Kryptologie (beliebig lang)

20. September, 2025; rev. 16. Januar 2026
Linda Riesen (rieselin)

1 Basics

1.1 Definitions

- (G,*) Gruppe:
  - Assozitativität $(a * b) * c = a * (b * c)$ für alle $a, b, c \in G$
  - Neutralelement Es existiert ein Element $e \in G$ mit $e * a = a$ für alle $a \in G$
  - Inverses Für jedes $a \in G$ existiert ein Inverses a^{-1} mit der Eigenschaft: $a * a^{-1} = e$
  - Abgeschlossenheit $a * b \in G$ für alle $a, b \in G$
- Untergruppe: U ist eine Untergruppe von G, wenn:
  - es sich um eine Teilmenge handelt, und
  - U selber eine Gruppe bildet.
  - Ist U eine Untergruppe von G, so gilt: $|U|$ teilt $|G|$.
  - Für jedes Element $g \in G$ heisst $< g >:= \{1, g, g^2, g^3, ...\}$ die von g erzeugte Untergruppe
    - $| < g > | = min\{k : g^k = 1\}$
    - $| < g > |$ teilt $|G|$
    - $g^{|G|} = 1$
- Abelsche Gruppe: Eigenschaften (G,*) Gruppe + Kommutativität
- Gruppe $\mathbb{Z}_n^*$:
  - Die Elemente sind alle zu n teilerfremden Zahlen in $\{1, 2, ..., n-1\}$
  - Die Gruppen-Operation entspricht der Multiplikation
- $\varphi(n) := |\mathbb{Z}_n^*|$
  - $\varphi(u * v) = \varphi(u) * \varphi(v)$ mit $ggT(u, v) = 1$
  - $\varphi(p) = p - 1$ mit p ist Primzahl
  - $\varphi(p^k) = p^k - p^{k-1}$ mit p ist Primzahl
- Ordnung eines Elements einer Gruppe Definition Die Ordnung eines Elementes g bezeichnet $| < g > |$.
- Satz von Euler: $a^{\varphi(n)} = 1 (\text{mod } n)$ für alle $a \in \mathbb{Z}_n^*$
- kl Satz von Fermat gilt für jd Primzahl p: $a^{p-1} = 1 (\text{mod } p)$ für alle $a \in \mathbb{Z}_n^*$
- Zyklishe Gruppe: falls es ein $g \in G$ gibt mit: $G = 1, g, g^2, g^3, ...\}$
  - g heisst **Erzeugendes** oder **Primitivwurzel**
    - Falls $g^{\frac{n}{p}} \neq 1$ für jeden Primteiler p von n (n=$|G|$), so ist g ein Erzeugendes
    - Beispiel: p = 467 => erzeugendes suchen: $|G|$ = 466 = 2*233
      - Test g = 466, $g^2 = 1$ (mod 467) = nicht erzeugendes
      - Test g = 465, $g^2 \neq 1$ $g^{233} = 1$ = nicht erzeugendes
      - Test g = 464, $g^2 \neq 1$ $g^{233} \neq 1$ = nicht erzeugendes
  - Für jede Primzahl p gilt: $\mathbb{Z}_p^*$ ist zyklisch.
- Ring: Menge R mit den Operationen "+" und "*" sodass:
  - R bezüglich Addition eine abelsche Gruppe bildet, und
  - Die Multiplikation assoziativ ist, d.h. $(a * b) * c = a * (b * c)$
  - Das Distributivgesetz ist erfüllt.
- Körper: Menge K mit den Operationen "+" und "*" sodass:
  - K bezüglich Addition eine abelsche Gruppe bildet, und
  - $K \backslash \{0\}$ bezüglich Multiplikatione eine abelsche Gruppe bildet
  - Das Distributivgesetz ist erfüllt.

1.2 RSA-Kryptosystem

Öffentlicher Schlüssel: (e, N)
Privater Schlüssel: d
Eigenschaften:

- N ist das Produkt zweier Primzahlen.
- $e \cdot d \equiv 1 \pmod{\varphi(N)}$.

Verschlüsselung einer Nachricht m**:** $c := m^e \mod N$
Entschlüsselung eines Chiffrates: $m := c^d \mod N$

2 Chinesischer Restsatz

Für paarweise teilerfremde Zahlen m_1, m_2, ... m_n hat jedes Gleichungssystem der Form:

$$x = a_1 (\text{mod } m_1)$$
$$x = a_2 (\text{mod } m_2)$$
$$...$$
$$x = a_n (\text{mod } m_n)$$

genau eine Lösung $m := m_1 \cdot m_2 \cdot ... m_n$

2.1 Spezialfall n = 2

Für teilerfremde Zahlen m_1, m_2 hat jedes Gleichungssystem der Form:

$$x = a_1 (\text{mod } m_1)$$
$$x = a_2 (\text{mod } m_2)$$

genau eine Lösung $m := m_1 \cdot m_2$

2.2 Nutzen für effizientes Modulo

Gesucht: $26 \cdot 37 \,(\text{mod } 105)$

- Repräsentation modulo 3,5,7: $26 \rightarrow (2,1,5)$, $37 \rightarrow (1,2,2)$
- elementweise Mulitplikation: $(2,1,5) \cdot (1,2,2) = (2,2,3)$
- Repräsentation modulo 105: $(2,2,3) \rightarrow 17$
- direkte Berechnung: $26 \cdot 37 = 17 \,(\text{mod } 105)$

Abbildung 1: Chinesischer Restsatz – Nutzen

2.3 Hilfssatz

Für jedes Element u einer Gruppe $\mathbb{Z}_m^*$ lässt sich $u^{-1} (\text{mod } m)$ effizient bestimmen.
Begründung: mit euklid. alg. lassen sich effizient x,y finden mit: $xu + ym = 1$ => $xu = 1 (\text{mod } m)$ => $x = u^{-1} (\text{mod } m)$

2.4 Diese Gleichungssysteme lösen

$$x = 2 (\text{mod } 3)$$
$$x = 5 (\text{mod } 7)$$

$$m := m_1 * m_2 = 3 * 7 = 21$$
$$M_1 := \frac{m_1}{m} = 7$$
$$M_2 := \frac{m_2}{m} = 3$$
$$u_1 := M_1^{-1} (\text{mod } m_1) = 7^{-1} (\text{mod } 3) = 1^{-1} (\text{mod } 3) = 1$$
$$u_2 := M_2^{-1} (\text{mod } m_2) = 3^{-1} (\text{mod } 7) = 5$$
$$x := \sum_{i=1}^{n} a_i \cdot (u_i M_i) (\text{mod } m)$$
$$= 2 * 1 * 7 + 5 * 5 * 3 (\text{mod } 21)$$
$$= 14 + 75 (\text{mod } 21) = 89 (\text{mod } 21) = 5 (\text{mod } 21)$$

