UNIVERSITÄT
LEIPZIG

Algorithmen und Datenstrukturen II

Vorlesung *do1spre2ece*

Leipzig, 18.06.2024
Peter F. Stadler & Thomas Gatter & Ronny Lorenz

ZAHLENTHEORIE

Zahlentheorie und Kryptographie

Public-Key Kryptographie mittels RSA (Rivest, Shamir, Adleman)

- Verschlüsseln von Nachrichten zwischen zwei Parteien
  - Geheime Schlüssel zum Entschlüsseln
  - Öffentliche Schlüssel zum Verschlüsseln
- Signaturen für Nachrichten
  - leicht zu verifizieren
  - nicht fälschbar
  - kleinste Änderungen in Nachricht erkennbar

Lustige Warnung: es gibt *keinen Beweis*, dass dieses Verfahren sicher ist

Zahlentheorie und Kryptographie

Ein Kommentar vorweg:
Krypto ist kompliziert wenn man sie korrekt implementieren möchte. Nutzen sie bestehende, aktuelle Libraries.

Diese VL folgt dem Kapitel im Cormen:
Introduction to Algorithms, Cormen et al, *Number-Theoretic Algorithms*

Grundlagen

- große Eingaben für diese VL-Einheit sind groß im Bezug zur Anzahl an Bit die nötig sind, um die Eingabe zu kodieren
- wir reden typischerweise über eine Integer-zahl, jedoch hat diese 512 oder mehr Bit

Die Anzahl der *Bit* gibt die Schlüssellänge an. Es wird zwischen **symmetrischen** und **asymmetrischen Verfahren** unterschieden.

→ Symmetrische Verfahren haben nur einen Schlüssel (AES, Blowfish, etc).

→ Asymmetrische Verfahren haben getrennte Schlüssel zum ver- und entschlüsseln.

Der Teil zum verschlüsseln kann öffentlich verfügbar sein.

Grundlagen I

- eine Zahl $p \in \mathbb{N} = \{0, 1, \dots\}$ ist eine **Primzahl**, wenn 1 und p die einzigen Teiler von p sind
- $\mathbb{P} = \{2, 3, 5, 7, 11, 13 \dots\} \subset \mathbb{N}$ ist die Menge aller Primzahlen
- wir schreiben $d|a$ ("d teilt a"), falls $\exists k \in \mathbb{Z} : a = kd$
  Bsp: 4|8 da $2 \in \mathbb{Z}$ und 8=2*4 gilt
- **Equivalenzklasse Modulo** n: $[a]_n = \{a + kn : k \in \mathbb{Z}\}$
  Bsp: $n = 3$: $[1, 4, 7, d \dots]_3 = \{1 + k * 3\} = \{1 + 0 * 3, 1 + 1 * 3, 1 + 2 * 3, \dots\}$
- wir schreiben $a \equiv b \mod n$ falls $a = qn + r$ und $b = q'n + r$
  Bsp: $4 \equiv 7 \mod 3$, da $4 = 1 * 3 + 1$ und $7 = 2 * 3 + 1$

Grundlagen II

- jede Natürliche Zahl hat eine Einzigartige **Faktorisierung**:
  $\forall a \in \mathbb{N} : a = p_1^{e_1} \cdot \ldots \cdot p_k^{e_k}, p_i \in \mathbb{P}$
- RSA basiert auf der Faktorisierung von Primzahlen
- wir benötigen eine Möglichkeit einen gemeinsamen Faktor zweier Zahlen zu finden: **g**rößter **g**emeinsamer **T**eiler $d = ggT(a, b) = ax + by$.
- ggT kann mit dem Algorithmus von Euklid effizient bestimmt werden

Grundlagen III

Erweiterter Euklidischer-Algorithmus

Euklid(*a*,*b*):
  if $b = 0$ **then**
    Return (a,1,0)
  (d,y',x) = **Euklid**(b, $a \mod b$)
  (d,x,y) = (d, x, $y' - \lfloor a/b \rfloor x$)
  Return (d,x,y)

– Der erweiterte Algorithmus bestimmt nicht nur den größten gemeinsamen Teiler d, sondern auch x und y, so dass $d = ggT(a, b) = ax + by$.

Grundlagen IV

Euklid wird so lange rekursiv aufgerufen bis $b = 0$. Beachten Sie die Rekursion in Zeile 3. Deren Ergebnis wird in Zeile 4 benutzt.

