

# Kryptographie

## Anwendungen wie

- 1 Pseudo-Random-Generatoren
- 2 Public-Key-Kryptographie
- 3 Digitale Unterschriften
- 4 Kryptographische Hashfunktionen

## Sicherheit gegen

- 1 deterministische Attacken
- 2 Attacken mit Quanten-Algorithmen

**Aber:** Der Aspekt einer sicheren und effizienten *Implementierung* muss aus zeitlichen Gründen vernachlässigt werden.

Fundamentale kryptographische Bausteine wie

- (a) **One-Way-Funktionen**: einfach auswertbare Funktionen, deren Invertierung aber schwierig ist.
- ▶ Methoden der Zahlentheorie bzw.
  - ▶ der Gitterkryptographie.
    - ★ Worst-Case-Komplexität
    - ★ Average-Case-Komplexität.
- (b) **Trapdoor-Funktionen**: One-Way-Funktionen, die mit Hilfe eines geheimen Schlüssels effizient invertierbar sind.

# One-Way-Funktionen

# One-Way-Funktionen

Eine Funktion  $f : \mathbb{N} \rightarrow [0, 1]$  heißt **vernachlässigbar**, wenn es zu **jedem**  $k \in \mathbb{N}$  eine Zahl  $n_k \in \mathbb{N}$  gibt, so dass für alle  $n \geq n_k$

$$f(n) \leq \frac{1}{n^k}$$

Schreibe  $x \in_R D$ , wenn  $x$  gemäß der Gleichverteilung aus  $D$  gezogen wird.

# One-Way-Funktionen

Eine Funktion  $f : \mathbb{N} \rightarrow [0, 1]$  heißt **vernachlässigbar**, wenn es zu **jedem**  $k \in \mathbb{N}$  eine Zahl  $n_k \in \mathbb{N}$  gibt, so dass für alle  $n \geq n_k$

$$f(n) \leq \frac{1}{n^k}$$

Schreibe  $x \in_R D$ , wenn  $x$  gemäß der Gleichverteilung aus  $D$  gezogen wird.

(a) Eine Funktion  $F : \{0, 1\}^* \rightarrow \{0, 1\}^*$  ist eine **One-Way-Funktion**, wenn  $F$  deterministisch effizient-berechenbar ist und wenn für jede Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen polynomieller Größe die W-keit

$$\Pr_{x \in_R \{0,1\}^n} [T_n \text{ berechnet auf Eingabe } F(x) \text{ ein } z \text{ mit } F(z) = F(x)]$$

einer erfolgreichen Invertierung **vernachlässigbar** ist.

# One-Way-Funktionen

Eine Funktion  $f : \mathbb{N} \rightarrow [0, 1]$  heißt **vernachlässigbar**, wenn es zu **jedem**  $k \in \mathbb{N}$  eine Zahl  $n_k \in \mathbb{N}$  gibt, so dass für alle  $n \geq n_k$

$$f(n) \leq \frac{1}{n^k}$$

Schreibe  $x \in_R D$ , wenn  $x$  gemäß der Gleichverteilung aus  $D$  gezogen wird.

- (a) Eine Funktion  $F : \{0, 1\}^* \rightarrow \{0, 1\}^*$  ist eine **One-Way-Funktion**, wenn  $F$  deterministisch effizient-berechenbar ist und wenn für jede Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen polynomieller Größe die W-keit  $\Pr_{x \in_R \{0,1\}^n} [T_n \text{ berechnet auf Eingabe } F(x) \text{ ein } z \text{ mit } F(z) = F(x)]$  einer erfolgreichen Invertierung **vernachlässigbar** ist.
- (b)  $F$  ist eine **One-Way-Permutation**, wenn  $F$  eine One-Way-Funktion ist und  $F$ , für jedes  $n \in \mathbb{N}$ , eine Permutation von  $\{0, 1\}^n$  ist.

# Was tun Tests $T$ ?

Ein effizienter Test  $T = (T_n : n \in \mathbb{N})$  ist eine (mgl. nicht-uniforme) Familie von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$ .

- 1 Interpretiere einen Test als eine **nicht-uniforme** Folge von *effizienten deterministischen* Algorithmen, also einen Algorithmus für jede Eingabelänge.
- 2 Der Test kann insbesondere die vermeintliche One-Way-Funktion  $F$  simulieren, denn  $F$  ist effizient berechenbar.
- 3 Nach all diesen Vorbereitungen kommt der „Test für den Test“:

*$T$  muss die Funktion  $F$  auf dem Funktionswert  $F(x)$  invertieren:  
Das unbekannte Argument  $x$  wird gleichverteilt ausgewürfelt.*



# Familien von One-Way-Funktionen

Gegeben sind

- eine Teilmenge  $I \subseteq \{0, 1\}^*$  von Indizes sowie
- endliche Mengen  $D_i$  und Funktionen  $f_i : D_i \rightarrow \{0, 1\}^*$  für jeden „Index“  $i \in I$ .

$\mathcal{F} = ((i, f_i) : i \in I)$  ist eine Familie von One-Way-Funktionen, falls:

1. Die Indexmenge  $I$  lässt sich effizient „sampeln“: Es gibt einen effizienten randomisierten Algorithmus **KeyGen** mit  $\text{KeyGen}(1^n) \in I \cap \{0, 1\}^n$  für jedes  $n \in \mathbb{N}$ .
2. Für jeden Index  $i \in I$  ist die Funktion  $f_i$  effizient auswertbar. D.h. es gibt einen effizienten randomisierten Algorithmus **Enc** mit  $\text{Enc}(i, x) = f_i(x)$  für jedes  $x \in D_i$ .
3. Sei  $T = (T_n : n \in \mathbb{N})$  eine Familie von Schaltkreisen polynomieller Größe. Dann ist

$\text{pr}_{i \in I \cap \{0, 1\}^n, x \in {}_R D_i} [ T_n(i, f_i(x)) = x' \text{ und } f_i(x) = f_i(x') ]$  vernachlässigbar.

(Index  $i$  wird zufällig durch **KeyGen** und  $x$  gemäß der Gleichverteilung aus  $D_i$  gezogen.)

Beide Definitionen (ob über eine Familie oder über eine einzelne Funktion) definieren dasselbe Konzept der „Nicht-Umkehrbarkeit“.

Wenn  $f : \{0, 1\}^* \rightarrow \{0, 1\}$  eine One-Way-Funktion ist, dann gilt  $P \neq NP$ .

Der Nachweis, dass irgendeine Funktion eine One-Way-Funktion ist, ist also mindestens so schwierig wie die Trennung von  $P$  und  $NP$ .

Vermutliche One-Way-Funktionen aus der Zahlentheorie:

- 1 **Faktorisierung**: Für zwei Primzahlen  $p$  und  $q$  ist  $N = p \cdot q$  die Ausgabe. Im Umkehrproblem ist die Zahl  $N$  zu faktorisieren.
- 2 Der **diskrete Logarithmus**: Für Primzahl  $p$ , ein erzeugendes Element  $g$  modulo  $p$  und eine natürliche Zahl  $i$  ist die Potenz  $g^i \bmod p$  zu berechnen. Im Umkehrproblem berechne den „Logarithmus“  $i$ , wenn  $p$ ,  $g$  und  $g^i \bmod p$  gegeben sind.
- 3 Die **Rabin-Funktion**: Für Primzahlen  $p, q$  mit  $p, q \equiv 3 \pmod{4}$ , die Zahl  $N = p \cdot q$  sowie eine Restklasse  $x < N$  ist  $(N, x^2 \bmod N)$  zu berechnen. Das Umkehrproblem ist die Bestimmung der Wurzel von  $y = x^2 \bmod N$ .

Man kann zeigen, dass das Umkehrproblem genauso schwierig ist wie die Faktorisierung von  $N$ .

- 4 Das **RSA-Problem**: Für Zahlen  $N = p \cdot q$ ,  $e$  und  $x$  ist  $y := x^e \bmod N$  zu berechnen. Im Umkehrproblem ist  $x$  zu bestimmen, wobei  $y$ ,  $N$  und  $e$  gegeben sind.

Das RSA-Problem ist nicht schwieriger als die Faktorisierung von  $N$ .

Quanten-Schaltkreise können alle obigen One-Way-Funktionen „brechen“, anscheinend aber nicht das LWE-Problem.

### Learning-With-Errors (LWE):

- Sei  $q$  eine Primzahl. Für  $n < m$  ist eine  $n \times m$  Matrix  $A$  mit Einträgen aus  $\mathbb{Z}_q$  sowie ein Fehlervektor  $e \in \mathbb{Z}_q^n$  mit kleiner  $L_2$ -Norm gegeben. Berechne

$$x \in \mathbb{Z}_q^n \mapsto y = x^T \cdot A + e^T \pmod{q}.$$

- Im Umkehrproblem ist die ursprüngliche Nachricht  $x$  aus dem verrauschten Codewort  $y = x^T \cdot A + e^T \pmod{q}$  zu bestimmen.

LWE treffen wir in der Gitter-Kryptographie wieder.

- Pseudo-Random-Generatoren, ✓
  - ▶ Private-Key-Kryptographie,
  - ▶ Derandomisierung,
  - ▶ Pseudo-Random-Funktionen,
    - ★ Kollisionsresistente Hashfunktionen,
    - ★ Nachrichten-Authentifizierung,
- Bit-Commitment,

# Pseudo-Random-Generatoren (PRGs)

PRGs „strecken“ einen wirklichen Zufallsstring von  $n$  auf  $p(n)$  Bits ohne dass ein Unterschied zur Gleichverteilung auffällt.

Sei  $p$  ein Polynom mit  $p(n) > n$ .

- (a) Ein **statistischer Test**  $T$  ist eine Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$ , die Eingaben  $y \in \{0, 1\}^{p(n)}$  akzeptieren oder verwerfen.
- (b) Ein **Generator**  $G : \{0, 1\}^* \rightarrow \{0, 1\}^*$  mit Streckung  $p(n)$  ist ein eff. det. Algorithmus, der für eine Saat  $x \in \{0, 1\}^n$  die Ausgabe  $G(x) \in \{0, 1\}^{p(n)}$  berechnet.
- (c) Sei  $G$  ein Generator mit Streckung  $p$  und sei  $T$  ein statistischer Test. Sei

$$g_n = \Pr_{x \in_R \{0,1\}^n} [ T_n \text{ akzeptiert } G(x) \mid |x| = n ] \text{ und}$$

$$r_n = \Pr_{y \in_R \{0,1\}^{p(n)}} [ T_n \text{ akzeptiert } y \mid |y| = p(n) ].$$

$G$  **besteht den Test**  $T$ , wenn

$$|g_n - r_n|$$

als Funktion von  $n$  vernachlässigbar ist:  $T$  hat einen vernachlässigbaren Vorteil.

# Pseudo-Random-Generatoren (PRGs)

PRGs „strecken“ einen wirklichen Zufallsstring von  $n$  auf  $p(n)$  Bits ohne dass ein Unterschied zur Gleichverteilung auffällt.

Sei  $p$  ein Polynom mit  $p(n) > n$ .

- (a) Ein **statistischer Test**  $T$  ist eine Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$ , die Eingaben  $y \in \{0, 1\}^{p(n)}$  akzeptieren oder verwerfen.
- (b) Ein **Generator**  $G : \{0, 1\}^* \rightarrow \{0, 1\}^*$  mit Streckung  $p(n)$  ist ein eff. det. Algorithmus, der für eine Saat  $x \in \{0, 1\}^n$  die Ausgabe  $G(x) \in \{0, 1\}^{p(n)}$  berechnet.
- (c) Sei  $G$  ein Generator mit Streckung  $p$  und sei  $T$  ein statistischer Test. Sei

$$g_n = \Pr_{x \in_R \{0,1\}^n} [ T_n \text{ akzeptiert } G(x) \mid |x| = n ] \text{ und}$$

$$r_n = \Pr_{y \in_R \{0,1\}^{p(n)}} [ T_n \text{ akzeptiert } y \mid |y| = p(n) ].$$

$G$  **besteht den Test**  $T$ , wenn

$$|g_n - r_n|$$

als Funktion von  $n$  vernachlässigbar ist:  $T$  hat einen vernachlässigbaren Vorteil.

- (d) Ein **Pseudo-Random-Generator (PRG)** besteht jeden statistischen Test.

# PRGs: Eine alternative Definition

Ein Generator  $G$  besteht genau dann alle **Bit-Tests**, wenn

für jeden Test  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$  und jede Ausgabe  $G(x) = y_1 \cdots y_{p(n)}$  für  $x \in \{0, 1\}^n$  gilt, dass

$$\left| \text{pr}_{x \in_R \{0,1\}^n} [ T_n(y_1, \dots, y_i) = y_{i+1} ] - \frac{1}{2} \right|$$

für jedes  $i < p(n)$  vernachlässigbar ist.



# PRGs: Eine alternative Definition

Ein Generator  $G$  besteht genau dann alle **Bit-Tests**, wenn

für jeden Test  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$  und jede Ausgabe  $G(x) = y_1 \cdots y_{p(n)}$  für  $x \in \{0, 1\}^n$  gilt, dass

$$\left| \text{pr}_{x \in_R \{0,1\}^n} [ T_n(y_1, \dots, y_i) = y_{i+1} ] - \frac{1}{2} \right|$$

für jedes  $i < p(n)$  vernachlässigbar ist.

$G : \{0, 1\}^* \rightarrow \{0, 1\}^*$  sei ein Generator mit  $|G(x)| = |G(x')|$  für Eingaben  $x, x'$  gleicher Länge. Dann gilt

$G$  ist ein Pseudo-Random-Generator  $\iff G$  besteht alle Bit-Tests.

# Was tun Tests $T$ ?

- 1 Interpretiere  $T$  wieder als *nicht-uniforme* Folge *effizienter deterministischer* Algorithmen.
- 2  $T$  kann insbesondere den vermeintlichen PRG  $G$  simulieren, denn  $G$  ist effizient.
- 3 Der „Test für den Test“:  
 *$T$  muss entscheiden, ob ein Wort  $y$  von  $G$  erzeugt wurde oder von der Gleichverteilung.*  
  
Bits-Tests sind bereits ausreichend.
- 4 Warum wird verlangt, dass  $T$  effizient ist?