2.5 Attacken auf RSA

2.5.1 Low Exponent Attacke

Wenn public exponent e klein und **gleiche Nachricht mehrmals an verschiedene Empfänger** gesendet kann Gleichungssystem aufgestellt werden dass mit chinesischem Restsatz auf originale Message aufgelöst werden kann

Bsp: $e = 3$, 3 recipients with same message

$$m^3 = c_1 (\text{mod } n_1)$$
$$m^3 = c_2 (\text{mod } n_2)$$
$$m^3 = c_3 (\text{mod } n_3)$$
$$x = m^3 (\text{mod } n_1 * n_2 * n_3) \Rightarrow \text{chinesischer Restsatz, 3. Wurzel}$$

2.5.2 Common Modulus Attacke

Falls $r|s$, dann gilt: $a = 1 (\text{mod } s) \Rightarrow a = 1 (\text{mod } r)$
Zwei Personen verwenden den gleichen Modulus

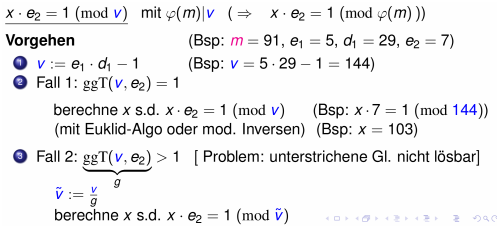


Abbildung 2: Common Modulus Attack

2.6 Primzahltest

Ziel: Finden von grossen Primzahlen

2.6.1 Begriffe:

- $\pi(x)$: # Primzahlen die $\leq x$ sind
- $\pi(x) \approx \frac{x}{ln(x)}$
- $\pi(\sqrt{n}) \approx \frac{\sqrt{n}}{ln(\sqrt{n})}$

2.6.2 Probedivision

Die Wahrscheinlichkeit, dass eine zufällig gewählte Zahl durch eine der Zahlen $p_1, p_2, ..., p_r$ teilbar ist, beträgt: $1 - \frac{\varphi(P)}{P} = 1 - (\frac{\varphi(1)}{p_1}) * 1 - (\frac{\varphi(1)}{p_2}) * ...1 - (\frac{\varphi(1)}{p_r})$

Tests mit einer Zufallskomponente: Grundidee: Teste auf eine Eigenschaft X (die effizient prüfbar ist) => es kommen noch paar falsche Zahlen mit aber beliebig kleines Risiko dabei

3 Fermat Test (Pseudoprimzhal Test)

$$a^{n-1} = 1 (\text{mod } n) \quad (1)$$
$$ggT(a, n) = 1 \quad (2)$$

Abbildung 3: Eine nicht-prime Zahl n heisst Pseudoprimzahl zur Basis a

Falls Primzahltest erfolgreich aber Zahl nicht Prim => called: Fermat Lügner (FL)

$$|FL(n)| =\leq \frac{\mathbb{Z}*_n}{2} \quad (1)$$
$$FL(n) = \mathbb{Z}*_n \quad (2)$$

Abbildung 4: Da FL eine Untergruppe von $\mathbb{Z}*_n$ => 2 Möglichkeiten

3.1 Für Fall 1:

Neue Zufallszahl a erzeugen => mit jeder Wiederholung t ist W'keit für false positive $\frac{1}{2}^t$

3.2 Für Fall 2: Carmichael-Zahlen

Eine zusammengesetzte natürliche Zahl n heisst Carmichael Zahl, wenn für alle $a \in \mathbb{Z}*_n$ gilt: $a^{n-1} = 1 \pmod{n}$.

3.2.1 Satz Carmichael Zahl:

Für jede zusammengesetzte Zahl $n \geq 3$ gilt: n ist Carmichael Zahl:

- n ist quadratfrei (=> n ist ungerade)
- für jeden Primteiler p von n gilt: $(p-1)|(n-1)$ (& n besitzt mindestens 3 Primfaktoren)

Für sichere Primzahl: Zahl auswählen und soll bedingung 2 nicht erfüllen für Carmichael zahl:
e.g.: 101 ist prim aber 100 = 2*50 < 3 primteiler => 101 nicht sicher prim

4 Public Key Verschlüsselung

- **Public Key Krypto-System** besteht aus 3 (effizienten) Algorithmen
  1. Schlüsselgenerator: erzeugt Schlüsselpaar (e, d)
  2. Verschlüsselungs-Algorithmus
  3. Entschlüsselungs-Algorithmus
- Anforderungen an Krypto System
  - **Entzifferbarkeit**: für jedes schlüsselpaar muss für jedes c mit e verschlüsselt mit d klartext hergestellt werden können
  - **Sicherheit**: ohne d zu kennen ist es mit vertretbarem aufwand unmöglich m zu erhalten
  - **Einweg-Funktion**: d kann nicht aus e oder beliebiger kombi von m u c berechnent werden
- Meist verwendet fpr Schlüsselaustausch (da eher langsam => nachher symmetrische Verfahren verwendet)
- Basieren oft darauf, dass diskreter Lograithmus schwierig zu berechnen ist

Einwegfunktion	**Umkehrfunktion**	**Verfahren**
Multiplikation	Faktorisierung	RSA
Exponentiation in $\mathbb{Z}_p^*$	diskreter Logarithmus in $\mathbb{Z}_p^*$	Diffie Hellman, El Gamal
Exponentiation in elliptischen Kurven	diskreter Logarithmus in elliptischen Kurven	Elliptic Curve Krypto
Punkt-Erzeugung in Gittern	Punkt-Rekonstruktion in Gittern	Gitter-basierte Verschlüsselung in Post Quantum Krypto

Tabelle 1: Einwegfunktionen und ihre Anwendungen

4.1 El Gamal Verschlüsselung

```
public(p=31, g=3, A=17), secret(a=7, b=12), m=9

check: A=g^a mod p = 17

c1 = g^b mod p = 3^12 mod 31 = 8
c2 = m*A^b mod p = 9*17^12 mod 31 = 18

C = (c1, c2) = (8,18)

s = c1^a mod p = 8^7 mod 31 = 2
s^(-1)= 2^(-1) mod 31 = 16

m = c2*s^(-1) mod p = 18*16 mod 31 = 9
```

Abbildung 5: El Gamal verschlüsselung

4.2 Diskreter Logarithmus in $\mathbb{Z}_\kappa^*$

Für gegebene $a, z \in \mathbb{Z}_\kappa^*$: $\log_a(z)$ löst $a^x = z(\mod n)$

- Ist die Basis ein **erzeugendes** Element, so existieren alle zugehörigen diskreten Logarithmen