- mit $a > b \geq 1$ und $b < F_{k+1}$ werden $< k$ rekursive Aufrufe durchgeführt
- Laufzeit: $O(\beta)$ arithmetische Operationen, $O(\beta^3)$ Bitoperationen für zwei β-Bit enkodierte Zahlen
  - Multiplikation benötigt $O(\beta^2)$ Bitoperationen
  - Geht das auch schneller?
- Korrektheit beruht auf den folgenden zwei Eigenschaften:
  (i) $ggT(a, b) = ggT(b, a \mod b)$ für $a \neq 0$
  (ii) $ggT(a, 0) = a$

Beispiel: Euklid

Beispiel: Euklid

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3	-11	14
78	21	3	3	3	-11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	-	3	1	0

Nehmen sie die Werte $a = 99$ und $b = 78$
und rechnen sie `Euklid` auf Papier nach!

Teilerfremde Zahlen

- Erinnerung: `Euklid(a,b).d` gibt den ggT zurück
- *a* und *b* sind teilerfremd falls `Euklid(a,b).d` = 1
- wir sagen auch *a* und *b* sind **relativ Prim**
- falls *a*, *b* teilerfremd zu *p*, dann ist $a * b$ teilerfremd zu *p*

- Beispiel: 8 hat Teiler 1,2,4,8 und 15 hat Teiler 1,3,5,15.
- Falls *a*, *b* teilerfremd zu *p*, dann ist $a \times b$ teilerfremd zu *p*.
  - Warum?

Uhren- oder Modulo Arithmetik

Gruppe:

- Gruppe $(S, \oplus)$ mit Menge S und binärer Operation $\oplus$ auf S.
- *Abgeschlossen:* $\forall a, b \in S : a \oplus b \in S$
- Id: $\exists e \in S : e \oplus a = a \oplus e = a$
- *Assoziativ:* $\forall a, b, c \in S : (a \oplus b) \oplus c = a \oplus (b \oplus c)$
- *Inverses:* $\forall a \in S : \exists$ (ein) $b \in S : a \oplus b = b \oplus a = e$

Beispiel: $(\mathbb{Z}, +)$, ganze Zahlen mit Addition, $e = 0$, Inverses: $-a$.

Modulo Arithmetik

Endliche Gruppe:

- Sei n eine natürliche Zahl
- $\mathbb{Z}_n$ sei die Menge der Zahlen $\{0 \dots n-1\}$
- Darauf lassen sich zwei nützliche Gruppen definieren:
  - $\oplus = +$: $(\mathbb{Z}_n, +_n)$
  - $\oplus = \times$: $(\mathbb{Z}_n, \times_n)$
- Sei $a \equiv a' \mod n$, $b \equiv b' \mod n$. In der jeweiligen Gruppe gilt:
  $\rightarrow \quad a + b \equiv a' + b' \mod n$
  $\rightarrow \quad ab \equiv a'b' \mod n$
- Multiplikative Gruppe modulo n: $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n : \text{ggT}(a, n) = 1\}$

Beispiel: Gruppen Modulo n

$a + b \mod 3$

$+_3$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$a * b \mod 3$

$*_3$	1	2
1	1	2
2	2	1

Sehen sie die *neutralen Elemente* und die *inversen Elemente?*

Eulers Φ (Phi)

$\Phi(n)$ zählt die natürlichen Zahlen $\leq n$, die teilerfremd zu n sind

$$\Phi(n) = n \prod_{p:p\in\mathbb{P}\wedge p|n} \left(1 - \frac{1}{p}\right)$$

- Falls $p \in \mathbb{P}$ dann $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$
- und $\Phi(p) = p - 1$
- Falls $p \notin \mathbb{P}$ dann $\Phi(n) < n - 1$

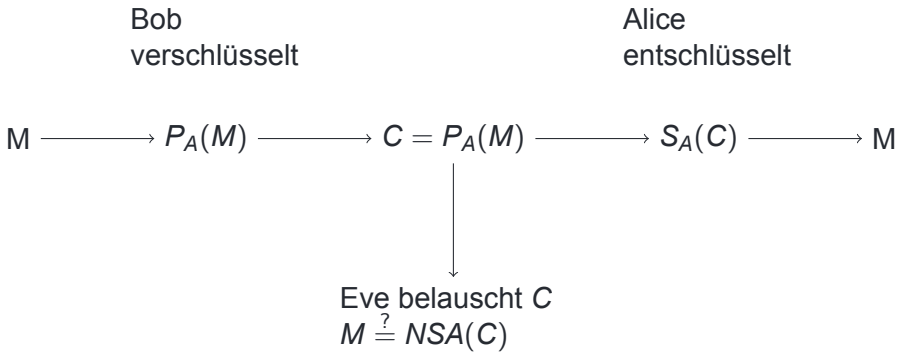
Euler's Φ liefert die Größe von Z_n^* für ein gebenes n. Konkreter: Von $\{1, \ldots, n\}$ behalten Sie nur die Primzahlen die Teiler von n sind. Dann rechnen sie das Produkt aus mit den Faktoren $(1 - 1/p)$ und multiplizieren sie mit n.