# Was tun Tests $T$ ?

- 1 Interpretiere  $T$  wieder als *nicht-uniforme* Folge *effizienter deterministischer* Algorithmen.
- 2  $T$  kann insbesondere den vermeintlichen PRG  $G$  simulieren, denn  $G$  ist effizient.
- 3 Der „Test für den Test“:

*$T$  muss entscheiden, ob ein Wort  $y$  von  $G$  erzeugt wurde oder von der Gleichverteilung.*

Bits-Tests sind bereits ausreichend.

- 4 Warum wird verlangt, dass  $T$  effizient ist? Definiere den Test  $T$  mit

$$T(y) = \begin{cases} 1 & G \text{ produziert } y \text{ als Ausgabe,} \\ 0 & \text{sonst.} \end{cases}$$

# Was tun Tests $T$ ?

- 1 Interpretiere  $T$  wieder als *nicht-uniforme* Folge *effizienter deterministischer* Algorithmen.
- 2  $T$  kann insbesondere den vermeintlichen PRG  $G$  simulieren, denn  $G$  ist effizient.
- 3 Der „Test für den Test“:

*$T$  muss entscheiden, ob ein Wort  $y$  von  $G$  erzeugt wurde oder von der Gleichverteilung.*

Bits-Tests sind bereits ausreichend.

- 4 Warum wird verlangt, dass  $T$  effizient ist? Definiere den Test  $T$  mit

$$T(y) = \begin{cases} 1 & G \text{ produziert } y \text{ als Ausgabe,} \\ 0 & \text{sonst.} \end{cases}$$

Offensichtlich ist

$$\Pr_{x \in_R \{0,1\}^n} [ T(G(x)) = 1 ] = 1,$$

während ein wirklicher Random-Generator höchstens die W-keit  $\frac{1}{2}$  erzielt.

Der Pseudo-Random-Generator  $G$  möge jede Saat um genau ein Bit strecken.

Sei  $q$  ein beliebiges Polynom. Für  $x \in \{0, 1\}^n$  definiere den Generator

$$G^*(x) = G^{q(n)-n}(x).$$

Dann ist  $G^*$  ein PRG mit Streckung  $q(n)$ .

Der Pseudo-Random-Generator  $G$  möge jede Saat um genau ein Bit strecken.

Sei  $q$  ein beliebiges Polynom. Für  $x \in \{0, 1\}^n$  definiere den Generator

$$G^*(x) = G^{q(n)-n}(x).$$

Dann ist  $G^*$  ein PRG mit Streckung  $q(n)$ .

Angenommen,  $G^*$  besteht den statistischen Test  $T^* = (T_n^* : n \in \mathbb{N})$  **nicht**. Setze

$$g_n^* = \text{pr}_{x \in_R \{0,1\}^n} [ T_n^* \text{ akzeptiert } G^*(x) ] \text{ und}$$

$$r_n = \text{pr}_{y \in_R \{0,1\}^{p(n)}} [ T_n^* \text{ akzeptiert } y ].$$

Da  $G^*$  durchfällt, ist  $|g_n^* - r_n| > n^{-k}$  für irgendein  $k \in \mathbb{N}$  und unendlich viele  $n$ .

Der Pseudo-Random-Generator  $G$  möge jede Saat um genau ein Bit strecken.

Sei  $q$  ein beliebiges Polynom. Für  $x \in \{0, 1\}^n$  definiere den Generator

$$G^*(x) = G^{q(n)-n}(x).$$

Dann ist  $G^*$  ein PRG mit Streckung  $q(n)$ .

Angenommen,  $G^*$  besteht den statistischen Test  $T^* = (T_n^* : n \in \mathbb{N})$  **nicht**. Setze

$$g_n^* = \text{pr}_{x \in_R \{0,1\}^n} [ T_n^* \text{ akzeptiert } G^*(x) ] \text{ und}$$

$$r_n = \text{pr}_{y \in_R \{0,1\}^{p(n)}} [ T_n^* \text{ akzeptiert } y ].$$

Da  $G^*$  durchfällt, ist  $|g_n^* - r_n| > n^{-k}$  für irgendein  $k \in \mathbb{N}$  und unendlich viele  $n$ . Setze

$$p_i = \text{pr}_{y \in_R \{0,1\}^{n+i}} [ T_n^* \text{ akzeptiert } G^{q(n)-(n+i)}(y) ].$$

Dann ist  $p_0 =$

Der Pseudo-Random-Generator  $G$  möge jede Saat um genau ein Bit strecken.

Sei  $q$  ein beliebiges Polynom. Für  $x \in \{0, 1\}^n$  definiere den Generator

$$G^*(x) = G^{q(n)-n}(x).$$

Dann ist  $G^*$  ein PRG mit Streckung  $q(n)$ .

Angenommen,  $G^*$  besteht den statistischen Test  $T^* = (T_n^* : n \in \mathbb{N})$  **nicht**. Setze

$$g_n^* = \text{pr}_{x \in_R \{0,1\}^n} [ T_n^* \text{ akzeptiert } G^*(x) ] \text{ und}$$

$$r_n = \text{pr}_{y \in_R \{0,1\}^{p(n)}} [ T_n^* \text{ akzeptiert } y ].$$

Da  $G^*$  durchfällt, ist  $|g_n^* - r_n| > n^{-k}$  für irgendein  $k \in \mathbb{N}$  und unendlich viele  $n$ . Setze

$$p_i = \text{pr}_{y \in_R \{0,1\}^{n+i}} [ T_n^* \text{ akzeptiert } G^{q(n)-(n+i)}(y) ].$$

Dann ist  $p_0 = g_n^*$  und  $p_{q(n)-n} =$



Der Pseudo-Random-Generator  $G$  möge jede Saat um genau ein Bit strecken.

Sei  $q$  ein beliebiges Polynom. Für  $x \in \{0, 1\}^n$  definiere den Generator

$$G^*(x) = G^{q(n)-n}(x).$$

Dann ist  $G^*$  ein PRG mit Streckung  $q(n)$ .

Angenommen,  $G^*$  besteht den statistischen Test  $T^* = (T_n^* : n \in \mathbb{N})$  **nicht**. Setze

$$\begin{aligned} g_n^* &= \text{pr}_{x \in_R \{0,1\}^n} [ T_n^* \text{ akzeptiert } G^*(x) ] \text{ und} \\ r_n &= \text{pr}_{y \in_R \{0,1\}^{p(n)}} [ T_n^* \text{ akzeptiert } y ]. \end{aligned}$$

Da  $G^*$  durchfällt, ist  $|g_n^* - r_n| > n^{-k}$  für irgendein  $k \in \mathbb{N}$  und unendlich viele  $n$ . Setze

$$p_i = \text{pr}_{y \in_R \{0,1\}^{n+i}} [ T_n^* \text{ akzeptiert } G^{q(n)-(n+i)}(y) ].$$

Dann ist  $p_0 = g_n^*$  und  $p_{q(n)-n} = r_n$ . Also gibt es  $i$  mit  $|p_i - p_{i+1}| > \frac{n^{-k}}{q(n)}$

**Zeige:** Für  $y \in \{0, 1\}^{n+i+1}$  fällt  $G$  durch den Test  $T_{n+i}(y) := T_n^*(G^{q(n)-(n+i+1)}(y))$ .

Es ist  $T_{n+i}(y) := T_n^*(G^{q(n)-(n+i+1)}(y))$ .

(a) Für  $x \in \{0, 1\}^{n+i}$  gilt

$$T_{n+i} \text{ akzeptiert } y = G(x) \iff T_n^* \text{ akzeptiert } z = G^{q(n)-(n+i+1)}(G(x))$$

und letzteres geschieht mit Wahrscheinlichkeit  $p_i$ .

Es ist  $T_{n+i}(y) := T_n^*(G^{q(n)-(n+i+1)}(y))$ .

(a) Für  $x \in \{0, 1\}^{n+i}$  gilt

$$T_{n+i} \text{ akzeptiert } y = G(x) \iff T_n^* \text{ akzeptiert } z = G^{q(n)-(n+i+1)}(G(x))$$

und letzteres geschieht mit Wahrscheinlichkeit  $p_i$ .

(b) Für  $y \in \{0, 1\}^{n+i+1}$  gilt

$$T_{n+i} \text{ akzeptiert } y \iff T_n^* \text{ akzeptiert } z = G^{q(n)-(n+i+1)}(y)$$

und letzteres geschieht mit Wahrscheinlichkeit  $p_{i+1}$ .

Es ist  $T_{n+i}(y) := T_n^*(G^{q(n)-(n+i+1)}(y))$ .

(a) Für  $x \in \{0, 1\}^{n+i}$  gilt

$$T_{n+i} \text{ akzeptiert } y = G(x) \iff T_n^* \text{ akzeptiert } z = G^{q(n)-(n+i+1)}(G(x))$$

und letzteres geschieht mit Wahrscheinlichkeit  $p_i$ .

(b) Für  $y \in \{0, 1\}^{n+i+1}$  gilt

$$T_{n+i} \text{ akzeptiert } y \iff T_n^* \text{ akzeptiert } z = G^{q(n)-(n+i+1)}(y)$$

und letzteres geschieht mit Wahrscheinlichkeit  $p_{i+1}$ .

Und  $G$  besteht tatsächlich den Test  $T_{n+i}$  **nicht**, denn  $|p_i - p_{i+1}| > \frac{n^{-k}}{q(n)}$ .

Wir haben ein **Hybrid-Argument** benutzt, das den PRG in kleinen Schritten in den RG überführt.

# Der Generator der linearen Kongruenzen

Der Generator

$$G_{m,a,b}(x) := a \cdot x + b \pmod{m}$$

erzeugt die Folge

$$x_0 := s, \quad x_{k+1} := G_{m,a,b}(x_k).$$

Dieser Generator wird häufig benutzt, ist aber leider **kein** PRG.

# Der Generator der linearen Kongruenzen

Der Generator

$$G_{m,a,b}(x) := a \cdot x + b \pmod{m}$$

erzeugt die Folge

$$x_0 := s, \quad x_{k+1} := G_{m,a,b}(x_k).$$

Dieser Generator wird häufig benutzt, ist aber leider **kein** PRG.

Beispielhaft eine Schwäche der Generatoren:

- Der Satz von Marsaglia besagt für jedes  $k$ : Alle  $k$ -Tupel

$$\left( \frac{x_{i+1}}{m}, \dots, \frac{x_{i+k}}{m} \right)$$

liegen auf höchstens

$$(m \cdot k!)^{1/k} =$$

# Der Generator der linearen Kongruenzen

Der Generator

$$G_{m,a,b}(x) := a \cdot x + b \pmod{m}$$

erzeugt die Folge

$$x_0 := s, \quad x_{k+1} := G_{m,a,b}(x_k).$$

Dieser Generator wird häufig benutzt, ist aber leider **kein** PRG.

Beispielhaft eine Schwäche der Generatoren:

- Der Satz von Marsaglia besagt für jedes  $k$ : Alle  $k$ -Tupel

$$\left( \frac{x_{i+1}}{m}, \dots, \frac{x_{i+k}}{m} \right)$$

liegen auf höchstens

$$(m \cdot k!)^{1/k} = 2^{\mathcal{O}(\log m/k + \log k)}$$

parallelen Hyperebenen.

- Wähle z.B.  $k = \mathcal{O}(\log m)$ .

# PRGs aus One-Way-Funktionen

- (a) Der **Blum-Micali-Generator** wählt eine große Primzahl  $p$  und eine erzeugende Restklasse  $g$  modulo  $p$ . Für eine Saat  $s_0$  und die Iteration

$$s_{i+1} = g^{s_i} \bmod p$$

berechne die Bitfolge  $(b_1, \dots, b_m)$  mit  $b_i = \begin{cases} 1 & \text{wenn } s_i < p/2 \\ 0 & \text{sonst.} \end{cases}$

Der Generator basiert auf dem diskreten Logarithmus  $x \mapsto g^x \bmod p$  als One-Way-Funktion.



# PRGs aus One-Way-Funktionen

- (a) Der **Blum-Micali-Generator** wählt eine große Primzahl  $p$  und eine erzeugende Restklasse  $g$  modulo  $p$ . Für eine Saat  $s_0$  und die Iteration

$$s_{i+1} = g^{s_i} \bmod p$$

berechne die Bitfolge  $(b_1, \dots, b_m)$  mit  $b_i = \begin{cases} 1 & \text{wenn } s_i < p/2 \\ 0 & \text{sonst.} \end{cases}$

Der Generator basiert auf dem diskreten Logarithmus  $x \mapsto g^x \bmod p$  als One-Way-Funktion.

- (b) Der **RSA-Generator** berechnet zuerst für eine Saat  $s_0$  die Folge

$$s_{i+1} = s_i^e \bmod N$$

für  $N = p \cdot q$  und gibt dann die Bitfolge  $(s_1 \bmod 2, \dots, s_m \bmod 2)$  aus.

# PRGs aus One-Way-Funktionen

- (a) Der **Blum-Micali-Generator** wählt eine große Primzahl  $p$  und eine erzeugende Restklasse  $g$  modulo  $p$ . Für eine Saat  $s_0$  und die Iteration

$$s_{i+1} = g^{s_i} \bmod p$$

berechne die Bitfolge  $(b_1, \dots, b_m)$  mit  $b_i = \begin{cases} 1 & \text{wenn } s_i < p/2 \\ 0 & \text{sonst.} \end{cases}$

Der Generator basiert auf dem diskreten Logarithmus  $x \mapsto g^x \bmod p$  als One-Way-Funktion.

- (b) Der **RSA-Generator** berechnet zuerst für eine Saat  $s_0$  die Folge

$$s_{i+1} = s_i^e \bmod N$$

für  $N = p \cdot q$  und gibt dann die Bitfolge  $(s_1 \bmod 2, \dots, s_m \bmod 2)$  aus.

- (c) Der **Rabin-Generator** berechnet zuerst für eine Saat  $s_0$  die Folge

$$s_{i+1} = s_i^2 \bmod N \quad \text{für } N = p \cdot q \text{ mit Primzahlen } p \equiv q \equiv 3 \pmod{4}$$

und gibt die Bitfolge  $(s_1 \bmod 2, \dots, s_m \bmod 2)$  aus.

Der Rabin-Generator beruht auf der Rabin-Funktion  $x \mapsto x^2 \bmod N$  als One-Way-Funktion.

# One-Way-Funktionen versus Pseudo-Random-Generatoren

# Existenz von One-Way-Funktionen und PRGs

- (a) Wenn  $G$  ein PRG mit Streckung  $2 \cdot n$  ist, dann ist  $G$  eine One-Way-Funktion.
- (b) Wenn es One-Way-Funktionen gibt, dann gibt es PRGs.