4.3 Diffie-Hellman Schlüsselaustausch

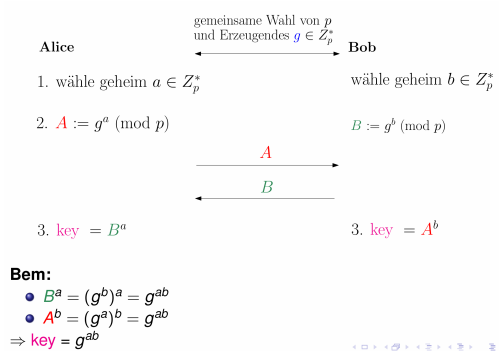


Abbildung 6: Diffie-Hellman Schlüsselaustausch

5 Faktorisierung

Algorithmisch schweres Problem (nur für Zahlen von bestimmter Form gibts Methoden)

5.1 (p-1)-Methode

Ein Primfaktor p von n hat die Eigenschaft: $p-1$ hat nur kleine Primfaktor-Potenzen: Sei $p-1 = p_1^{k_1} * ... * p_r^{k_r}$ "klein" bedeutet: $p_i^{k_i} \leq B$ Schranke (= powersmooth)

Für jedes Vielfache k von p-1 gilt:

$$k = s * (p-1) \tag{1}$$

$$a^k = a^{s(p-1)} = (a^{p-1})^s = 1^s = 1 \pmod p \tag{2}$$

$$a^k - 1 = 0 \pmod p \Rightarrow \text{p ist Teiler von } a^k - 1 \tag{3}$$

$$\gcd(a^k - 1, n) \geq p \tag{4}$$

Falls Primfaktor-Pontenzen von p-1 alle $\leq B$ sind:

$$k := \prod_{\substack{q \text{ prim} \\ q^e \leq B}} q^e \tag{1}$$

$$\text{k ist Vielfaches von p-1} \tag{2}$$

$$\gcd(a^k - 1, n) \geq p \tag{3}$$

Test: (falls erfolgreich: liefert Faktor von n)

$$1 < \gcd(a^k - 1, n) < n$$

5.1.1 Algorithm

```python
# iterate till a prime factor is obtained
while(True):

    # recomputing a as required
    a = (a**i) % n

    # finding gcd of a-1 and n
    # using math function
    d = math.gcd((a-1), n)

    # check if factor obtained
    if (d > 1):

        #return the factor
        return d

        break

    # else increase exponent by one
    # for next round
    i += 1
```

Abbildung 7: Pollard's p-1 Algorithm

5.2 Pollard ρ-Methode

geeignet für Zahlen, die einen kleinen Primfaktor p enthalten

Suche Zahlen x, y sodass:

$$x \neq y \text{ (mod n)} \quad (1)$$
$$x = y \text{ modulo einen Primfaktor p von n} \quad (2)$$

Test:

$$p \text{ teilt } x - y => \gcd(x - y, n) \text{ liefert Faktor von n} \quad (1)$$

5.2.1 Algorithm

1. Start with random x and c. Take y equal to x and f(x) = x^2 + c.
2. While a divisor isn't obtained
   1. Update x to f(x) (modulo n) [Tortoise Move]
   2. Update y to f(f(y)) (modulo n) [Hare Move]
   3. Calculate GCD of |x-y| and n
   4. If GCD is not unity
      1. If GCD is n, repeat from step 2 with another set of x, y and c
      2. Else GCD is our answer

Abbildung 8: Pollard ρ-Methode

6 Quadratisches Sieb

6.1 Begriffe

- F: Faktorbasis: enthält Primzahlen bis zu einer gewissen Schranke B und -1
- t heisst B-glatt (b-smooth) mod n: wenn t (mod n) nur aus Primfaktoren von F besteht.
- M: besteht aus allen Zahlen aus einer Grundmenge, deren Quadrate B-glatt mod n sind.

Grundidee:

$$x^2 = y^2 \text{ (mod n)} \quad (2)$$
$$x \neq y, x \neq -y \text{ (mod n)} \quad (3)$$
$$=> \gcd(x - y, n) \text{ : Faktor von n} \quad (4)$$

Input: Zahl n, Faktorbasis F
1. Fixiere eine Menge S von "kleinen" Zahlen
2. $m := \lfloor \sqrt{n} \rfloor$
3. **for** (jedes $x \in S$)
   3.1 Bestimme $q(x) := (m + x)^2 - n$
   3.2 Prüfe, ob $q(x)$ aus Primfaktoren in F zusammengesetzt ist
   Falls ja → Zahl mit gewünschter Eigenschaft gefunden
end

Abbildung 10: Naive Step 2 (not suitable as based on factorisation)

1. Bestimme eine Menge F von **kleinen** Primzahlen.
2. Finde Werte b_i, so dass $b_i^2 \pmod n$ nur aus Primfaktoren aus F besteht.
   $M :=$ Menge dieser b_i.
3. Finde $b_1, \ldots, b_r \in M$,
   gerade Zahlen $\alpha_0, \alpha_1, \ldots, \alpha_r \in \mathbb{N}$, und
   Primfaktoren $p_1, \ldots, p_k \in F$ so dass
   $b_1^2 \cdot b_2^2 \cdots b_r^2 = (-1)^{\alpha_0} \cdot (p_1)^{\alpha_1} \cdots (p_k)^{\alpha_k}$

setze $x := b_1 \cdots b_r$, $y := (-1)^{\alpha_0/2} \cdot (p_1)^{\alpha_1/2} \cdots (p_k)^{\alpha_k/2}$

Abbildung 9: Quadratisches Sieb Vorgehens-Skizze

6.2 Step 2

Goal: Find q(x)

- q(x) ist ein Quadrat modulo n (Wurzel: m +x)
- q(x) ist nicht "zu gross" (< n) => Faktorzerlegung wird nicht zu lang
- q(x) ist nicht "zu klein" => Faktorzerlegung wird nicht zu kurz

Verbesserung 3.2

1. Erstelle erste zwei Zeilen der vorherigen Tabelle.
2. Für jeden Faktor p F:
   - Finde einen Eintrag, der durch p teilbar ist.
   - Gehe mit Schritten der Länge p nach links und nach rechts.
   - Teile die gefunden Einträge so oft wie möglich durch p

Quadratisches Sieb
Input: Zahl n, Faktorbasis F
Output: B-glatte Zahlen modulo n
1. Fixiere eine Menge S von "kleinen" Zahlen
2. $m := \lfloor \sqrt{n} \rfloor$
3. **for** (jedes $x \in S$): berechne $q(x)$ **end**
4. multipliziere alle negativen Zahlen mit -1 // Sieb mit -1
5. **for** (jedes $p \in F$) // Sieb mit p
   5.1 Löse Gleichung $q(x) = 0 \pmod p$ $\rightarrow$ ergibt x_1 und ev. auch x_2
   5.2 Markiere in S die Elemente
   $\ldots, x_1 - 2p, x_1 - p, x_1, x_1 + p, x_1 + 2p, \ldots$ und
   $\ldots, x_2 - 2p, x_2 - p, x_2, x_2 + p, x_2 + 2p, \ldots$
   5.3 **for** (markierte x): teile $q(x)$ so oft wie möglich durch p **end**
   end
6. Gib diejenigen $q(x)$ aus, bei denen die fortlaufenden Divisionen zum Resultat 1 führen.

