$ mod $n)^d$ mod n = $(x^e)^d$ mod n = x
- **Aufwand für Auswahl des Schlüssels (einmalig)**
  - Erzeugung der Primzahlen p, q und von $n = p{\cdot}q$ $\mathcal{O}(\|n\|^3)$
  - Wahl von e und Berechnung von $d = e^{-1} \bmod \varphi(n)$ $\mathcal{O}(\|n\|^2)$

  Mehr dazu in §4.2
- **Aufwand für Ver- und Entschlüsselung**
  - Kein Aufwand für Umwandlung zwischen Text und Zahlen
  - Potenzierung von $8|w|/\|n\|$ Blöcken $\mathcal{O}(|w|{\cdot}\|n\|^2)$
  - Blocklänge geht quadratisch in Laufzeit ein
- **Sicherheit des geheimen Schlüssels**
  - Bestimmung von d ist genauso schwer wie Faktorisierung von n
  - Faktorisierung braucht i.w. exponentiell viele Schritte in $\|n\|$

  Mehr dazu in §4.3