(a)  $G$  sei ein PRG mit Streckung  $2n$ . Angenommen,  $G$  ist **keine** One-Way-Funktion.

- Es gibt eine Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$  und  $T_n$  invertiert eine Eingabe  $y = G(x)$  für  $x \in \{0, 1\}^n$  mit W-keit  $\geq \frac{1}{\text{poly}(n)}$ .
- Ein neuer Test  $T_n^*$  für den PRG  $G$ :

# Existenz von One-Way-Funktionen und PRGs

- (a) Wenn  $G$  ein PRG mit Streckung  $2 \cdot n$  ist, dann ist  $G$  eine One-Way-Funktion.
- (b) Wenn es One-Way-Funktionen gibt, dann gibt es PRGs.

(a)  $G$  sei ein PRG mit Streckung  $2n$ . Angenommen,  $G$  ist **keine** One-Way-Funktion.

- Es gibt eine Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$  und  $T_n$  invertiert eine Eingabe  $y = G(x)$  für  $x \in \{0, 1\}^n$  mit W-keit  $\geq \frac{1}{\text{poly}(n)}$ .
- Ein neuer Test  $T_n^*$  für den PRG  $G$ : Wende  $T_n$  auf einen String  $y \in \{0, 1\}^{2n}$  an und akzeptiere genau dann, wenn  $T_n$  ein  $x'$  mit  $G(x') = y$  bestimmt.

# Existenz von One-Way-Funktionen und PRGs

- (a) Wenn  $G$  ein PRG mit Streckung  $2 \cdot n$  ist, dann ist  $G$  eine One-Way-Funktion.
- (b) Wenn es One-Way-Funktionen gibt, dann gibt es PRGs.

(a)  $G$  sei ein PRG mit Streckung  $2n$ . Angenommen,  $G$  ist **keine** One-Way-Funktion.

- Es gibt eine Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$  und  $T_n$  invertiert eine Eingabe  $y = G(x)$  für  $x \in \{0, 1\}^n$  mit W-keit  $\geq \frac{1}{\text{poly}(n)}$ .
- Ein neuer Test  $T_n^*$  für den PRG  $G$ : Wende  $T_n$  auf einen String  $y \in \{0, 1\}^{2n}$  an und akzeptiere genau dann, wenn  $T_n$  ein  $x'$  mit  $G(x') = y$  bestimmt.
  - ▶ Für einen zufälligen String  $y \in \{0, 1\}^{2n}$  ist  $T_n^*(y) = 1$  mit einer W-keit von

# Existenz von One-Way-Funktionen und PRGs

- (a) Wenn  $G$  ein PRG mit Streckung  $2 \cdot n$  ist, dann ist  $G$  eine One-Way-Funktion.
- (b) Wenn es One-Way-Funktionen gibt, dann gibt es PRGs.

(a)  $G$  sei ein PRG mit Streckung  $2n$ . Angenommen,  $G$  ist **keine** One-Way-Funktion.

- Es gibt eine Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$  und  $T_n$  invertiert eine Eingabe  $y = G(x)$  für  $x \in \{0, 1\}^n$  mit W-keit  $\geq \frac{1}{\text{poly}(n)}$ .
- Ein neuer Test  $T_n^*$  für den PRG  $G$ : Wende  $T_n$  auf einen String  $y \in \{0, 1\}^{2n}$  an und akzeptiere genau dann, wenn  $T_n$  ein  $x'$  mit  $G(x') = y$  bestimmt.
  - ▶ Für einen zufälligen String  $y \in \{0, 1\}^{2n}$  ist  $T_n^*(y) = 1$  mit einer W-keit von höchstens  $\frac{2^n}{2^{2n}} = \frac{1}{2^n}$ .
  - ▶ Ist hingegen  $y$  eine Ausgabe von  $G$ , dann ist  $T_n^*(y) = 1$  mit einer W-keit von

# Existenz von One-Way-Funktionen und PRGs

- (a) Wenn  $G$  ein PRG mit Streckung  $2 \cdot n$  ist, dann ist  $G$  eine One-Way-Funktion.
- (b) Wenn es One-Way-Funktionen gibt, dann gibt es PRGs.

(a)  $G$  sei ein PRG mit Streckung  $2n$ . Angenommen,  $G$  ist **keine** One-Way-Funktion.

- Es gibt eine Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$  und  $T_n$  invertiert eine Eingabe  $y = G(x)$  für  $x \in \{0, 1\}^n$  mit W-keit  $\geq \frac{1}{\text{poly}(n)}$ .
- Ein neuer Test  $T_n^*$  für den PRG  $G$ : Wende  $T_n$  auf einen String  $y \in \{0, 1\}^{2n}$  an und akzeptiere genau dann, wenn  $T_n$  ein  $x'$  mit  $G(x') = y$  bestimmt.
  - ▶ Für einen zufälligen String  $y \in \{0, 1\}^{2n}$  ist  $T_n^*(y) = 1$  mit einer W-keit von höchstens  $\frac{2^n}{2^{2n}} = \frac{1}{2^n}$ .
  - ▶ Ist hingegen  $y$  eine Ausgabe von  $G$ , dann ist  $T_n^*(y) = 1$  mit einer W-keit von **mindestens**  $\frac{1}{\text{poly}(n)}$ .
- $T_n^*$  ist effizient auswertbar und unterscheidet zwischen  $G$  und einem wirklichen Zufallsgenerator. Also ist  $G$  kein PRG: **Widerspruch**.



# Existenz von One-Way-Funktionen und PRGs

- (a) Wenn  $G$  ein PRG mit Streckung  $2 \cdot n$  ist, dann ist  $G$  eine One-Way-Funktion.
- (b) Wenn es One-Way-Funktionen gibt, dann gibt es PRGs.

(a)  $G$  sei ein PRG mit Streckung  $2n$ . Angenommen,  $G$  ist **keine** One-Way-Funktion.

- Es gibt eine Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$  und  $T_n$  invertiert eine Eingabe  $y = G(x)$  für  $x \in \{0, 1\}^n$  mit W-keit  $\geq \frac{1}{\text{poly}(n)}$ .
- Ein neuer Test  $T_n^*$  für den PRG  $G$ : Wende  $T_n$  auf einen String  $y \in \{0, 1\}^{2n}$  an und akzeptiere genau dann, wenn  $T_n$  ein  $x'$  mit  $G(x') = y$  bestimmt.
  - ▶ Für einen zufälligen String  $y \in \{0, 1\}^{2n}$  ist  $T_n^*(y) = 1$  mit einer W-keit von höchstens  $\frac{2^n}{2^{2n}} = \frac{1}{2^n}$ .
  - ▶ Ist hingegen  $y$  eine Ausgabe von  $G$ , dann ist  $T_n^*(y) = 1$  mit einer W-keit von **mindestens**  $\frac{1}{\text{poly}(n)}$ .
- $T_n^*$  ist effizient auswertbar und unterscheidet zwischen  $G$  und einem wirklichen Zufallsgenerator. Also ist  $G$  kein PRG: **Widerspruch**.

(b) Die Konstruktion eines PRGs aus einer One-Way-Funktion ist weitaus schwieriger.

# One-Way-Permutationen und Hard-Core-Bits

Die Konstruktion eines Pseudo-Random-Generators für eine vorgegebene **One-Way-Permutation** ist einfacher.

### Der Satz von Goldreich und Levin

Sei  $F : \{0, 1\}^* \rightarrow \{0, 1\}^*$  eine **One-Way-Permutation**. Dann ist

$$\left| \Pr_{x, r \in_R \{0, 1\}^n} [ T_n(F(x), r) = \underbrace{\bigoplus_{i=1}^n (x_i \wedge r_i)}_{\text{ein Hard-Core-Bit bezüglich } F} ] - \frac{1}{2} \right|$$

Die Konstruktion eines Pseudo-Random-Generators für eine vorgegebene **One-Way-Permutation** ist einfacher.

### Der Satz von Goldreich und Levin

Sei  $F : \{0, 1\}^* \rightarrow \{0, 1\}^*$  eine **One-Way-Permutation**. Dann ist

$$\left| \Pr_{x, r \in_R \{0, 1\}^n} [ T_n(F(x), r) = \underbrace{\bigoplus_{i=1}^n (x_i \wedge r_i)}_{\text{ein Hard-Core-Bit bezüglich } F} ] - \frac{1}{2} \right|$$

für jede Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  polynomieller Größe als Funktion von  $n$  vernachlässigbar.

Sogar auf den ersten Blick „einfache“ Eigenschaften der Lösung  $x$  lassen sich nur mit vernachlässigbarem Vorteil ableiten.

Als Konsequenz des Satzes von Goldreich und Levin:

Sei  $F : \{0, 1\}^* \rightarrow \{0, 1\}^*$  eine **One-Way-Permutation**. Dann ist die Funktion

$$G : \bigcup_{n=1}^{\infty} \{0, 1\}^{2n} \rightarrow \{0, 1\}^*$$

mit

$$G(x, r) := (F(x), r, \bigoplus_{i=1}^n (x_i \wedge r_i))$$

ein PRG mit Streckung  $N + 1$  für eine Saat  $(x, r)$  mit  $|x| = |r| = N/2$ .

Als Konsequenz des Satzes von Goldreich und Levin:

Sei  $F : \{0, 1\}^* \rightarrow \{0, 1\}^*$  eine **One-Way-Permutation**. Dann ist die Funktion

$$G : \bigcup_{n=1}^{\infty} \{0, 1\}^{2n} \rightarrow \{0, 1\}^*$$

mit

$$G(x, r) := (F(x), r, \oplus_{i=1}^n (x_i \wedge r_i))$$

ein PRG mit Streckung  $N + 1$  für eine Saat  $(x, r)$  mit  $|x| = |r| = N/2$ .

- $F$  ist bijektiv und  $x$  ist zufällig  $\implies F(x)$  ist zufällig.
- $r$  ist Teil der Saat  $(x, r)$  und  $(F(x), r)$  ist ein Zufallstring der Länge  $2|x|$ .

- Pseudo-Random-Generatoren, ✓
  - ▶ Private-Key-Kryptographie,
  - ▶ Derandomisierung,
  - ▶ Pseudo-Random-Funktionen,
    - ★ Kollisionsresistente Hashfunktionen,
    - ★ Nachrichten-Authentifizierung,
- **Bit-Commitment, ✓**

# Ein Protokoll für Bit-Commitment

Wie können sich Alice und Bob über Telefon auf ein **zufälliges Bit** einigen, wenn man sich gegenseitig nicht über den Weg traut?

Bit-Commitment mit einer One-Way-Permutation.

1. Bob wählt zwei Strings  $x, r \in \{0, 1\}^n$  und schickt Nachricht  $(F(x), r)$  an Alice:  
Bob legt sich somit auf  $x$  fest, ohne  $x$  zu veröffentlichen.



# Ein Protokoll für Bit-Commitment

Wie können sich Alice und Bob über Telefon auf ein **zufälliges Bit** einigen, wenn man sich gegenseitig nicht über den Weg traut?

Bit-Commitment mit einer One-Way-Permutation.

1. Bob wählt zwei Strings  $x, r \in \{0, 1\}^n$  und schickt Nachricht  $(F(x), r)$  an Alice:  
Bob legt sich somit auf  $x$  fest, ohne  $x$  zu veröffentlichen.
2. Alice antwortet durch das Versenden eines Bits  $b$ .
3. Bob sendet  $x$ . Beide einigen sich auf das Bit

$$b' := \left( \bigoplus_{i=1}^n (x_i \wedge r_i) \right) \oplus b.$$

# Ein Protokoll für Bit-Commitment

Wie können sich Alice und Bob über Telefon auf ein **zufälliges Bit** einigen, wenn man sich gegenseitig nicht über den Weg traut?

Bit-Commitment mit einer One-Way-Permutation.

1. Bob wählt zwei Strings  $x, r \in \{0, 1\}^n$  und schickt Nachricht  $(F(x), r)$  an Alice:  
Bob legt sich somit auf  $x$  fest, ohne  $x$  zu veröffentlichen.
2. Alice antwortet durch das Versenden eines Bits  $b$ .
3. Bob sendet  $x$ . Beide einigen sich auf das Bit

$$b' := \left( \bigoplus_{i=1}^n (x_i \wedge r_i) \right) \oplus b.$$

- ▶ Alice kann  $F$  auswerten und sich davon überzeugen, dass sich Bob in seiner ersten Nachricht tatsächlich auf  $x$  festgelegt hat. (Hier wird benutzt, dass  $F$  bijektiv ist.)

# Ein Protokoll für Bit-Commitment

Wie können sich Alice und Bob über Telefon auf ein **zufälliges Bit** einigen, wenn man sich gegenseitig nicht über den Weg traut?

Bit-Commitment mit einer One-Way-Permutation.

1. Bob wählt zwei Strings  $x, r \in \{0, 1\}^n$  und schickt Nachricht  $(F(x), r)$  an Alice:  
Bob legt sich somit auf  $x$  fest, ohne  $x$  zu veröffentlichen.
2. Alice antwortet durch das Versenden eines Bits  $b$ .
3. Bob sendet  $x$ . Beide einigen sich auf das Bit

$$b' := \left( \bigoplus_{i=1}^n (x_i \wedge r_i) \right) \oplus b.$$

- ▶ Alice kann  $F$  auswerten und sich davon überzeugen, dass sich Bob in seiner ersten Nachricht tatsächlich auf  $x$  festgelegt hat. (Hier wird benutzt, dass  $F$  bijektiv ist.)
- ▶ Kann Alice das Bit  $\bigoplus_{i=1}^n (x_i \wedge y_i)$  nicht mgl. mit Hilfe von  $F(x)$  voraussagen?

# Ein Protokoll für Bit-Commitment

Wie können sich Alice und Bob über Telefon auf ein **zufälliges Bit** einigen, wenn man sich gegenseitig nicht über den Weg traut?

Bit-Commitment mit einer One-Way-Permutation.