Abbildung 11: Algorithm Step 2

6.3 Step 3

lineares Gleichungssystem => unbekannte bestimmen u Quadrate auszuwählen die Perfect Square erzeugen (= alle Primfaktoren haben gerade Exponenten

Veranschaulichung anhand eines Beispiels
- $n = 117$
- $F = \{-1, 2, 3, 5, 7\}$ (**Bem:** -1 gehört als 'Nicht-Primzahl' dazu.)
- $M := \{15, 25, 37, 47\}$.
- Beleg, dass Quadrate aus M nur aus Primfaktoren aus F bestehen:
  - $b_1 = 15$: $15^2 = (-1) \cdot 3^2 \pmod n$
  - $b_2 = 25$: $25^2 = 2^3 \cdot 5 \pmod n$
  - $b_3 = 37$: $37^2 = (-1) \cdot 5 \cdot 7 \pmod n$
  - $b_4 = 47$: $47^2 = (-1) \cdot 2 \cdot 7 \pmod n$

Gesucht: $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ so dass bei $(15^2)^{\lambda_1} \cdot (25^2)^{\lambda_2} \cdot (37^2)^{\lambda_3} \cdot (47^2)^{\lambda_4}$ alle Elemente von F gerade Exponenten haben. Respektive:
(1) $\lambda_1 + \lambda_3 + \lambda_4 = 0 \pmod 2$ [(-1) hat geraden Exponent]
(2) $\lambda_2 + \lambda_4 = 0 \pmod 2$ [2 hat geraden Exponent]
(3) $\lambda_2 + \lambda_3 = 0 \pmod 2$ [5 hat geraden Exponent]
(4) $\lambda_3 + \lambda_4 = 0 \pmod 2$ [7 hat geraden Exponent]

Abbildung 12: Example Step 3

- Gleichungssystem Lösen mod 2 => nicht triviale lösung mit mind. 2 λ, λ durch b ersetzen
- rechnung mit b aufstellen = x , rechnung mit faktoren von b = y
- gcd(x-y, 1) = teiler

7 Wurzeln (Algorithm von Tonelli)

Für Quadratisches Sieb q(x) => muss Wurzel gezogen werden: Suche Lösungen von: $x^2 = a \pmod p$ Wenn p = Primzahl;

$$(\frac{p-1}{2})^2 = (p - \frac{p-1}{2})^2$$

Daraus folgen: Sätze

- Die Hälfte der Elemente von $\mathbb{Z}_p^*$ hat eine Wurzel, die andere Hälfte nicht.
- Jedes Element aus $\mathbb{Z}_p^*$ hat entweder 0 oder 2 Wurzeln.
- v. Euler: Für jedes $a \in \mathbb{Z}_p^*$ gilt:

$$a^{\frac{p-1}{2}} \pmod p = \begin{cases} 1, & \text{falls } a \text{ eine Wurzel hat} \\ -1, & \text{falls } a \text{ keine Wurzel hat} \end{cases}$$

Notation

- a hat Wurzel in $\mathbb{Z}_p^*$: a ist QRp (quadratischer Rest modulo p)
- a hat keine Wurzel in $\mathbb{Z}_p^*$: a ist QRNp (quadratischer Nicht-Rest modulo p)

7.1 Algorithm von Tonelli

Test v Euler (ob hat Wurzel) siehe oben, dann p mod 4 berechnen und fall 1 / 2 benützen

7.1.1 Fall 1: $p = 3$ (mod 4)

Benutzte Beobachtung: Findet man ein ungerades u mit $a^u = 1 \pmod p$ so gilt:

- $a^{u+1} = a$
- $a^{\frac{u+1}{2}}$ ist Wurzel von a

Folgerung mit Euler: $a^{\frac{p-1}{2}} = 1, p = 3 \pmod 4 => \frac{p-1}{2}$ ungerade $=> u = \frac{p-1}{2}$ gibt $a^{\frac{u+1}{2}} = a^{\frac{p+1}{4}}$ ist Wurzel
Ist $p = 3 \pmod 4$ so gilt für alle $a \in QRp$: $a^{\frac{p+1}{4}}$ ist eine Wurzel von a

7.1.2 Fall 2: $p = 1$ (mod 4)

Benutzte Beobachtung: Findet man ein $h \in \mathbb{Z}_p^*$, ein ungerades und ein gerades g mit $a^u h^g = 1 \pmod p$ so gilt:

- $a^{u+1} h^g = a$
- $a^{\frac{u+1}{2}} h^{\frac{g}{2}}$ ist Wurzel von a

Grundidee

- h := irgendein Element ohne Wurzel ('quadratischer Nichtrest')
- gemäss dem Euler-Kriterium ist $a^{\frac{p-1}{2}} h^{p-1} = 1$
- Halbiere die Exponenten der obigen Gleichung so lange, bis der Exponent von a ungerade wird. (Sollte das Produkt 1 werden, erhöhe Exponenten von h um $\frac{p-1}{2}$ Dies entspricht einer Multiplikation mit 1.)

Fall 2: $p = 1 \pmod 4$ (Forts.)
Beispiele:

- Gleichung: $x^2 = 2 \pmod{73}$.
  Algorithmus von Tonelli:
  - Setze $a := 2$.
  - Wähle (zum Beispiel) $h = 10$.
    (10 ist quadratischer Nichtrest, da $10^{36} = -1 \pmod{73}$)
  - Einsetzen ergibt: $a^{\frac{p-1}{2}} \cdot h^{p-1} = 2^{36} \cdot 10^{72} = 1 \pmod{73}$.
  - Halbierung der Exponenten: $2^{18} \cdot 10^{36} = -1 \pmod{73}$.
  - Korrektur: $\frac{p-1}{2} = 36$. Addition im Exponenten ergibt:
    $2^{18} \cdot 10^{36+36} = 2^{18} \cdot 10^{72} = 1 \pmod{73}$
  - Halbierung der Exponenten: $2^9 \cdot 10^{36} = -1 \pmod{73}$
  - Korrektur: $2^9 \cdot 10^{36+36} = 2^9 \cdot 10^{72} = 1 \pmod{73}$
  - Einsetzen von $u := 9$ und $g := 72$ in der Beobachtung auf einer vorherigen Folie ergibt:
    $a^{\frac{u+1}{2}} \cdot h^{\frac{g}{2}} = 2^{\frac{9+1}{2}} \cdot 10^{\frac{72}{2}} = 2^5 \cdot 10^{36} = 41 \pmod{73}$ ist eine Lösung.