PUBLIC KEY INFRASTRUCTURE

Ein paar Begriffe und Definitionen

- Alice und Bob wollen kommunizieren
- Eve (“Eavesdropper”) möchte lauschen (Eve arbeitet für die NSA)
- **Öffentlicher Schlüssel:** P (Alice: P_A, Bob: P_B)
- **Geheimer Schlüssel:** S (Alice: S_A, Bob: S_B)
- $P_A(\cdot)$ (etc) *seien die entsprechenden Funktionen*
- Sei $M \in \mathcal{D}$ die zu sendende Nachricht
- Es gelte:
  - $M = S_A(P_A(M))$
  - $M = P_A(S_A(M))$
- Wir hoffen: S_A kann nur von Alice in vertretbarer Zeit berechnet werden!

PKI Kryptographie



Es gibt keine bekannte schnelle Funktion NSA **die** C **in** M**, ohne Kenntnis von** S_A**, umwandelt**

Das RSA Kryptosystem

In sechs Schritten zu sicherer Kommunikation:

1. **Wähle zufällige Primzahlen** p, q, $p \neq q$, beide $\geq$ 512–2048 bit
   $p = 11, q = 13$
2. **Berechne** $n = pq$
   $n = 143$
3. **Wähle** e**, ungerade und klein** (z.B. mit 16 bit: $2^{16} + 1 = 65537$)**, relativ prim zu** $(p-1)(q-1) = \Phi(n)$
   $e = 23$ (rel. prim zu 120)
4. **Berechne** d mit $de \equiv 1 \mod (p-1)(q-1) = \Phi(n)$
   $d = 47, 47 * 23 \mod 120 = 1$
   `Euklid` ...
5. **Öffentlicher Schlüssel:** $P = (e, n)$
   $P = (23, 143)$
6. **Geheimer Schlüssel:** $S = (d, n)$
   $S = (47, 143)$

Beispiel

Beispiel

1. **Verschlüsseln** von $m = 7$
2. $c \equiv m^e \mod n \quad 2 \equiv 7^{23} \mod 143$
3. **Entschlüsseln** von $c = 2$
4. $m \equiv c^d \mod n \quad 7 \equiv 2^{47} \mod 143$

Es gibt nicht so viele kleine Primzahlen, dass sich viele Beispiele finden ließen.

Die Sicherheit von RSA baut darauf, dass n (hier 143) nicht einfach in die beiden Primzahlen p, q zerlegt werden kann. Dafür existiert kein Beweis! *(Link1) (Link2)*

Korrektheit von RSA

Beweis

Wir können *nicht* zeigen, dass RSA sicher ist. Aber es ist zu zeigen das RSA korrekt arbeitet. Also Ver- und Entschlüsselung zusammen die originale Antwort geben.

- für alle $m \in \mathcal{D}$ gilt:
- $P(S(m)) = S(P(m)) = m^{ed} \mod n$
- $ed = 1 + k(p-1)(q-1)$
- $m^{ed} = m^{ed-1}m = m^{k(p-1)(q-1)}m = (m^{p-1})^{k(q-1)}m$
  $\equiv 1^{k(q-1)}m \equiv m \mod p$
- analog für $\mod q$
- damit auch für $\mod pq = n$

RSA und Primzahlen

- **RSA basiert darauf,** dass es *schwer* ist n in p, q zu faktorisieren (mit p, q zwei sehr großen (> 512 Bit) Primzahlen)
- um aber RSA nutzen zu können, müssen wir p, q haben, und niemand sonst darf diese Zahlen kennen

Das heißt aber, wir müssen testen, dass p, q Primzahlen sind
…indem wir versuchen p, q zu faktorisieren?