1. Bob wählt zwei Strings  $x, r \in \{0, 1\}^n$  und schickt Nachricht  $(F(x), r)$  an Alice:  
Bob legt sich somit auf  $x$  fest, ohne  $x$  zu veröffentlichen.
2. Alice antwortet durch das Versenden eines Bits  $b$ .
3. Bob sendet  $x$ . Beide einigen sich auf das Bit

$$b' := \left( \bigoplus_{i=1}^n (x_i \wedge r_i) \right) \oplus b.$$

- ▶ Alice kann  $F$  auswerten und sich davon überzeugen, dass sich Bob in seiner ersten Nachricht tatsächlich auf  $x$  festgelegt hat. (Hier wird benutzt, dass  $F$  bijektiv ist.)
- ▶ Kann Alice das Bit  $\bigoplus_{i=1}^n (x_i \wedge y_i)$  nicht mgl. mit Hilfe von  $F(x)$  voraussagen?
  - ★ Das Goldreich-Levin-Theorem sagt NEIN!
- ▶ Alice und Bob haben dieselbe Ausgangslage. Hat Alice ein zufälliges Bit  $b$  oder Bob zufällige Strings  $x, y$  ausgewürfelt, dann ist Bit  $b'$  zufällig.

- Pseudo-Random-Generatoren, ✓
  - ▶ Private-Key-Kryptographie, ✓
  - ▶ Derandomisierung,
  - ▶ Pseudo-Random-Funktionen,
    - ★ Kollisionsresistente Hashfunktionen,
    - ★ Nachrichten-Authentifizierung,
- Bit-Commitment, ✓

# Private-Key-Kryptographie

Alice und Bob möchten Nachrichten über einen öffentlichen Kanal austauschen. Sie haben einen **privaten**, nur ihnen bekannten Schlüssel  $z$  *zufällig* gleichverteilt ausgewürfelt.

1. In einem **One-Time-Pad** verschickt Bob die Nachricht  $y := x \oplus z$ .
2. Da  $x = y \oplus z$ , kann Alice die ursprüngliche Nachricht  $x$  bestimmen.

# Private-Key-Kryptographie

Alice und Bob möchten Nachrichten über einen öffentlichen Kanal austauschen. Sie haben einen **privaten**, nur ihnen bekannten Schlüssel  $z$  *zufällig* gleichverteilt ausgewürfelt.

1. In einem **One-Time-Pad** verschickt Bob die Nachricht  $y := x \oplus z$ .
2. Da  $x = y \oplus z$ , kann Alice die ursprüngliche Nachricht  $x$  bestimmen.

Ein „Man-In-The-Middle“ hat keinerlei Vorteil, da jeder Wert für  $y$  die gleiche W-keit besitzt gesendet zu werden: Das One-Time-Pad ist *informationstheoretisch sicher*.

# Private-Key-Kryptographie

Alice und Bob möchten Nachrichten über einen öffentlichen Kanal austauschen. Sie haben einen **privaten**, nur ihnen bekannten Schlüssel  $z$  *zufällig* gleichverteilt ausgewürfelt.

1. In einem **One-Time-Pad** verschickt Bob die Nachricht  $y := x \oplus z$ .
2. Da  $x = y \oplus z$ , kann Alice die ursprüngliche Nachricht  $x$  bestimmen.

Ein „Man-In-The-Middle“ hat keinerlei Vorteil, da jeder Wert für  $y$  die gleiche W-keit besitzt gesendet zu werden: Das One-Time-Pad ist *informationstheoretisch sicher*.

(a) **Allgemein**: Angenommen, Bob schickt die verschlüsselte Nachricht

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

an Alice, die Nachricht  $x \in \{0, 1\}^*$  mit der Funktion  $h$  wiederherstellt, d.h. es gilt

$$h(f(x, z), z) = x.$$

**Zeige**: Wenn das Verschlüsselungsschema informationstheoretisch sicher ist, dann muss der private Schlüssel  $z$  mindestens so lang wie die Nachricht  $x$  sein.

(b) Was tun, wenn Alice und Bob **wiederholt** Nachrichten austauschen möchten?



# Private-Key-Kryptographie

Alice und Bob möchten Nachrichten über einen öffentlichen Kanal austauschen. Sie haben einen **privaten**, nur ihnen bekannten Schlüssel  $z$  *zufällig* gleichverteilt ausgewürfelt.

1. In einem **One-Time-Pad** verschickt Bob die Nachricht  $y := x \oplus z$ .
2. Da  $x = y \oplus z$ , kann Alice die ursprüngliche Nachricht  $x$  bestimmen.

Ein „Man-In-The-Middle“ hat keinerlei Vorteil, da jeder Wert für  $y$  die gleiche W-keit besitzt gesendet zu werden: Das One-Time-Pad ist *informationstheoretisch sicher*.

(a) **Allgemein**: Angenommen, Bob schickt die verschlüsselte Nachricht

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

an Alice, die Nachricht  $x \in \{0, 1\}^*$  mit der Funktion  $h$  wiederherstellt, d.h. es gilt

$$h(f(x, z), z) = x.$$

**Zeige**: Wenn das Verschlüsselungsschema informationstheoretisch sicher ist, dann muss der private Schlüssel  $z$  mindestens so lang wie die Nachricht  $x$  sein.

- (b) Was tun, wenn Alice und Bob **wiederholt** Nachrichten austauschen möchten?
- ▶ Alice und Bob arbeiten mit einem PRG  $G$  und verwenden  $G(z)$  statt  $z$ .
  - ▶ Ersetze informationstheoretische durch algorithmische Sicherheit.

- Pseudo-Random-Generatoren, ✓
  - ▶ Private-Key-Kryptographie, ✓
  - ▶ **Derandomisierung**, ✓
  - ▶ Pseudo-Random-Funktionen,
    - ★ Kollisionsresistente Hashfunktionen,
    - ★ Nachrichten-Authentifizierung,
- Bit-Commitment, ✓

Wenn es einen Pseudo-Random-Generator gibt, dann gilt für jedes  $\epsilon > 0$

$$\text{BPP} \subseteq \bigcap_{\epsilon > 0} \text{DTIME}(2^{n^\epsilon}).$$

Der randomisierte Algorithmus  $R$  akzeptiere  $L \in \text{BPP}$  in Worst-Case-Laufzeit  $\mathcal{O}(n^k)$ .

Nach Annahme: Es gibt einen PRG  $G_{\epsilon,k}$ , der  $n^\epsilon$  Zufallsbits auf Länge  $\Theta(n^k)$  streckt.

Für eine Eingabe  $w$  der Länge  $n$  wird  $R$  auf Eingabe  $w$  deterministisch simuliert:

Wenn es einen Pseudo-Random-Generator gibt, dann gilt für jedes  $\epsilon > 0$

$$\text{BPP} \subseteq \bigcap_{\epsilon > 0} \text{DTIME}(2^{n^\epsilon}).$$

Der randomisierte Algorithmus  $R$  akzeptiere  $L \in \text{BPP}$  in Worst-Case-Laufzeit  $\mathcal{O}(n^k)$ .  
Nach Annahme: Es gibt einen PRG  $G_{\epsilon,k}$ , der  $n^\epsilon$  Zufallsbits auf Länge  $\Theta(n^k)$  streckt.

Für eine Eingabe  $w$  der Länge  $n$  wird  $R$  auf Eingabe  $w$  deterministisch simuliert:

1. Zuerst werden nacheinander alle  $2^{n^\epsilon}$  binären Wörter  $r$  der Länge  $n^\epsilon$  produziert:
  - ▶  $G_{\epsilon,k}$  streckt  $r$  auf das Wort  $r^*$  der Länge  $\mathcal{O}(n^k)$ .
  - ▶ Simuliere  $R$  auf  $w$  mit den Zufallsbits in  $r^*$ .
2. Akzeptiere Eingabe  $w \iff$  mehr als die Hälfte aller Simulationen akzeptieren.

Wir erhalten einen effizienten **nicht-uniformen** Test  $T_n$  (mit „Orakelstring“  $w$ ):

Wenn es einen Pseudo-Random-Generator gibt, dann gilt für jedes  $\epsilon > 0$

$$\text{BPP} \subseteq \bigcap_{\epsilon > 0} \text{DTIME}(2^{n^\epsilon}).$$

Der randomisierte Algorithmus  $R$  akzeptiere  $L \in \text{BPP}$  in Worst-Case-Laufzeit  $\mathcal{O}(n^k)$ .  
Nach Annahme: Es gibt einen PRG  $G_{\epsilon,k}$ , der  $n^\epsilon$  Zufallsbits auf Länge  $\Theta(n^k)$  streckt.

Für eine Eingabe  $w$  der Länge  $n$  wird  $R$  auf Eingabe  $w$  deterministisch simuliert:

1. Zuerst werden nacheinander alle  $2^{n^\epsilon}$  binären Wörter  $r$  der Länge  $n^\epsilon$  produziert:
  - ▶  $G_{\epsilon,k}$  streckt  $r$  auf das Wort  $r^*$  der Länge  $\mathcal{O}(n^k)$ .
  - ▶ Simuliere  $R$  auf  $w$  mit den Zufallsbits in  $r^*$ .
2. Akzeptiere Eingabe  $w \iff$  mehr als die Hälfte aller Simulationen akzeptieren.

Wir erhalten einen effizienten **nicht-uniformen** Test  $T_n$  (mit „Orakelstring“  $w$ ):

- $T_n$  simuliert  $R$  auf Eingabe  $w$  mit  $G_{\epsilon,k}(r)$  als Zufallsbits (für eine zufällige Saat  $r$ ).
- Der PRG  $G_{\epsilon,k}$  besteht den Test und die deterministische Simulation gelingt!

# Was passiert bei Super-PRGs?

## Satz von Impagliazzo und Wigderson

Wenn es eine Sprache  $L \in E$  gibt, so dass  $L$  nur durch Schaltkreise der Größe  $2^{\Omega(n)}$  berechenbar ist, dann folgt

$$P = BPP.$$

Führt die Nicht-Uniformität in Schaltkreisen nicht zu einer drastischen Ressourcen-Reduktion, dann gibt es PRGs, die eine zufällige Saat logarithmischer Länge auf polynomielle Länge strecken können, ohne dass es auffällt!

Existiert eine solche Sprache  $L$ ? Vermutlich ja, z.B.  $L = \text{KNF-SAT}$ .

# Fine-Grained-Complexity

- Die **Exponential-Time-Hypothesis (ETH)** besagt, dass 3-SAT für Formeln mit  $n$  Variablen nur Algorithmen mit Laufzeit  $2^{\Theta(n)}$  besitzt.
- Man spricht von der **Strong-Exponential-Time-Hypothesis (SETH)**, wenn sogar Laufzeit  $2^{n-o(n)}$  notwendig ist.

Für den Satz von Impagliazzo und Wigderson genügt die nicht-uniforme ETH-Variante.

- 1 Gilt ETH, dann
  - ▶ besitzen Färbungsproblem, Independent Set und Vertex Cover nur Algorithmen mit Laufzeit  $2^{\Omega(n)}$ ,
  - ▶ werden Cliques der Größe  $k$  im Worst-Case nicht von Algorithmen mit Laufzeit  $n^{o(k)}$  gefunden.
- 2 Neben ETH und SETH werden auch Konsequenzen der Fast-Optimalität von Brute-Force-Lösungen für
  - ▶ 3-Sum („bestimme 3 aus  $n$  Zahlen mit Summe 0“),
  - ▶ Orthogonal Vectors („bestimme 2 aus  $n$  Vektoren, die senkr. auf. stehen“)
  - ▶ All-Pairs-Shortest-Path („bestimme kürzeste Wege zwischen allen Knoten“)

untersucht.

- Pseudo-Random-Generatoren, ✓
  - ▶ Private-Key-Kryptographie, ✓
  - ▶ Derandomisierung, ✓
  - ▶ **Pseudo-Random-Funktionen, ✓**
    - ★ Kollisionsresistente Hashfunktionen,
    - ★ Nachrichten-Authentifizierung,
- Bit-Commitment, ✓



# Pseudo-Random-Funktionen

Eine Familie  $\mathcal{F} = (f_\ell \mid \ell \in \{0, 1\}^*)$  mit Funktionen  $f_\ell : \{0, 1\}^{|\ell|} \rightarrow \{0, 1\}^{|\ell|}$  heißt eine Familie von **Pseudo-Random-Funktionen (PRFs)** genau dann, wenn gilt:

- (a)  $\mathcal{F}$  ist effizient auswertbar: Es gibt einen effizienten deterministischen Algorithmus, der  $f_\ell(x)$  in Zeit polynomiell in  $|\ell|$  bestimmt.
- (b)  $\mathcal{F}$  ist nicht effizient von Zufallsfunktionen unterscheidbar: Für jede Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen polynomieller Größe ist

$$| \text{pr}_{\ell \in_R \{0,1\}^n} [T_n(f_\ell) = 1] - \text{pr}_{f: \{0,1\}^n \rightarrow \{0,1\}^n} [T_n(f) = 1] |$$

als Funktion von  $n$  vernachlässigbar. Für  $g \in \{f_\ell, f\}$  darf  $T_n(g)$  ein **Orakel** für  $g$  mehrfach, aber höchstens polynomiell (in  $n = |\ell|$ ) Mal befragen:

- ▶ Für eine Anfrage  $x$  wird das Orakel mit  $g(x)$  antworten.

# Pseudo-Random-Funktionen aus PRGs

Wie baut man Pseudo-Zufallsfunktionen aus einem PRG  $G$ ?

- 1 O.B.d.A. hat  $G$  die Streckung  $2n$ : Für  $z \in \{0, 1\}^*$  ist

$$G(z) = G_0(z) \cdot G_1(z)$$

wobei  $G_0(z)$  die erste Hälfte und  $G_1(z)$  die zweite Hälfte von  $G(z)$  ist.

- 2 Für alle  $n, m \in \mathbb{N}$  und jedes  $x = x_1 \cdots x_m \in \{0, 1\}^m$  definiere die Funktion

$$f_\ell^G : \{0, 1\}^m \rightarrow \{0, 1\}^n$$

durch

$$f_\ell^G(x) :=$$

# Pseudo-Random-Funktionen aus PRGs

Wie baut man Pseudo-Zufallsfunktionen aus einem PRG  $G$ ?

- 1 O.B.d.A. hat  $G$  die Streckung  $2n$ : Für  $z \in \{0, 1\}^*$  ist

$$G(z) = G_0(z) \cdot G_1(z)$$

wobei  $G_0(z)$  die erste Hälfte und  $G_1(z)$  die zweite Hälfte von  $G(z)$  ist.

- 2 Für alle  $n, m \in \mathbb{N}$  und jedes  $x = x_1 \cdots x_m \in \{0, 1\}^m$  definiere die Funktion

$$f_\ell^G : \{0, 1\}^m \rightarrow \{0, 1\}^n$$

durch

$$f_\ell^G(x) := G_{x_m} \circ \cdots \circ G_{x_1}(\ell).$$

# Pseudo-Random-Funktionen aus PRGs

Wie baut man Pseudo-Zufallsfunktionen aus einem PRG  $G$ ?