Abbildung 13: Algorithmus von Tonelli Fall 2 Beispiel

8 Algorithmen Logarithmus

Etablierte Annahme: Dieses Problem ist schwierig zu lösen

8.1 Enumeration

Die Zahlen $x = 1, 2, 3, .., n$ der Reihe nach durchgehen und prüfen, ob die Gleichung $a^x = g$ erfüllt ist. Exponentielle Laufzeit

8.2 Baby Step - Giant Step Algorithm

schnellere Laufzeit, grösserer Speicher-Bedarf

8.2.1 Vorüberlegungen

Für jedes m gilt: Jede ganze Zahl lässt sich darstellen als $x = q * m + r$. (basierend auf ganzahldivison + rest)

Vorüberlegungen II:

- Setze $n :=$ Ordnung der betrachteten (zyklischen) Gruppe G.
- Setze $m := \lceil \sqrt{n} \rceil$
- Ansatz für x: $x = q \cdot m + r$ für entsprechende Werte q,r.
  (Hinweis: q, r sind spezifiert durch die Ganzzahl-Division $x : m$.)
- Einsetzen des Ansatzes gibt: $g^x = g^{qm+r} = g^{qm} \cdot g^r \overset{!}{=} a$
- Diese Gleichung lässt sich umformen zu $g^r = a \cdot g^{-qm} = a \cdot (g^{-m})^q$
- Verbleibende Gleichung ist somit: $\underbrace{g^r}_{\text{Baby Steps}} = \underbrace{a \cdot (g^{-m})^q}_{\text{Giant Steps}}$

Grundidee des Algorithmus

- Berechnung von g^r für alle r mit $0 \le r \le m - 1$ (Baby Steps)
- Berechnung von $a \cdot (g^{-m})^q$ für alle q mit $0 \le q \le \frac{n}{m}$ (Giant Steps)
- Suche nach einem Paar (r, q), bei dem obige 2 Werte gleich sind.

Bem: m wurde so gewählt, dass #(Baby Steps) $\approx$ #(Giant Steps) ist.

Abbildung 14: Baby Step Giant Step Algorithm

8.3 Index Calculus

1. Bestimme eine Menge F von **kleinen** Primzahlen
2. $M :=$ Menge von gewissen b_i, deren Primfaktoren alle in F liegen.
3. Für jedes $b_i \in M$: Bestimme jeweiligen Logarithmus $x_i := \log_g(b_i)$

   Dies ergibt:

b_1,	b_2,	$\ldots$,	b_s
$\shortparallel$	$\shortparallel$		$\shortparallel$
g^{x_1}	g^{x_2}		g^{x_s}

4. Finde y, so dass ag^y sich als Produkt aus M darstellen lässt:

$$\begin{aligned} ag^y &= b_1^{e_1} \cdot b_2^{e_2} \cdot \ldots \cdot b_s^{e_s} \\ &= (g^{x_1})^{e_1} \cdot (g^{x_2})^{e_2} \cdot \ldots \cdot (g^{x_s})^{e_s} \\ &= g^{x_1 e_1 + x_2 e_2 + \ldots + x_s e_s} \end{aligned}$$

5. Aus vorherigem Schritt folgt:
   - $a = g^{x_1 e_1 + x_2 e_2 + \ldots + x_s e_s - y}$
   - $x = x_1 e_1 + x_2 e_2 + \ldots + x_s e_s - y$

Abbildung 15: Index Calculus Grundidee

Erfordert komplexe techniken und ist nicht allgemein einsetzbar:

- spezielle modulare gleichungs-systeme, chinesischer restsatz, effiziente logarithmus berechnungen
- Der Algorithmus benötigt u.a. eine eindeutige Primfaktorzerlegung (formal ausgedrückt: eine bestimmte Ring-Struktur).
- Der Algorithmus funktioniert fast nur in der multiplikativen Gruppe von $\mathbb{Z}_n^*$.

8.4 Pollard ρ Methode

- Ermitteln von s und t mit der Eigenschaft $a^s = g^t$. (Einsetzen in die Ursprungs-Gleichung liefert $g^{sx} = g^t$.)
- Lösen der Gleichung $s^x = t \pmod G$

Falls $u = v \pmod n$ und d ein Teiler von v und n ist, dann gilt
- $d \mid u$, und
- $\frac{u}{d} = \frac{v}{d} \left(\mod \frac{n}{d}\right)$

Abbildung 16: Modulare Äquivalenzen

Betrachtete Gleichung: $ax = b \pmod n$
Notation: $d := \mathrm{ggT}(a, n)$.

- Fall 1: $d = 1$. Dann ist $x = a^{-1} \cdot b \pmod n$ [einzige Lösung]
- Fall 2: $d > 1$. Gemäss vorheriger Beobachtung:
  - Falls $d \nmid b$: Die Gleichung hat keine Lösung.
  - Falls $d \mid b$: $\frac{a}{d}x = \frac{b}{d} \left(\mod \frac{n}{d}\right)$.
    - Da $\frac{a}{d}$ und $\frac{n}{d}$ teilerfremd sind, entspricht diese Gleichung Fall 1.
    - Mit $z := \left(\frac{a}{d}\right)^{-1} \left(\mod \frac{n}{d}\right)$ ist die Lösung somit $x = z \cdot \frac{b}{d} \left(\mod \frac{n}{d}\right)$.
    - Alle Lösungen der ursprünglichen Gleichung: $z \cdot \frac{b}{d} + k \cdot \frac{n}{d} \pmod n$ $(*)$ mit $k \in \{0, 1, 2, \ldots, d-1\}$.

Abbildung 17: Modulare Lineare Gleichungen

Vorüberlegungen

8.4.1 Schritt 1

Hat x_i die Form $x_i = a^r g^s$, so ist $x_{i+1} = a^{\tilde{r}} g^{\tilde{s}}$ mit

$$\tilde{r} = \begin{cases} \tilde{r} = r + 1, & \tilde{s} = s & \text{falls } x_i \in G_1 \\ \tilde{r} = 2r, & \tilde{s} = 2s, & \text{falls } x_i \in G_2 \\ \tilde{r} = r, & \tilde{s} = s + 1, & \text{falls } x_i \in G_3 \end{cases}$$

Abbildung 18: Schritt 1 Pollard ρ

- Die Folge $x0, x1, x2, \ldots$ ist so konstruiert, dass sie sich ähnlich wie eine Folge von zufälligen Elementen verhält.
- Analog zum Pollard-ρ-Algorithmus für die Faktorisierung lässt sich zeigen, dass bei $k \geq 1.2\sqrt{|G|}$ Elementen mit Wahrscheinlichkeit > 0.5 ein Paar mit $x_i = x_j$ dabei ist.