→ **Nicht nötig:** Wir können einfach zufällige Zahlen auf *"prim sein"* testen. Haben wir zwei, sind wir fertig.
- und diese Prozedur hilft *nicht* beim Faktorisierungsproblem von n!

Finden von Primzahlen

Primzahltheorem:

- $\pi(n) =$ (Anzahl der Primzahlen $\leq n$)
- man kann zeigen, dass $\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1$
- die Wahrscheinlichkeit, dass ein zufäliges $k \in \mathbb{N}$ prim ist: $1/\ln n$
- falls k prim, dann: $a^{k-1} \equiv 1 \mod k \quad (\forall a \in \{1 \dots k-1\})$
- $a = 2$ allein reicht als Test *fast* aus: die Fehlerrate ist 10^{-20} bei 512-bit Zahlen!
- allerdings gibt es sog. *Carmichael*-Zahlen bei denen der Test immer versagt (also falsch "prim" ausgibt)
- dort hilft z.B. der Miller-Rabin Test weiter

$\rightarrow$ Wir können also sehr leicht große Primzahlen finden
(durch zufällige Wahl eines k und dann Primzahltest)

Beispiel

- $a = 2 \quad k = 47 \quad 2^{46} \equiv 1 \mod 47$
- $a \in \{2 \ldots 46\}: \quad [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,$
  $1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$
- $a = 2 \quad k = 49 = 7 * 7 \quad 2^{48} \equiv 15 \mod 49$
- $a \in \{2 \ldots 48\}: \quad [15, 43, 29, 43, 8, 0, 43, 36, 8, 15, 22, 15, 0,$
  $36, 8, 22, 1, 1, 22, 0, 29, 29, 36, 36, 29, 29,$
  $0, 22, 1, 1, 22, 8, 36, 0, 15, 22, 15, 8, 36, 43, 0, 8, 43, 29, 43, 15, 1]$

Wir müssen allerdings nur für $a = 2$ testen um mit großer Sicherheit k als Prim bestimmen zu können

Faktorisierungsalgorithmen

- es gibt Algorithmen zur Primfaktorzerlegung, die effizienter arbeiten als *brute force*
- der **Pollard-ρ (Rho) Algorithmus** ist ein Beispiel
- der benötigte Hauptspeicher ist gering
- Wenn $n = p_i \times \cdots \times p_k$, dann ist die Laufzeit proportional zu $\sqrt{\min p}$ (also zur Wurzel des kleinsten Primfaktors)
- deshalb sollten die Primfaktoren p, q für RSA nicht zu unterschiedlich gross sein

(Algorithmus auf Wikipedia)

Faktorisierung von $n = pq$: Pollard-Rho

Pollard-Rho Algorithmus

$i := 1 \quad x_1 := \text{Rand}\,(0, n-1) \quad y := x_1 \quad k := 2$;
while *True* **do**
    $i := i + 1$;
    $x_i := x_{i-1}^2 - 1 \mod n$;
    $d := \text{ggT}\,(x_i - y, n)$;
    if $d \neq 1$ *und* $d \neq n$ **then**
        return d ;
        `/* Wenn alle Primfaktoren gewünscht sind, sollte hier print d stehen, dann ''terminiert'' diese Variante allerdings nicht */`
    if $i = k$ **then**
        $y := x_i$;
        $k := 2k$;

erwartete Zeit: $O(n^{1/4})$
... und damit *exponentiell* in β da $\beta = \lceil \log_2 n \rceil$

Beispiel: Pollard-Rho

Beispiel

Sei $n = 323 = 19 * 17$

i	$x_{i+1} = (x_i^2 - 1)$	$x_{i+1} \mod 323$	d	y
1	-	2	-	2
2	3	3	1	3
3	8	8	1	3
4	63	63	1	63
5	3968	92	1	63
6	8463	65	1	63
7	4224	25	**19**	63

Zusammenfassung

- Das RSA-Kryptosystem basiert auf einfacher Zahlentheorie
- Das Finden von zufälligen Primzahlen ist einfach
- Es gibt keine bekannte, schnelle Methode $n = pq$ zu faktorisieren
- Es gibt aber auch keinen Beweis der Sicherheit (!)
- RSA wird typischerweise benutzt um einen asymmetrischen Session-Key zu verschlüsseln
- “Basis”-RSA hat einige Schwachpunkte, die allerdings in guten Implementationen nicht zum Tragen kommen:
  Es gilt aber: Bauen Sie sich RSA nicht selbst!