- 1 O.B.d.A. hat  $G$  die Streckung  $2n$ : Für  $z \in \{0, 1\}^*$  ist

$$G(z) = G_0(z) \cdot G_1(z)$$

wobei  $G_0(z)$  die erste Hälfte und  $G_1(z)$  die zweite Hälfte von  $G(z)$  ist.

- 2 Für alle  $n, m \in \mathbb{N}$  und jedes  $x = x_1 \cdots x_m \in \{0, 1\}^m$  definiere die Funktion

$$f_\ell^G : \{0, 1\}^m \rightarrow \{0, 1\}^n$$

durch

$$f_\ell^G(x) := G_{x_m} \circ \cdots \circ G_{x_1}(\ell).$$

**Zeige:** Für jeden Pseudo-Random-Generator  $G$  ist

$$\mathcal{F}^G := (f_\ell^G : \ell \in \{0, 1\}^*)$$

eine Familie von Pseudo-Random-Funktionen.

Für  $m = n = |\ell|$ : Sind die Funktionen

$$f_\ell^G(x) := G_{x_m} \circ \dots \circ G_{x_1}(\ell) : \{0, 1\}^n \rightarrow \{0, 1\}^n$$

PRFs?

$T = (T_n : n \in \mathbb{N})$  unterscheide die Funktionen  $f_\ell^G$  aus  $\mathcal{F}^G$  von zufälligen Funktionen.

- Wie antwortet das Orakel für Nachfragen nach Funktionswerten

$$f_\ell^G(x_i) = G_{x_{i,n}} \circ \dots \circ G_{x_{i,1}}(\ell)$$

mit Argumenten  $x_1, \dots, x_M \in \{0, 1\}^n$ ? Baue kleinstmöglichen Baum  $B_\ell$ :

- ▶ Die Wurzel ist mit  $\ell$  markiert. Ist Knoten  $v$  mit  $u$  markiert, dann ist das linke (bzw. rechte) Kind mit  $G_0(u)$  (bzw.  $G_1(u)$ ) markiert.
- ▶  $B_\ell$  hat  $M$  Blätter der Tiefe  $n$  mit der jeweiligen Markierung  $f_\ell^G(x_i)$ .

Für  $m = n = |\ell|$ : Sind die Funktionen

$$f_\ell^G(x) := G_{x_m} \circ \dots \circ G_{x_1}(\ell) : \{0, 1\}^n \rightarrow \{0, 1\}^n$$

PRFs?

$T = (T_n : n \in \mathbb{N})$  unterscheide die Funktionen  $f_\ell^G$  aus  $\mathcal{F}^G$  von zufälligen Funktionen.

- Wie antwortet das Orakel für Nachfragen nach Funktionswerten

$$f_\ell^G(x_i) = G_{x_{i,n}} \circ \dots \circ G_{x_{i,1}}(\ell)$$

mit Argumenten  $x_1, \dots, x_M \in \{0, 1\}^n$ ? Baue kleinstmöglichen Baum  $B_\ell$ :

- ▶ Die Wurzel ist mit  $\ell$  markiert. Ist Knoten  $v$  mit  $u$  markiert, dann ist das linke (bzw. rechte) Kind mit  $G_0(u)$  (bzw.  $G_1(u)$ ) markiert.
- ▶  $B_\ell$  hat  $M$  Blätter der Tiefe  $n$  mit der jeweiligen Markierung  $f_\ell^G(x_i)$ .
- Wie antwortet das Orakel für eine zufällige Funktion  $f$ ?
  - ▶ Baue einen  $B_\ell$  entsprechenden Baum  $B_f$ , markiere Knoten mit Zufallstrings. (Identische Markierungen sind negativ-exponentiell wahrscheinlich.)

Benutze ein **Hybrid-Argument**, das  $B_f$  schrittweise in  $B_\ell$  überführt.

- Baue die Bäume gemäß der zeitlichen Reihenfolge der Anfragen von  $T$ .
- Sei  $\mathcal{O}_i$  das Orakel, das die ersten  $i$  Knoten von  $B_f$  mit rein zufälligen Funktionswerten markiert und danach nur den Generator  $G$  benutzt.
  - ▶  $\mathcal{O}_0$  baut den Baum  $B_\ell$ .
  - ▶ Im letzten Schritt  $N$  der Baum-Konstruktion baut  $\mathcal{O}_N$  den Baum  $B_f$ .

Benutze ein **Hybrid-Argument**, das  $B_f$  schrittweise in  $B_\ell$  überführt.

- Baue die Bäume gemäß der zeitlichen Reihenfolge der Anfragen von  $T$ .
- Sei  $\mathcal{O}_i$  das Orakel, das die ersten  $i$  Knoten von  $B_f$  mit rein zufälligen Funktionswerten markiert und danach nur den Generator  $G$  benutzt.
  - ▶  $\mathcal{O}_0$  baut den Baum  $B_\ell$ .
  - ▶ Im letzten Schritt  $N$  der Baum-Konstruktion baut  $\mathcal{O}_N$  den Baum  $B_f$ .
- Sei  $p_i := \text{pr}[T_n^{\mathcal{O}_i}(1^n) = 1]$  die W-keit, dass  $T$  mit Orakel  $\mathcal{O}_i$  akzeptiert.
  - ▶ Nach Annahme ist  $|p_N - p_0| \geq \frac{1}{\text{poly}(n)}$  und
  - ▶  $\mathbb{E}_{i \in \{1, \dots, N\}} [|p_i - p_{i-1}|] \geq \frac{1}{\text{poly}(n)}$  gilt für die erwartete Differenz.

Ein Test, der  $G$  überführt:



Benutze ein **Hybrid-Argument**, das  $B_f$  schrittweise in  $B_\ell$  überführt.

- Baue die Bäume gemäß der zeitlichen Reihenfolge der Anfragen von  $T$ .
- Sei  $\mathcal{O}_i$  das Orakel, das die ersten  $i$  Knoten von  $B_f$  mit rein zufälligen Funktionswerten markiert und danach nur den Generator  $G$  benutzt.
  - ▶  $\mathcal{O}_0$  baut den Baum  $B_\ell$ .
  - ▶ Im letzten Schritt  $N$  der Baum-Konstruktion baut  $\mathcal{O}_N$  den Baum  $B_f$ .
- Sei  $p_i := \text{pr}[T_n^{\mathcal{O}_i}(1^n) = 1]$  die W-keit, dass  $T$  mit Orakel  $\mathcal{O}_i$  akzeptiert.
  - ▶ Nach Annahme ist  $|p_N - p_0| \geq \frac{1}{\text{poly}(n)}$  und
  - ▶  $\mathbb{E}_{i \in \{1, \dots, N\}}[|p_i - p_{i-1}|] \geq \frac{1}{\text{poly}(n)}$  gilt für die erwartete Differenz.

Ein Test, der  $G$  überführt: Wähle  $i \leq N$  zufällig und wende Test  $T_n$  mit Orakel  $\mathcal{O}_{i-1}$  an.

*Benutze für die ersten  $i - 1$  Knoten zufällige und für die letzten  $N - i$  Knoten durch  $G$  bestimmte Markierungen. Der **ite Knoten** – einmal zufällig, einmal mit  $G$  markiert – muss den Unterschied machen.*

- Pseudo-Random-Generatoren, ✓
  - ▶ Private-Key-Kryptographie, ✓
  - ▶ Derandomisierung, ✓
  - ▶ Pseudo-Random-Funktionen, ✓
    - ★ Kollisionsresistente Hashfunktionen, ✓
    - ★ Nachrichten-Authentifizierung,
- Bit-Commitment, ✓

# Kollisionsresistente Hashfunktionen

$I \subseteq \{0, 1\}^*$  ist eine Menge von Indizes. Für jedes  $i \in I$  gelte  $n_i < N_i \in \mathbb{N}$ .

Eine Familie  $(H_i \mid i \in I)$  von Funktionen  $H_i : \{0, 1\}^{N_i} \rightarrow \{0, 1\}^{n_i}$  hat genau dann die Eigenschaft der **Kollisionsresistenz** (engl.: collision resistance), wenn

für jede Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$

$$\text{pr}_{i \in I \cap \{0, 1\}^n} [T_n(i) = (x, y), \text{ wobei } x \neq y \text{ und } H_i(x) = H_i(y)]$$

als Funktion von  $n$  vernachlässigbar ist. (Kollisionen lassen sich effizient nur mit vernachlässigbarer Wahrscheinlichkeit berechnen.)

$I \subseteq \{0, 1\}^*$  ist eine Menge von Indizes. Für jedes  $i \in I$  gelte  $n_i < N_i \in \mathbb{N}$ .

Eine Familie  $(H_i \mid i \in I)$  von Funktionen  $H_i : \{0, 1\}^{N_i} \rightarrow \{0, 1\}^{n_i}$  hat genau dann die Eigenschaft der **Kollisionsresistenz** (engl.: collision resistance), wenn

für jede Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen  $T_n$  der Größe  $\text{poly}(n)$

$$\text{pr}_{i \in I} \text{pr}_{x, y \in \{0, 1\}^{n_i}} [T_n(i) = (x, y), \text{ wobei } x \neq y \text{ und } H_i(x) = H_i(y)]$$

als Funktion von  $n$  vernachlässigbar ist. (Kollisionen lassen sich effizient nur mit vernachlässigbarer Wahrscheinlichkeit berechnen.)

**Integritätsprüfung:** Ein Software-Paket  $P$  wird über einen öffentlichen Kanal verteilt.

Für Paket  $Q$  und den *verlässlich einsehbaren* „Digest“  $h(P)$  wird die Echtheit der Software durch die Gleichheit  $h(P) = h(Q)$  bestätigt.

Wenn die Hashfunktionen  $H_i$  Strings der Länge  $N_i$  auf Strings der Länge  $n_i$  mit  $n_i < N_i$  abbilden, dann kann  $H_i$  durch eine Pseudo-Random-Funktion

$$f_\ell : \{0, 1\}^m \rightarrow \{0, 1\}^n$$

mit  $m = N_i$  und  $n = n_i = |\ell|$  implementiert werden:

Wenn die Hashfunktionen  $H_i$  Strings der Länge  $N_i$  auf Strings der Länge  $n_i$  mit  $n_i < N_i$  abbilden, dann kann  $H_i$  durch eine Pseudo-Random-Funktion

$$f_\ell : \{0, 1\}^m \rightarrow \{0, 1\}^n$$

mit  $m = N_i$  und  $n = n_i = |\ell|$  implementiert werden:

*Kollisionen können für wirkliche Zufallsfunktionen nicht vorausgesagt werden und deshalb auch nicht für Pseudo-Random-Funktionen.*

In Anwendungen hat die Familie der SHA-2 Hashfunktionen bisher alle Angriffe überstanden. Der Nachfolger, die SHA-3 Hashfunktionen, sind nicht unumstritten.

- Pseudo-Random-Generatoren, ✓
  - ▶ Private-Key-Kryptographie, ✓
  - ▶ Derandomisierung, ✓
  - ▶ Pseudo-Random-Funktionen, ✓
    - ★ Kollisionsresistente Hashfunktionen, ✓
    - ★ Nachrichten-Authentifizierung, ✓
- Bit-Commitment, ✓



# Nachrichten-Authentifizierung mit privatem Schlüssel

Bob möchte eine Nachricht  $x$  an Alice über einen öffentlichen Kanal schicken.

Die Nachrichten sind so zu verschlüsseln, dass Alice verifizieren kann, dass die Nachricht – so wie erhalten – von Bob geschickt wurde.

# Nachrichten-Authentifizierung mit privatem Schlüssel

Bob möchte eine Nachricht  $x$  an Alice über einen öffentlichen Kanal schicken.

Die Nachrichten sind so zu verschlüsseln, dass Alice verifizieren kann, dass die Nachricht – so wie erhalten – von Bob geschickt wurde.

- 1 Alice und Bob einigen sich auf einen geheimen Schlüssel  $g \in \{0, 1\}^n$  und eine Familie  $\mathcal{F} = (f_\ell \mid \ell \in \{0, 1\}^*)$  von PRFs  $f_\ell : \{0, 1\}^{|\ell|} \rightarrow \{0, 1\}^{|\ell|}$ .
- 2 An Stelle der Nachricht  $x \in \{0, 1\}^n$  schickt Bob das Paar  $(x, f_g(x))$  an Alice.
- 3 Alice akzeptiert alle Nachrichten  $(x', y')$  mit  $f_g(x') = y'$ .
  - ▶ Alice weiß, dass nur Bob den geheimen Schlüssel  $g$  kennt.

# Nachrichten-Authentifizierung mit privatem Schlüssel

Bob möchte eine Nachricht  $x$  an Alice über einen öffentlichen Kanal schicken.

Die Nachrichten sind so zu verschlüsseln, dass Alice verifizieren kann, dass die Nachricht – so wie erhalten – von Bob geschickt wurde.

- 1 Alice und Bob einigen sich auf einen geheimen Schlüssel  $g \in \{0, 1\}^n$  und eine Familie  $\mathcal{F} = (f_\ell \mid \ell \in \{0, 1\}^*)$  von PRFs  $f_\ell : \{0, 1\}^{|\ell|} \rightarrow \{0, 1\}^{|\ell|}$ .
- 2 An Stelle der Nachricht  $x \in \{0, 1\}^n$  schickt Bob das Paar  $(x, f_g(x))$  an Alice.
- 3 Alice akzeptiert alle Nachrichten  $(x', y')$  mit  $f_g(x') = y'$ .
  - ▶ Alice weiß, dass nur Bob den geheimen Schlüssel  $g$  kennt.

Verändert ein Man-In-The-Middle Nachrichten, dann wird ein statistischer Test definiert!

Für wirklich zufällige Funktionen ist die Wahrscheinlichkeit einer gelungenen Täuschung vernachlässigbar  $\implies$  Authentifizierung mit PRFs gelingt.