8.4.2 Schritt 2

Bemerkungen

- Analog zum Pollard-Algorithmus für die Faktorisierung lässt sich zeigen, dass eine Kollision der Form xi = x2i in ungefähr gleich vielen Schritten wie irgendeine Kollision xi = xj gefunden wird.

- Schritt 1 liefert ein Paar $x_i = x_j$.
- Es gibt Zahlen $r, s, \tilde{r}, \tilde{s}$, so dass $x_i = a^r g^s$ und $x_j = a^{\tilde{r}} g^{\tilde{s}}$.
- Gleichsetzen ergibt $a^r g^s = a^{\tilde{r}} g^{\tilde{s}}$ resp. $a^{r-\tilde{r}} = g^{\tilde{s}-s}$.
- Also: $a^t = g^u$ mit $t := r - \tilde{r}$ und $u := \tilde{s} - s$.
- Einsetzen von $g^x = a$ in die obige Gleichung liefert $g^{tx} = g^u$.
- Dies ist äquivalent zur Gleichung $tx = u \pmod{(|G|)}$.
  Diese Gleichung kann via $(*)$ von Folie 3 bestimmt werden.
  (Auswählen derjeniger Lösung, die $g^x = a$ erfüllt.)

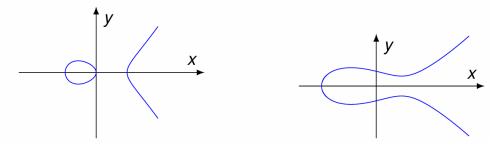
  Hinweis: Da g ein erzeugendes Element ist, hat die obige Gleichung auf alle Fälle eine Lösung.

Abbildung 19: Schritt 2 Pollard ρ

- Somit reicht es, die Tripel (xi ri si) jeweils nur solange zu speichern, bis der Index i die nächst-höhere Zweier-Potenz erreicht hat.
- Damit ist der Pollard-Algorithmus deutlich Speicher-effizienter als der Babystep-Giantstep Algorithmus.

9 Elliptische Kurven

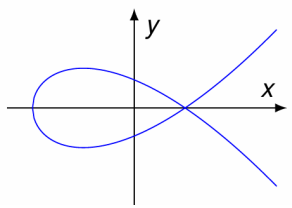
Für gegebene a, b mit $4a^3 + 27b^2 \neq 0$ beinhaltet die entsprechende elliptische Kurve alle Punkte (x, y) mit $x, y \in K$ und der Eigenschaft $y^2 = x^3 + ax + b$ sowie einen Zusatzpunkt O.
$E_{a,b}$ bezeichnet die Elemente der elliptischen Kurve der Form $y^2 = x^3 + ax + b$.



$$y^2 = x^3 + ax + b$$

Abbildung 20: Gleichung Elliptische Kurve (Küste mit Insel / Kleiderbügel)

Grenzfall: Küste und Insel Treffen sich: Ausschliessen in Definition oben

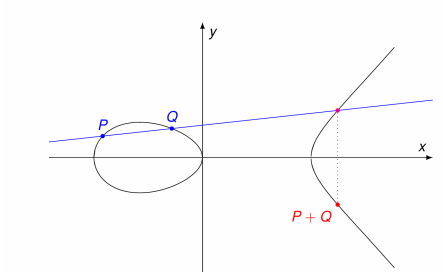


$$4a^3 + 27b^2 = 0$$

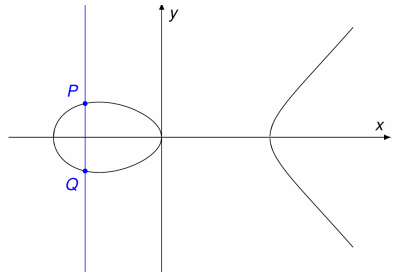
Abbildung 21: Form Ellipse Küste und Insel Grenzfall

9.1 Addition: Zusatzkonstrukt benötigt

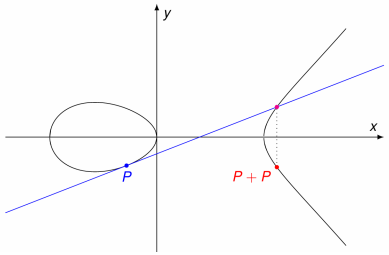
Addition benötigt Zusatzkonstrukt von 'unendlich fernen Punkt' O



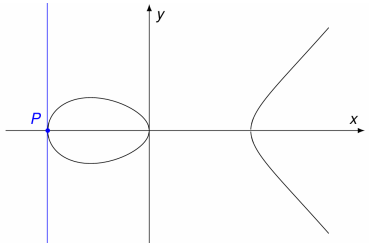
(a) Standard-Fall: P+Q



(b) Sonderfall 1: gleiche x-Koordinate $P + Q = O$



(c) Sonderfall 2: P = Q, $y \neq 0$



(d) Sonderfall 3: P = Q, $y = 0$ => $P + P = O$

Abbildung 22: Addition von Punkten auf elliptischen Kurven

9.1.1 Additionsoperationen Rechnung

Für Punkte $P = (x_1, y_1)$ und $Q = (x_2, y_2)$ auf der Kurve gilt:

a) Falls $x_1 \neq x_2$:

$$m := \frac{y_2 - y_1}{x_2 - x_1}, \quad x_3 := m^2 - x_1 - x_2, \quad y_3 := -m(x_3 - x_1) - y_1, \quad P+Q := (x_3, y_3)$$

b) Falls $x_1 = x_2$ und $y_1 = y_2 \neq 0$:

$$m := \frac{3x_1^2 + a}{2y_1}, \quad x_3 := m^2 - 2x_1, \quad y_3 := -m(x_3 - x_1) - y_1, \quad P+Q := (x_3, y_3)$$

c) Falls $x_1 = x_2$ und $y_1 = -y_2$ setzen wir

$$P + Q := O.$$

d) $P + O = O + P = P$.

Bem: $2P = P + P$.

9.2 Spezifische Eigenschaften von Elliptischen Kurven

- Verbindet man zwei Punkte, die beide auf der Kurve liegen, und versch. x-Koordinaten haben dann gibt es einen weiteren Schnittpunkt.
- Legt man die Tangente an einen Punkt auf der Kurve, dann gibt es einen weiteren Schnittpunkt.

9.2.1 Pari/GP Befehle

- **ellinit:** Aufruf: E = ellinit([a,b],p) im Bsp: E = ellinit([1,6],11)
- **elladd:** Aufruf: elladd(E,[x1 y1], [x2 y2]) z.B.: elladd(E,[2,7],[3,6])
- **ellpow:** Aufruf: ellpow(E,[x y],n) z.B.: ellpow(E, [2,4], 2)
- **ellmul:** Aufruf: ellmul(E,[x y],n) z.B.: ellmul(E, [2,4], 2)

- GF(mod) alle ellipt. kurve sind mod angegeben als GF (Galois Field)

	$\mathbb{Z}_p^*$	ell. Kurve
Potenz	g^x	$x \cdot P$
Ordnung	kleinstes n so dass $g^n = 1$	kleinstes n so dass $n \cdot P = 0$
Logarithmus	$g^x = a$	$x \cdot P = a$

Abbildung 23: Multiplikative Gruppe vs Additive Gruppe

9.3 Kryptosystem Miller-Koblitz

Idee: basierend auf Punkten auf Elliptischen Kurven (=> schwierig disk. log. zu lösen)

10 Kryptographie Elliptische Kurven

10.1 Schlüssel-Austausch für elliptische Kurven:

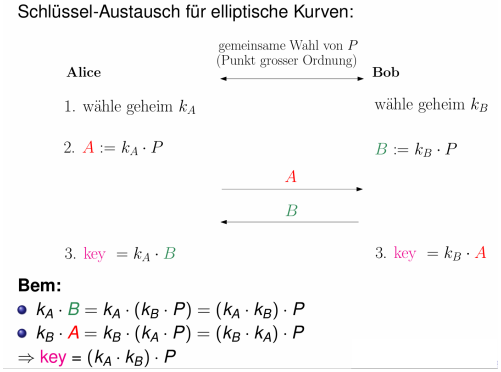


Abbildung 24: Diffie Hellman Schlüssel-Austausch

10.2 Verschlüsselung für elliptische Kurven:

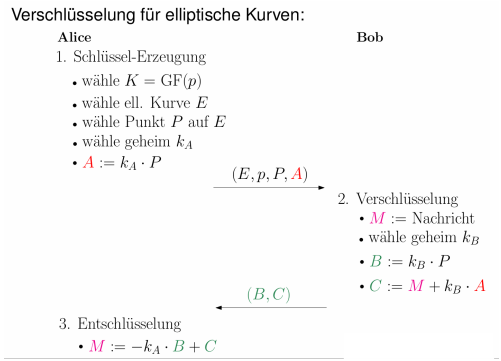


Abbildung 25: El Gamal Verschlüsselung

10.3 Nachricht => Punkt auf elliptischer Kurve

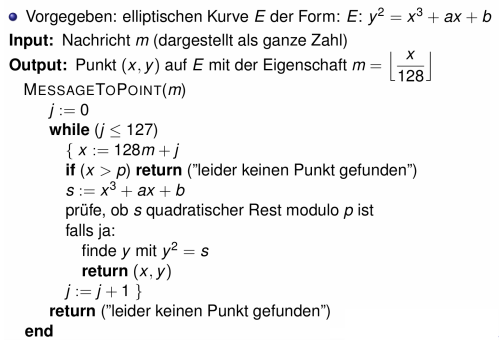


Abbildung 26: Message to Point on Curve

Zum entschlüsseln => $m := \lfloor \frac{x}{128} \rfloor$

11 Digitale Signaturen

use public key krypto to sign documents:

1. A signiert Dok m mit private key: d_A => Signiertes Dokument: Paar $(m, d_A(m))$
2. B verifiziert Signatur mit public key e_A => berechnet $e_A(d_A(m))$ => $= m$?

normalerweise wird statt gesamtes dokument nur hash wert von m signiert (mit öffentlich, kollisionsresistenter hashfunktion)

11.1 Digitale Signatur mit RSA

See above

11.2 Digital Signature Algorithm DSA (basierend auf El Gamal)

Im El Gamal Verfahren sind Verschlüsselung und Entschlüsselung nicht direkt vertauschbar. => leichte modifikationen

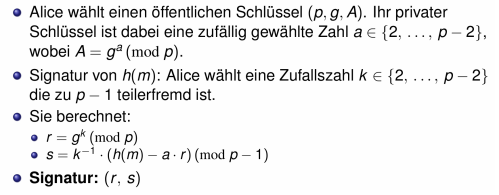


Abbildung 27: Signatur El Gamal

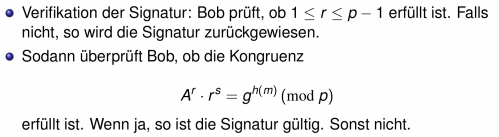


Abbildung 28: Überprüfung El Gamal

11.3 Elliptic Curve Digital Signature Algorithm ECDSA

ECDSA basiert auf der Arithmetik elliptischer Kurven über endlichen Körpern.

Schlüsselerzeugung: Es wird eine elliptische Kurve E über $GF(p)$ und ein Punkt P von Primordnung n gewählt. Der geheime Schlüssel ist eine Zufallszahl $d < n$; der öffentliche Schlüssel lautet $Q = dP$.

Signaturerstellung: Zur Signierung einer Nachricht m wird eine zufällige Zahl $k < n$ gewählt und der Punkt $kP = (x, y)$ berechnet. Die Signaturkomponenten ergeben sich aus $r = x \bmod n$ und $s = (h(m) + rd)\, k^{-1} \bmod n$, wobei $h(m)$ der Hashwert der Nachricht ist. Die Signatur ist das Paar (r, s).

Verifikation: Zur Prüfung einer Signatur (r, s) wird $u = h(m)s^{-1} \bmod n$ und $v = rs^{-1} \bmod n$ berechnet. Anschliessend bestimmt man $uP + vQ = (a, b)$. Die Signatur ist gültig, falls $a \equiv r \pmod n$ gilt.

Funktionsweise: Da $uP + vQ = kP$ gilt, besitzt der resultierende Punkt dieselbe x-Koordinate wie kP, also r. Dadurch ist die Verifikation korrekt.

Sicherheitsaspekte: Umgang mit d und k: Der geheime Schlüssel d darf nur der Unterzeichner kennen. Ebenso darf die Nonce k, die für jede Signatur zufällig gewählt wird, niemals veröffentlicht oder wiederverwendet werden. Andernfalls lässt sich d leicht berechnen:

$$s = (h(m) + rd)k^{-1} \implies d = (ks - h(m))r^{-1} \bmod n$$

Wenn dasselbe k für mehrere Signaturen verwendet wird, erkennt ein Angreifer dies an gleichen r-Werten. Aus zwei Signaturen (r, s_1) und (r, s_2) mit derselben Nonce lässt sich k bestimmen:

$$s_1 - s_2 = (h(m_1) - h(m_2))k^{-1} \implies k = (s_1 - s_2)^{-1}(h(m_1) - h(m_2)) \bmod n$$

Mit bekanntem k kann dann der geheime Schlüssel d berechnet werden. Daher muss jede Nonce eindeutig und geheim bleiben.