# Trapdoor-Funktionen und Public-Key-Kryptographie

# Trapdoor-Funktionen

Für eine Menge  $I \subseteq \{0, 1\}^*$  von Indizes, eff. randomisierten Algorithmen  $KeyGen, Enc, Dec$ , endliche Mengen  $D_i$  von Klartexten und Geheimnissen  $g_i \in \{0, 1\}^*$  für jedes  $i \in I$  heißt

$$\mathcal{F} = ((KeyGen, Enc, Dec, \{(i, g_i, D_i) : i \in I\}))$$

eine Familie von **Trapdoor-Funktionen**, falls:

- (a) Sei  $J_n := \{(i, g_i) : i \in I \cap \{0, 1\}^n\}$ . Paare  $(i, g_i)$  lassen sich effizient „sampeln“, d.h. für den Sicherheitsparameter  $n \in \mathbb{N}$  gilt  $KeyGen(1^n) = (i, g_i) \in J_n$ .
- (b) Für jeden Index  $i \in I$  und jeden Klartext  $x \in D_i$  ist  $Enc(i, x)$  ein Geheimtext. (Ein Klartext kann mehrere Geheimtexte haben, da  $Enc$  randomisiert arbeitet.)
- (c)  $Dec$  entschlüsselt einen Geheimtext mit Hilfe des Geheimnisses  $g_i$ , d.h. für jeden Index  $i \in I$  und jeden Klartext  $x \in D_i$  gilt  $Dec(i, g_i, Enc(x, i)) = x$ .
- (d) Für jede Familie  $T = (T_n : n \in \mathbb{N})$  von Schaltkreisen polyn. Größe ist die W-keit

$$\text{pr}_{i \in I \cap \{0, 1\}^n, x \in D_i} [T_n(i, f_i(x)) = x' \text{ und } f_i(x) = f_i(x')] ]$$

einer erfolgreichen Invertierung vernachlässigbar (als Funktion des Sicherheitsparameters): Ziehe  $i \in I \cap \{0, 1\}^n$  zufällig mit  $KeyGen$  und  $x$  gleichverteilt aus  $D_i$ .

# Public-Key-Kryptographie

Sei  $\mathcal{F} = (\text{KeyGen}, \text{Enc}, \text{Dec}, \{(i, g_i, D_i) : i \in I\})$  eine Familie von Trapdoor-Funktionen.

1. Alice würfelt ein Paar

$$(i, g_i)$$

von **öffentlichem** und **privatem** Schlüssel mit Algorithmus  $\text{KeyGen}$  aus und gibt den öffentlichen Schlüssel  $i$  bekannt.

2. Bob berechnet den Geheimtext

$$y := \text{Enc}(i, x)$$

für den Klartext  $x \in D_i$ .

3. Alice entschlüsselt mit Algorithmus  $\text{Dec}$ , d.h. es ist

$$\text{Dec}(i, g_i, \text{Enc}(x, i)) = x.$$

- Die **Rabin-Funktion**

$$Enc(x) := x^2 \pmod n$$

für ein Produkt  $n = p \cdot q$  von Primzahlen  $p, q$  ist einfach zu invertieren, wenn  $p, q$  bekannt sind. Ansonsten ist die Bestimmung einer Wurzel modulo  $n$  genauso schwierig wie die Bestimmung der Primfaktoren von  $n$ . Hier ist  $i = n$  und  $g_i = (p, q)$  ist das Geheimnis.

- Die **Rabin-Funktion**

$$Enc(x) := x^2 \pmod n$$

für ein Produkt  $n = p \cdot q$  von Primzahlen  $p, q$  ist einfach zu invertieren, wenn  $p, q$  bekannt sind. Ansonsten ist die Bestimmung einer Wurzel modulo  $n$  genauso schwierig wie die Bestimmung der Primfaktoren von  $n$ .

Hier ist  $i = n$  und  $g_i = (p, q)$  ist das Geheimnis.

- Die **RSA-Funktion**

$$Enc(x) := x^e \pmod n$$

für ein Produkt  $n = p \cdot q$  von Primzahlen  $p, q$  ist einfach zu invertieren, wenn  $d = e^{-1}$  modulo  $\phi(n)$  bekannt ist.  $\phi$  ist die Eulersche  $\phi$ -Funktion.

Diesmal ist  $i = (n, e)$  und  $g_i = d$  ist das Geheimnis.



## Das El-Gamal-System:

Sei  $p$  eine Sophie-Germain-Primzahl, d.h.  $p$  und  $q = (p - 1)/2$  sind prim.

- 1 Die Untergruppe  $R = \{a^2 \bmod p : 1 \leq a \leq p - 1\}$  aller quadratischen Reste modulo  $p$  hat also **Primzahl-Ordnung** und ist **zyklisch**. Wähle  $g \in R$ .
- 2 Alice wählt eine Restklasse  $a \in R$  als **privaten Schlüssel** und  $(g, h, p)$  mit der Restklasse  $h := g^a$  als **öffentlichen Schlüssel**.

Das **EI-Gamal-System**:

Sei  $p$  eine Sophie-Germain-Primzahl, d.h.  $p$  und  $q = (p - 1)/2$  sind prim.

- 1 Die Untergruppe  $R = \{a^2 \bmod p : 1 \leq a \leq p - 1\}$  aller quadratischen Reste modulo  $p$  hat also **Primzahl-Ordnung** und ist **zyklisch**. Wähle  $g \in R$ .
- 2 Alice wählt eine Restklasse  $a \in R$  als **privaten Schlüssel** und  $(g, h, p)$  mit der Restklasse  $h := g^a$  als **öffentlichen Schlüssel**.
- 3 Bob wählt eine **zufällige** Restklasse  $b \in R$  und sendet

$$Enc(x) := (g^b, h^b \cdot x)$$

als Verschlüsselung der ursprünglichen Nachricht  $x$ .

Das **EI-Gamal-System**:

Sei  $p$  eine Sophie-Germain-Primzahl, d.h.  $p$  und  $q = (p - 1)/2$  sind prim.

- 1 Die Untergruppe  $R = \{a^2 \bmod p : 1 \leq a \leq p - 1\}$  aller quadratischen Reste modulo  $p$  hat also **Primzahl-Ordnung** und ist **zyklisch**. Wähle  $g \in R$ .
- 2 Alice wählt eine Restklasse  $a \in R$  als **privaten Schlüssel** und  $(g, h, p)$  mit der Restklasse  $h := g^a$  als **öffentlichen Schlüssel**.
- 3 Bob wählt eine **zufällige** Restklasse  $b \in R$  und sendet

$$Enc(x) := (g^b, h^b \cdot x)$$

als Verschlüsselung der ursprünglichen Nachricht  $x$ .

- 4 Entschlüsselung: Alice berechnet in  $R$

$$(g^b)^{-a} \cdot h^b \cdot x$$

Das **EI-Gamal-System**:

Sei  $p$  eine Sophie-Germain-Primzahl, d.h.  $p$  und  $q = (p - 1)/2$  sind prim.

- 1 Die Untergruppe  $R = \{a^2 \bmod p : 1 \leq a \leq p - 1\}$  aller quadratischen Reste modulo  $p$  hat also **Primzahl-Ordnung** und ist **zyklisch**. Wähle  $g \in R$ .
- 2 Alice wählt eine Restklasse  $a \in R$  als **privaten Schlüssel** und  $(g, h, p)$  mit der Restklasse  $h := g^a$  als **öffentlichen Schlüssel**.
- 3 Bob wählt eine **zufällige** Restklasse  $b \in R$  und sendet

$$Enc(x) := (g^b, h^b \cdot x)$$

als Verschlüsselung der ursprünglichen Nachricht  $x$ .

- 4 Entschlüsselung: Alice berechnet in  $R$

$$(g^b)^{-a} \cdot h^b \cdot x = g^{-ab+ab} \cdot x$$

Das **EI-Gamal-System**:

Sei  $p$  eine Sophie-Germain-Primzahl, d.h.  $p$  und  $q = (p - 1)/2$  sind prim.

- 1 Die Untergruppe  $R = \{a^2 \bmod p : 1 \leq a \leq p - 1\}$  aller quadratischen Reste modulo  $p$  hat also **Primzahl-Ordnung** und ist **zyklisch**. Wähle  $g \in R$ .
- 2 Alice wählt eine Restklasse  $a \in R$  als **privaten Schlüssel** und  $(g, h, p)$  mit der Restklasse  $h := g^a$  als **öffentlichen Schlüssel**.
- 3 Bob wählt eine **zufällige** Restklasse  $b \in R$  und sendet

$$Enc(x) := (g^b, h^b \cdot x)$$

als Verschlüsselung der ursprünglichen Nachricht  $x$ .

- 4 Entschlüsselung: Alice berechnet in  $R$

$$(g^b)^{-a} \cdot h^b \cdot x = g^{-ab+ab} \cdot x = x.$$

- 5 Es ist  $i = (g, h, p)$  und  $g_i = a$ .

# Semantische Sicherheit

**IND-CPA** (Ununterscheidbarkeit unter Angriff mit „frei wählbarem Klartext“, engl: indistinguishability under chosen-plaintext-attack).

Die Trapdoor-Funktion  $\mathcal{F}$  muss das folgende Spiel gegen jeden Angreifer  $\mathcal{A}$  mit einer  $W$ -keit gewinnen, die nur vernachlässigbar geringer als  $1/2$  ist:

1.  $\mathcal{F}$  wählt zufällige öffentliche und private Schlüssel. Der öffentliche Schlüssel  $i$  wird dem Angreifer  $\mathcal{A}$  mitgeteilt.
2.  $\mathcal{A}$  wählt zwei Nachrichten  $m_0$  und  $m_1$  gleicher Länge aus.
3.  $\mathcal{F}$  wendet die randomisierte Verschlüsselung  $Enc$  auf  $m_0$  sowie  $m_1$  an und teilt  $\mathcal{A}$  die Verschlüsselung  $Enc(i, m_b)$  (für zufälliges  $b \in \{0, 1\}$ ) mit.
4.  $\mathcal{A}$  kann um die Verschlüsselung weiterer Nachrichten bitten. Nach Erhalt aller Verschlüsselungen muss  $\mathcal{A}$  von  $Enc(i, m_b)$  auf Bit  $b$  zurückschließen.

Das El Gamal-System ist IND-CPA sicher, wenn die

**Decisional-Diffie-Hellman-Vermutung** gilt: Effiziente Algorithmen können

$$g^x, g^y, g^{x \cdot y} \text{ von } g^x, g^y, g^z$$

für zufällige  $x, y, z \in \mathbb{Z}_p$  nur mit vernachlässigbarem Vorteil unterscheiden.

Auch RSA kann zu einem IND-CPA sicheren Verfahren umgebaut werden.



Die Sicherheitsanforderung der aktiven Sicherheit ist stärker:

**IND-CCA1** bzw. **IND-CCA2** (Ununterscheidbarkeit unter Angriff mit „frei wählbarem Chiffretext“, engl: indistinguishability under chosen-ciphertext-attack)

Statt „nur“ Verschlüsselungen anfordern zu können, erhält  $\mathcal{A}$  Zugriff auf ein Entschlüsselungsorakel. (Natürlich darf das Orakel nicht auf die Verschlüsselungen von  $m_0, m_1$  angesetzt werden.)

# Digitale Unterschriften

## Anforderungen an eine digitale Unterschrift:

(a) **Integrität** (engl. Integrity):

Der Empfänger einer Nachricht muss sicher sein, dass die Nachricht nicht zwischenzeitlich verändert wurde.

(b) **Authentifizierung** (engl. Authentication)

Der Empfänger einer Nachricht muss sicher sein, dass die Nachricht tatsächlich vom behaupteten Sender verschickt wurde.

- ▶ PRFs unterstützen Authentifizierung: Der Empfänger akzeptiert die Nachricht  $(x, f_k(x))$  als „vom Absender geschickt“ genau dann, wenn  $f_k(x) = y$ .
- ▶ Aber der geheime Schlüssel  $k$  muss abgesprochen sein.

(c) **Nichtabstreitbarkeit** (engl. Non-Repudiation):

Eine dritte Partei muss die Urheberschaft für die gesandte Nachricht nur mit öffentlicher Information nachvollziehen können.

Benutze private sowie öffentliche Schlüssel  $g_{\text{pri}}$ ,  $g_{\text{pub}}$  und verwende Trapdoor-Funktionen, um alle drei Eigenschaften zu erfüllen.

Insbesondere baue effiziente randomisierte Algorithmen Keygen, Sign und Verify:

- (1) Ein Algorithmus **Keygen**, der für den Sicherheitsparameter  $n$  ein Paar  $(g_{\text{pri}}, g_{\text{pub}})$  von Schlüsseln bestimmt,
- (2) ein Algorithmus **Sign**, der die Unterschrift  $\text{Sign}(x, g_{\text{pri}})$  bestimmt und
- (3) ein Algorithmus **Verify**, der die signierte Nachricht  $(x, \text{Sign}(x, g_{\text{pri}}))$  mit Hilfe des öffentlichen Schlüssels  $g_{\text{pub}}$  überprüft. Verify muss
  - ▶ eine legale Unterschrift  $\text{Sign}(x, g_{\text{pri}})$  mit W-keit 1 akzeptieren und
  - ▶ eine durch einen Schaltkreis  $T_n$  (der Größe  $\text{poly}(n)$ ) gefälschte Unterschrift mit W-keit  $1 - \frac{1}{\text{poly}(n)}$  entdecken.

Ein auf dem RSA-Kryptosystem basierendes Unterschriften-Schema wählt

- $g_{\text{pub}} := (N, e)$  für das Produkt  $N = p \cdot q$  zweier Primzahlen und
- den privaten Schlüssel  $g_{\text{pri}} := (N, d)$ , wobei  $e \cdot d \equiv 1 \pmod{\phi(n)}$  gelte.

Die Algorithmen Keygen, Sign und Verify:

- (\*) **Keygen**: Würfle zwei große Primzahlen  $p, q$  aus und bestimme

$$d = e^{-1} \pmod{\phi(N)}.$$

- (\*) Als Unterschrift wähle

$$\text{Sign}(x, g_{\text{pri}}) := x^d \pmod{N}.$$

- (\*) **Verify**( $x$ , Unterschrift,  $g_{\text{pub}}$ ) akzeptiert genau dann, wenn

$$\text{Unterschrift}^e = x \pmod{N}.$$

- ▶ Jeder kann Nicht-Abstreitbarkeit nachweisen, denn  $\text{Sign}(x, g_{\text{pri}})$  muss den nur dem Sender bekannten privaten Schlüssel  $d$  benutzen.