12 Post Quantum Crypto

12.1 Gitter (Lattice) Definition

besteht (in der Ebene) aus allen Punkten, die von ganzzahligen Linear-Kombinationen von zwei gegebenen Basis-Vektoren gebildet werden.

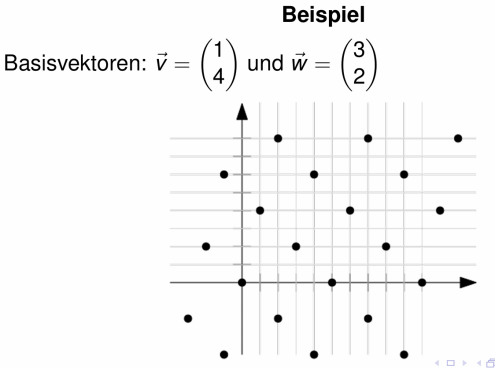


Abbildung 29: Gitter Beispiel

Dasselbe Gitter kann von unendlich vielen verschiedenen Paaren von Basisvektoren erzeugt werden.

12.2 Closest Vector Problem

Welcher Gitterpunkt ist am nächsten zu einem gegebenen Punkt P?

- Das Closest Vector Problem ist eher einfach zu lösen, wenn die Basen $\vec{v}$ und $\vec{w}$ nahezu orthogonal (rechtwinklig) zueinander sind.
- Das Closest Vector Problem ist schwierig zu lösen, wenn die Basen $\vec{v}$ und $\vec{w}$ nahezu kolinear (deckend) sind – auch für Quantencomputer.

12.3 Closest Vector Problem Kryptosystem

- Privater Schlüssel: "gute"Basis (fast orthogonale Vektoren)
- Öffentlicher Schlüssel: ßchlechte"Basis (fast kollineare Vektoren) (= Gleichungssystem)
- Mitteilung: Punkt P im Koordinatensystem plus leichte Modifikation

Beispiel

- privater Schlüssel: $\vec{v} = \begin{pmatrix}1\\4\end{pmatrix}$, $\vec{w} = \begin{pmatrix}3\\2\end{pmatrix}$,
- öffentlicher Schlüssel: $\vec{v_2} = \begin{pmatrix}8\\2\end{pmatrix}$ und $\vec{w_2} = \begin{pmatrix}11\\4\end{pmatrix}$
- Nachricht von Bob: $(2,1)$. Ergibt $P = 2\vec{v_2} + \vec{w_2} = \begin{pmatrix}27\\8\end{pmatrix}$
- Addition vom "Fehler" $\begin{pmatrix}1\\0\end{pmatrix}$ ergibt $\begin{pmatrix}28\\8\end{pmatrix}$. Gesendete Nachricht $\begin{pmatrix}28\\8\end{pmatrix}$

Abbildung 30: Beispiel Closest Vector Problem Krypto

Idee: (funktioniert nicht (Hintertür Angriffe möglich)

- Aufgabe für Entschlüssler/Angreifer: Closest Vector Problem lösen.
- Für Alice (mit "guter Basis"') einfach.
- Für Angreifer (mit ßchlechter Basis") schwierig.

12.4 Verbesserung Closest Vector Problem: Learning with Errors

Abwandlung des Closest Vector Problems: Übersicht:

- Öffentlicher Schlüssel: Basisvektoren => überbestimmtes Gleichungssystem mit ”kleinen Fehlern” (alles in einer modularen Restgruppenklasse gerechnet)
- Privater Schlüssel: Gute Basis => Lösung des Gleichungssystems
- Gesendete Nachricht: Punkt => Addition von mehreren Gleichungen plus Zusatzwert

- Öffentlicher Schlüssel: überbestimmtes Gleichungssystem – alle Gleichungen modulo 97, plus Fehlervektor

$$\begin{pmatrix}5&7&2\\4&8&3\\6&9&4\\18&15&17\\9&14&25\\3&6&1\\2&4&6\end{pmatrix}\begin{pmatrix}x\\y\\z\end{pmatrix} \approx \begin{pmatrix}25\\29\\36\\2\\15\\18\\28\end{pmatrix} + \begin{pmatrix}2\\3\\1\\3\\2\\3\\1\end{pmatrix}$$

- Zusätzlicher Fehlervektor mit **kleinen Werten**: $\vec{f} = \begin{pmatrix}2\\3\\1\\3\\2\\3\\1\end{pmatrix}$
- privater Schlüssel: $x = 1$, $y = 2$, $z = 3$

Abbildung 31: Learning with Errors Beispiel

Ohne Kenntnis von $\vec{f}$ ist es schwierig, den privaten Schlüssel herauszufinden.

- Öffentlicher Schlüssel:

$$\begin{pmatrix}5&7&2\\4&8&3\\6&9&4\\18&15&17\\9&14&25\\3&6&1\\2&4&6\end{pmatrix}\begin{pmatrix}x\\y\\z\end{pmatrix} \approx \begin{pmatrix}27\\32\\37\\5\\17\\21\\29\end{pmatrix}$$

- Verschlüsselung von Bob:
  - Bob wählt zufällig einige Gleichungen aus und addiert sie (mod 97)

$$(30\quad 36\quad 23)\begin{pmatrix}x\\y\\z\end{pmatrix} \approx (85)$$

  - Codierung von Bit 0: Bob schickt obige Gleichung.
  - Codierung von Bit 1: Bob addiert $\lfloor \frac{97}{2} \rfloor = 48$ zur rechten Seite:

$$(30\quad 36\quad 23)\begin{pmatrix}x\\y\\z\end{pmatrix} \approx (85 + 48 = 36)$$

Abbildung 32: Verschlüsselung Learning with Errors

Decodierung von Alice

- Alice setzt privaten Schlüssel ein: $(30\quad 36\quad 23)\begin{pmatrix}1\\2\\3\end{pmatrix} = (74)$
- Alice subtrahiert diese Zahl vom Wert auf rechter Seite von Bob.
  Falls das Resultat näher bei 0 ist, decodiert sie 0.
  Falls das Resultat näher bei 48 ist, decodiert sie 1.

Abbildung 33: Entschlüsselung Learning with Errors

Conclusion Theoretisch (gemäss bisherigen Erkenntnissen) funktioniert! (Es ist noch nichts bekannt, was diese Methode entkräften würde.) Allerdings: Es

gibt noch diverse Aspekte, welche das System in dieser Form (für praktische Anwendungen wohl viel zu) umständlich machen, u.a.

- sehr grosser öffentlicher Schlüssel
- aufwändige bitweise Codierung: Für ein einzelnes Bit muss ein ganzes Gleichungssystem versendet werden!
- Hinweis: Es gibt weitere Twists, mit denen sich diese Probleme entschärfen/beheben lassen.