# Gitter-Kryptographie

- (a) Ein  $n$ -dimensionales Gitter ist eine **diskrete, additive Gruppe**  $\mathcal{G} \subseteq \mathbb{R}^n$ : Es gibt  $\varepsilon > 0$ , so dass der Ball mit Radius  $\varepsilon$  um jeden Vektor  $x \in \mathcal{G}$  nur  $x$  enthält.
- (b) Sei  $B = [b_1, \dots, b_k]$  die  $n \times k$ -Matrix mit linear unabhängigen Spalten  $b_1, \dots, b_k$ . Das von  $B$  aufgespannte Gitter ist

$$\mathcal{G}(B) := B \cdot \mathbb{Z}^k = \left\{ \sum_{i=1}^k \alpha_i \cdot b_i : \alpha_1, \dots, \alpha_k \in \mathbb{Z} \right\}.$$

Das Parallelotop (bzw. die Grundmasche)  $P(B)$  wird definiert durch

$$P(B) := \left\{ \sum_{i=1}^k \alpha_i \cdot b_i : 0 \leq \alpha_1, \dots, \alpha_n < 1 \right\}.$$

# Eigenschaften eines Gitters

- (a) Zu jedem Gitter  $\mathcal{G} \subseteq \mathbb{R}^n$  gibt es linear unabhängige Vektoren  $b_1, \dots, b_k \in \mathcal{G}$  mit  $\mathcal{G} = \mathcal{G}(B)$ .



# Eigenschaften eines Gitters

- (a) Zu jedem Gitter  $\mathcal{G} \subseteq \mathbb{R}^n$  gibt es linear unabhängige Vektoren  $b_1, \dots, b_k \in \mathcal{G}$  mit  $\mathcal{G} = \mathcal{G}(B)$ .
- (b) Die  $n \times n$ -Matrix  $B$  habe Rang  $n$ .
  1.  $\mathcal{G}(B) = \mathcal{G}(C) \iff C = B \cdot U$  gilt für eine **unimodulare**<sup>a</sup> Matrix  $A \implies$   
Verschiedene Basen desselben Gitters besitzen im Absolutbetrag dieselbe Determinante.
  2. Das Volumen des Parallelotops  $P(B)$  (bzw. die Diskriminante) stimmt mit  $|\det(B)|$  überein  $\implies$   
alle Parallelotope für dasselbe Gitter haben dasselbe Volumen: Die Diskriminante ist eine Invariante

# Eigenschaften eines Gitters

- (a) Zu jedem Gitter  $\mathcal{G} \subseteq \mathbb{R}^n$  gibt es linear unabhängige Vektoren  $b_1, \dots, b_k \in \mathcal{G}$  mit  $\mathcal{G} = \mathcal{G}(B)$ .
- (b) Die  $n \times n$ -Matrix  $B$  habe Rang  $n$ .
- $\mathcal{G}(B) = \mathcal{G}(C) \iff C = B \cdot U$  gilt für eine **unimodulare**<sup>a</sup> Matrix  $U \implies$   
Verschiedene Basen desselben Gitters besitzen im Absolutbetrag dieselbe Determinante.
  - Das Volumen des Parallelotops  $P(B)$  (bzw. die Diskriminante) stimmt mit  $|\det(B)|$  überein  $\implies$   
alle Parallelotope für dasselbe Gitter haben dasselbe Volumen: Die Diskriminante ist eine Invariante
- (c) Seien  $\mathcal{G}(B), \mathcal{G}(C)$  vorgegeben. Dann sind die Probleme
- „gehört der Vektor  $v$  zum Gitter  $\mathcal{G}(B)$ ?“
- sowie
- „stimmen  $\mathcal{G}(B)$  und  $\mathcal{G}(C)$  überein?“
- effizient lösbar.

---

<sup>a</sup>Eine  $n \times n$ -Matrix  $U$  heißt unimodular, wenn alle Einträge von  $U$  ganzzahlig sind und wenn die Determinante von  $U$  die Werte  $-1$  oder  $1$  annimmt.

# Shortest-Vektor-Probleme

Es gelte  $\mathcal{G} := \mathcal{G}(A)$  für die nicht-singuläre  $n \times n$ -Matrix  $A$ .

- (a) Für  $1 \leq k \leq n$  ist  $\lambda_k(\mathcal{G}) := \lambda$  falls  $\lambda$  kleinstmöglich ist mit der Eigenschaft, dass es  $k$  linear unabhängige Vektoren der Länge höchstens  $\lambda$  in  $\mathcal{G}$  gibt.

# Shortest-Vektor-Probleme

Es gelte  $\mathcal{G} := \mathcal{G}(A)$  für die nicht-singuläre  $n \times n$ -Matrix  $A$ .

(a) Für  $1 \leq k \leq n$  ist  $\lambda_k(\mathcal{G}) := \lambda$  falls  $\lambda$  kleinstmöglich ist mit der Eigenschaft, dass es  $k$  linear unabhängige Vektoren der Länge höchstens  $\lambda$  in  $\mathcal{G}$  gibt.

(b) Sei  $\gamma : \mathbb{N} \rightarrow \mathbb{N}$  eine Funktion.

- ▶ Das Shortest-Vector-Problem **SVP**: Bestimme einen kürzesten, vom Nullvektor verschiedenen Vektor in  $\mathcal{G}$ .
- ▶ In **SVP** $_\gamma$  bestimme einen Vektor  $w \in \mathcal{G}$  mit  $\|w\| \leq \gamma(n) \cdot \lambda_1(\mathcal{G})$ .
- ▶ Im Promise-Problem **Gap-SVP** $_\gamma$  wird gefragt, ob  $\lambda_1(\mathcal{G}) \leq 1$  oder  $\lambda_1(\mathcal{G}) > \gamma(n)$  gilt.
- ▶ Im Shortest-Independent-Vectors-Problem **SIVP** $_\gamma$  werden  $n$  linear unabhängige Vektoren  $w_1, \dots, w_n \in \mathcal{G}$  gesucht mit

$$\max_{1 \leq i \leq n} \|w_i\| \leq \gamma(n) \cdot \lambda_n(\mathcal{G}).$$

- ▶ Im Closest-Vector-Problem **CVP** für einen Vektor  $w$  wird ein Vektor  $v \in \mathcal{G}$  mit dem geringsten Abstand zu  $w$  gesucht.

Analog zu SVP werden die Problemvarianten **CVP** $_\gamma$  und **Gap-CVP** $_\gamma$  eingeführt.

# Shortest-Vektor-Probleme, der Wissensstand

- SVP ist ein NP-hartes Problem unter randomisierten Reduktionen.
- Wenn NP keine Teilmenge von  $\text{RTIME}(2^{\text{poly}(\log n)})$  ist, dann besitzt  $\text{SVP}_\gamma$  für kein  $\gamma \leq 2^{\log^{1-\varepsilon} n}$  (mit  $\varepsilon > 0$ ) effiziente Algorithmen.
- Für  $\gamma \geq \sqrt{n}$  gilt
  - ▶  $\text{SVP}_\gamma, \text{CVP}_\gamma \in \text{NP} \cap \text{coNP}$  und ein NP-Härte Resultat ist nicht zu erwarten.
  - ▶ Der **Graubereich**, wenn  $\gamma$  polynomiell in der Gitterdimension ist:
    - ★ die besten deterministischen Algorithmen benötigen bisher super-exponentielle Zeit  $2^{\Theta(n \cdot \log_2 n)}$  bzw. exponentielle Zeit *und* exponentiellen Speicherplatz.
    - ★ dieser Graubereich stellt sich für kryptographische Anwendungen als überaus wichtig heraus.
  - ▶ Analoge Ergebnisse für  $\text{SIVP}_\gamma$ .

Aber wir sollten doch an der Average-Case-Komplexität interessiert sein!

# Short-Integer-Solution

# Short-Integer-Solution

Die natürlichen Zahlen  $n$ ,  $m$ ,  $q$  und  $\beta$  seien vorgegeben.

(a) Für eine  $n \times m$ -Matrix  $A$  mit Einträgen aus  $\mathbb{Z}_q$  definiere das Gitter

$$\mathcal{G}^\perp(A) := \{z \in \mathbb{Z}^m : A \cdot z \equiv 0 \pmod{q}\}.$$

(b) Eine Instanz von

**SIS** $_{n,q,\beta,m}$

wird durch eine  $n \times m$ -Matrix  $A$  mit Einträgen aus  $\mathbb{Z}_q$  beschrieben. Bestimme  $z \in \mathcal{G}^\perp(A)$  mit  $z \neq 0$  und

$$\|z\| \leq \beta.$$

Bestimme die **Average-Case-Komplexität** von  $\text{SIS}_{n,q,\beta,m}$ , also für die Suche nach einem kurzen Vektor in  $\mathcal{G}^\perp(A)$  mit einer zufällig ausgewürfelten Matrix  $A$  über  $\mathbb{Z}_q$ .

# Wann ist $\text{SIS}_{n,q,\beta,m}$ lösbar?

Sei  $A$  eine  $n \times m$ -Matrix und es gelte

$$m > \lceil n \cdot \log q \rceil \text{ und } \beta \geq \sqrt{m}.$$

Dann gibt es einen Vektor  $z \in \{-1, 0, 1\}^m \cap \mathcal{G}^\perp(A)$  mit  $\|z\| < \beta$ .

Warum? Sei  $M := \lceil n \cdot \log q \rceil$ . Dann ist  $m \geq M$ .

- Es gibt  $2^M > q^n$  Vektoren  $z \in \{0, 1\}^M \times \{0\}^{m-M}$ .



# Wann ist $\text{SIS}_{n,q,\beta,m}$ lösbar?

Sei  $A$  eine  $n \times m$ -Matrix und es gelte

$$m > \lceil n \cdot \log q \rceil \text{ und } \beta \geq \sqrt{m}.$$

Dann gibt es einen Vektor  $z \in \{-1, 0, 1\}^m \cap \mathcal{G}^\perp(A)$  mit  $\|z\| < \beta$ .

Warum? Sei  $M := \lceil n \cdot \log q \rceil$ . Dann ist  $m \geq M$ .

- Es gibt  $2^M > q^n$  Vektoren  $z \in \{0, 1\}^M \times \{0\}^{m-M}$ .
- Also gibt es zwei Vektoren  $z_1 \neq z_2 \in \{0, 1\}^M \times \{0\}^{m-M}$  mit  $A \cdot z_1 = A \cdot z_2 \pmod{q}$ .

# Wann ist $\text{SIS}_{n,q,\beta,m}$ lösbar?

Sei  $A$  eine  $n \times m$ -Matrix und es gelte

$$m > \lceil n \cdot \log q \rceil \text{ und } \beta \geq \sqrt{m}.$$

Dann gibt es einen Vektor  $z \in \{-1, 0, 1\}^m \cap \mathcal{G}^\perp(A)$  mit  $\|z\| < \beta$ .

Warum? Sei  $M := \lceil n \cdot \log q \rceil$ . Dann ist  $m \geq M$ .

- Es gibt  $2^M > q^n$  Vektoren  $z \in \{0, 1\}^M \times \{0\}^{m-M}$ .
- Also gibt es zwei Vektoren  $z_1 \neq z_2 \in \{0, 1\}^M \times \{0\}^{m-M}$  mit  $A \cdot z_1 = A \cdot z_2 \pmod{q}$ .
- Aber dann gehört  $z := z_1 - z_2$  zum Gitter  $\mathcal{G}^\perp(A)$ . Da  $z \in \{-1, 0, 1\}^m$ , folgt

$$\|z\| \leq \sqrt{M}$$

# Wann ist $\text{SIS}_{n,q,\beta,m}$ lösbar?

Sei  $A$  eine  $n \times m$ -Matrix und es gelte

$$m > \lceil n \cdot \log q \rceil \text{ und } \beta \geq \sqrt{m}.$$

Dann gibt es einen Vektor  $z \in \{-1, 0, 1\}^m \cap \mathcal{G}^\perp(A)$  mit  $\|z\| < \beta$ .

Warum? Sei  $M := \lceil n \cdot \log q \rceil$ . Dann ist  $m \geq M$ .

- Es gibt  $2^M > q^n$  Vektoren  $z \in \{0, 1\}^M \times \{0\}^{m-M}$ .
- Also gibt es zwei Vektoren  $z_1 \neq z_2 \in \{0, 1\}^M \times \{0\}^{m-M}$  mit  $A \cdot z_1 = A \cdot z_2 \pmod{q}$ .
- Aber dann gehört  $z := z_1 - z_2$  zum Gitter  $\mathcal{G}^\perp(A)$ . Da  $z \in \{-1, 0, 1\}^m$ , folgt

$$\|z\| \leq \sqrt{M} < \sqrt{m}$$

# Wann ist $\text{SIS}_{n,q,\beta,m}$ lösbar?

Sei  $A$  eine  $n \times m$ -Matrix und es gelte

$$m > \lceil n \cdot \log q \rceil \text{ und } \beta \geq \sqrt{m}.$$

Dann gibt es einen Vektor  $z \in \{-1, 0, 1\}^m \cap \mathcal{G}^\perp(A)$  mit  $\|z\| < \beta$ .

Warum? Sei  $M := \lceil n \cdot \log q \rceil$ . Dann ist  $m \geq M$ .

- Es gibt  $2^M > q^n$  Vektoren  $z \in \{0, 1\}^M \times \{0\}^{m-M}$ .
- Also gibt es zwei Vektoren  $z_1 \neq z_2 \in \{0, 1\}^M \times \{0\}^{m-M}$  mit  $A \cdot z_1 = A \cdot z_2 \pmod{q}$ .
- Aber dann gehört  $z := z_1 - z_2$  zum Gitter  $\mathcal{G}^\perp(A)$ . Da  $z \in \{-1, 0, 1\}^m$ , folgt

$$\|z\| \leq \sqrt{M} < \sqrt{m} \leq \beta.$$

# SIS<sub>n,q,β,m</sub>: Diskussion der Parameter

- Die Zahl  $n$  ist der Sicherheitsparameter.
- Die Anzahl  $m$  der Spalten von  $A$  wird größer als  $n$  gewählt, aber ist höchstens polynomiell in  $n$ .
- Die Primzahl  $q$  ist in vielen Anwendungen ungefähr quadratisch in  $n$ .
- Schließlich wird  $\beta$  deutlich kleiner als  $q$  gewählt, denn der Vektor  $z = (q, 0, \dots, 0)$  gehört zu  $\mathcal{G}^\perp(A)$ , falls  $\beta \geq q$  gilt.

Man wählt in Anwendungen zum Beispiel die Parameter

$$q(n) = \Omega(n^2), \beta(n) \geq \sqrt{m(n)} \text{ und } m(n) \approx n \log_2 q(n).$$

# Eine Worst-Case nach Average-Case Reduktion

# Der Hauptsatz

Die folgende Annahme gelte:

$SIVP_\gamma$  mit  $\gamma = \mathcal{O}(\sqrt{n})$  ist durch effiziente Quantenalgorithmen im **Worst-Case nicht** lösbar.

Dann folgt:

- (a) Es gelte  $m(n) = \Omega(n \cdot \log n)$ ,  $\beta = \sqrt{m(n)}$  und  $q(n) \geq \Theta(n^2 \log n)$  bei jeweils polynomieller Beschränkung  $\implies$

$SIS_{n,q,\beta,m}$  ist im **Average-Case** nur mit vernachlässigbarer  $W$ -keit durch effiziente Quantenalgorithmen lösbar.

- (b) Des weiteren ist

$$(f_A : \{0, 1\}^{m(n)} \rightarrow \mathbb{Z}_{q(n)}^n \mid A \in \mathbb{Z}_{q(n)}^{n \times m(n)}) \text{ mit } f_A(z) := A \cdot z \pmod{q}$$

eine Familie von Hashfunktionen, die selbst gegen effiziente Quantenalgorithmen kollisionsresistent sind.

# Kollisionsresistente Hashfunktionen

Angenommen, die Familie

$$(f_A \mid A \in \mathbb{Z}_{q(n)}^{n \times m(n)})_{n, q(n), \beta(n), m(n)} \text{ mit } f_A(z) = A \cdot z \pmod{q} \text{ (für } z \in \{0, 1\}^{m(n)})$$

ist nicht kollisionsresistent.

- Dann gibt es einen effizienten Algorithmus, der für eine Matrix  $A$  mit  $W$ -keit  $\geq \frac{1}{\text{poly}(n)}$  ein Paar  $(x, y)$  mit  $x \neq y$  und  $f_A(x) = f_A(y)$  bestimmt.



# Kollisionsresistente Hashfunktionen

Angenommen, die Familie

$$(f_A \mid A \in \mathbb{Z}_{q(n)}^{n \times m(n)})_{n, q(n), \beta(n), m(n)} \text{ mit } f_A(z) = A \cdot z \pmod{q} \text{ (für } z \in \{0, 1\}^{m(n)})$$

ist nicht kollisionsresistent.

- Dann gibt es einen effizienten Algorithmus, der für eine Matrix  $A$  mit  $W$ -keit  $\geq \frac{1}{\text{poly}(n)}$  ein Paar  $(x, y)$  mit  $x \neq y$  und  $f_A(x) = f_A(y)$  bestimmt.
- Dann ist aber  $z := x - y \in \{-1, 0, 1\}^{m(n)}$  eine Lösung von  $\text{SIS}_{n, q(n), \beta(n), m(n)}$ , denn  $\beta(n) = \sqrt{m}$  wird gefordert.

# Kollisionsresistente Hashfunktionen

Angenommen, die Familie

$$(f_A \mid A \in \mathbb{Z}_{q(n)}^{n \times m(n)})_{n,q(n),\beta(n),m(n)} \text{ mit } f_A(z) = A \cdot z \pmod{q} \text{ (für } z \in \{0, 1\}^{m(n)})$$

ist nicht kollisionsresistent.

- Dann gibt es einen effizienten Algorithmus, der für eine Matrix  $A$  mit  $W$ -keit  $\geq \frac{1}{\text{poly}(n)}$  ein Paar  $(x, y)$  mit  $x \neq y$  und  $f_A(x) = f_A(y)$  bestimmt.
- Dann ist aber  $z := x - y \in \{-1, 0, 1\}^{m(n)}$  eine Lösung von  $\text{SIS}_{n,q(n),\beta(n),m(n)}$ , denn  $\beta(n) = \sqrt{m}$  wird gefordert.
- Also lassen sich Lösungen für  $\text{SIS}_{n,q(n)\beta(n),m(n)}$  effizient im Average-Case bestimmen, im Widerspruch zum Hauptsatz, Teil (a).

Wenn  $\mathcal{F} := (f_i : D_i \rightarrow \{0, 1\}^* \mid i \in I)$  eine Familie von kollisionsresistenten Hashfunktionen ist, dann ist  $\mathcal{F}$  eine One-Way-Funktion  $\implies$  SIS liefert One-Way-Funktionen!

# Beweis des Hauptsatzes: Das Smoothing Lemma

# Beweis des Hauptsatzes: Das Smoothing Lemma (1/2)

Angenommen, SIVP ist für das Gitter  $\mathcal{G}(B)$  zu lösen.

„Was tut“ die Normalverteilung  $\mathcal{N}(0, s^2)$ , eingeschränkt auf das Parallelotop  $P(B)$  für eine hinreichend große Standardabweichung  $s$ ?

- (a) Bezeichne die uniforme Verteilung auf einer Teilmenge  $S \subseteq \mathbb{R}^n$  mit  $U_S$ .
- (b) Die  $n$ -dimensionale Dichtefunktion der Normalverteilung (mit Mittelwert 0 und Standardabweichung  $s$ ) wird definiert durch

$$\rho_s(x) := \frac{1}{s^n} \cdot \exp\left(-\pi \cdot \frac{\|x\|_2^2}{s^2}\right).$$

# Beweis des Hauptsatzes: Das Smoothing Lemma (1/2)

Angenommen, SIVP ist für das Gitter  $\mathcal{G}(B)$  zu lösen.

„Was tut“ die Normalverteilung  $\mathcal{N}(0, s^2)$ , eingeschränkt auf das Parallelotop  $P(B)$  für eine hinreichend große Standardabweichung  $s$ ?

- (a) Bezeichne die uniforme Verteilung auf einer Teilmenge  $S \subseteq \mathbb{R}^n$  mit  $U_S$ .
- (b) Die  $n$ -dimensionale Dichtefunktion der Normalverteilung (mit Mittelwert 0 und Standardabweichung  $s$ ) wird definiert durch

$$\rho_s(x) := \frac{1}{s^n} \cdot \exp\left(-\pi \cdot \frac{\|x\|_2^2}{s^2}\right).$$

- (c) Die Normalverteilung  $\mathcal{N}(0, s^2) \bmod P_B$ :
  - ▶ Erzeuge Vektoren  $v = \sum_{i=1}^n \alpha_i \cdot b_i$  mit der konventionellen Normalverteilung,
  - ▶ danach „reduziere“  $v$  durch  $\sum_{i=1}^n (\alpha_i \bmod 1) \cdot b_i$  in das Parallelotop  $P(B)$ .  
Für eine reelle Zahl  $x$  ist  $x \bmod 1 := x - \lfloor x \rfloor$  und  $\lceil x \rceil$  die zu  $x$  nächstliegende ganze Zahl.

# Beweis des Hauptsatzes: Das Smoothing Lemma (2/2)

Ziehe  $x \in \mathbb{R}^n$  mit der  $n$ -dimensionalen Normalverteilung  $\mathcal{N}(0, s^2)$ . Dann ist

$$\|x\| \leq \sqrt{n} \cdot s$$

mit Wahrscheinlichkeit  $1 - 2^{-\Omega(n)}$ . Das **Smoothing-Lemma** nutzt dies aus:

*Die Normalverteilung  $\mathcal{N}(0, s^2) \bmod P_B$  auf dem Parallelotop  $P(B)$  wird bei entsprechend großer Standardabweichung zur Gleichverteilung  $U_{P_B}$  und löst die Gitterstruktur auf.*

# Beweis des Hauptsatzes: Das Smoothing Lemma (2/2)

Ziehe  $x \in \mathbb{R}^n$  mit der  $n$ -dimensionalen Normalverteilung  $\mathcal{N}(0, s^2)$ . Dann ist

$$\|x\| \leq \sqrt{n} \cdot s$$

mit Wahrscheinlichkeit  $1 - 2^{-\Omega(n)}$ . Das **Smoothing-Lemma** nutzt dies aus:

*Die Normalverteilung  $\mathcal{N}(0, s^2) \bmod P_B$  auf dem Parallelotop  $P(B)$  wird bei entsprechend großer Standardabweichung zur Gleichverteilung  $U_{P_B}$  und löst die Gitterstruktur auf.*

Sei  $B$  eine  $n \times n$ -Matrix mit vollem Rank. Wenn  $s > \sqrt{n} \cdot \lambda_n(\mathcal{G}(B))$ , dann gilt

$$\Delta(U_{P_B}, \mathcal{N}(0, s^2) \bmod P_B) \leq 2^{-n}.$$

$\Delta$  ist die Variationsdistanz zwischen Gleich- und Normalverteilung.

Approximiere  $\lambda_n(\mathcal{G}(B))$



# Beweis des Hauptsatzes: Approximiere $\lambda_n(\mathcal{G}(B))$

Zeige, dass SIVP einen  $\gamma$ -approximativen Algorithmus für das Gitter  $\mathcal{G}(B)$  besitzt, falls SIS im Average-Case gelöst werden kann.

Bestimme eine möglichst kurze Basis mit Hilfe eines SIS-Orakels.

# Beweis des Hauptsatzes: Approximiere $\lambda_n(\mathcal{G}(B))$

Zeige, dass SIVP einen  $\gamma$ -approximativen Algorithmus für das Gitter  $\mathcal{G}(B)$  besitzt, falls SIS im Average-Case gelöst werden kann.

Bestimme eine möglichst kurze Basis mit Hilfe eines SIS-Orakels.

SIVP $_{\gamma}$  ist im **Worst-Case** zu lösen, das SIS-Orakel funktioniert nur im **Average-Case**.

- ! Die Anfragen an das Orakel müssen (fast) unabhängig von Matrix  $B$  sein!
- ? Wie soll das funktionieren?

# Beweis des Hauptsatzes: Approximiere $\lambda_n(\mathcal{G}(B))$

Zeige, dass SIVP einen  $\gamma$ -approximativen Algorithmus für das Gitter  $\mathcal{G}(B)$  besitzt, falls SIS im Average-Case gelöst werden kann.

Bestimme eine möglichst kurze Basis mit Hilfe eines SIS-Orakels.

SIVP $_{\gamma}$  ist im **Worst-Case** zu lösen, das SIS-Orakel funktioniert nur im **Average-Case**.

! Die Anfragen an das Orakel müssen (fast) unabhängig von Matrix  $B$  sein!

? Wie soll das funktionieren?

- 1 Berechne schrittweise eine neue Basis  $B'$  aus der alten Basis  $B$ , so dass die maximale Länge eines Basisvektors in jedem Schritt mindestens halbiert wird.
  - ▶ Versuche durch Aufruf der Prozedur „**Finde kurzen Vektor**“ *kurze, linear unabhängige* Vektoren  $v_i$  zu bestimmen.
- 2 Wiederhole solange bis eine  $\gamma$ -approximative Lösung für SIVP gefunden ist.

# Beweis des Hauptsatzes: Finde kurzen Vektor (1/2)

## *Finde kurzen Vektor:*

1. Sei  $q = \Omega(n^2 \log n)$ ,  $m = \Theta(n \log q)$  und  $\beta = \sqrt{m}$ . (Also ist  $\text{SIS}_{n,q,\beta,m}$  lösbar.)
2. Ziehe  $m$  Vektoren  $x_1, \dots, x_m \in \mathbb{R}^n$  gemäß  $\mathcal{N}(0, s^2)$  für  $s \approx \sqrt{n}$

# Beweis des Hauptsatzes: Finde kurzen Vektor (1/2)

## *Finde kurzen Vektor:*

1. Sei  $q = \Omega(n^2 \log n)$ ,  $m = \Theta(n \log q)$  und  $\beta = \sqrt{m}$ . (Also ist  $\text{SIS}_{n,q,\beta,m}$  lösbar.)
2. Ziehe  $m$  Vektoren  $x_1, \dots, x_m \in \mathbb{R}^n$  gemäß  $\mathcal{N}(0, s^2)$  für  $s \approx \sqrt{n} \cdot \lambda_n(\mathcal{G}(B))$ .  
/\* Hochwahrscheinlich ist  $\|x_i\| \leq \sqrt{n} \cdot s$

# Beweis des Hauptsatzes: Finde kurzen Vektor (1/2)

## *Finde kurzen Vektor:*

1. Sei  $q = \Omega(n^2 \log n)$ ,  $m = \Theta(n \log q)$  und  $\beta = \sqrt{m}$ . (Also ist  $\text{SIS}_{n,q,\beta,m}$  lösbar.)
2. Ziehe  $m$  Vektoren  $x_1, \dots, x_m \in \mathbb{R}^n$  gemäß  $\mathcal{N}(0, s^2)$  für  $s \approx \sqrt{n} \cdot \lambda_n(\mathcal{G}(B))$ .  
/\* Hochwahrscheinlich ist  $\|x_i\| \leq \sqrt{n} \cdot s \approx n \cdot \lambda_n(\mathcal{G}(B))$ . Die Vektoren sind also kurz genug, werden aber leider nicht zum Gitter  $\mathcal{G}(B)$  gehören. \*/

# Beweis des Hauptsatzes: Finde kurzen Vektor (1/2)

## Finde kurzen Vektor:

1. Sei  $q = \Omega(n^2 \log n)$ ,  $m = \Theta(n \log q)$  und  $\beta = \sqrt{m}$ . (Also ist  $\text{SIS}_{n,q,\beta,m}$  lösbar.)

2. Ziehe  $m$  Vektoren  $x_1, \dots, x_m \in \mathbb{R}^n$  gemäß  $\mathcal{N}(0, s^2)$  für  $s \approx \sqrt{n} \cdot \lambda_n(\mathcal{G}(B))$ .

/\* Hochwahrscheinlich ist  $\|x_i\| \leq \sqrt{n} \cdot s \approx n \cdot \lambda_n(\mathcal{G}(B))$ . Die Vektoren sind also kurz genug, werden aber leider nicht zum Gitter  $\mathcal{G}(B)$  gehören. \*/

3. Reduziere jeden der  $m$  Vektoren  $x_i = \sum_{j=1}^n \alpha_j \cdot b_j$  in die Grundmasche  $P(B)$ , d.h. berechne die  $m$  Vektoren  $y_i = \sum_{j=1}^n (\alpha_j \bmod 1) \cdot b_j$ .

/\* Smoothing-Lemma: Die Vektoren  $y_i$  liegen beinahe gleichverteilt in  $P(B)$ . Die Vektoren  $x_i - y_i$  gehören zum Gitter  $\mathcal{G}(B)$ , sind aber möglicherweise zu lang! \*/

# Beweis des Hauptsatzes: Finde kurzen Vektor (1/2)

## Finde kurzen Vektor:

1. Sei  $q = \Omega(n^2 \log n)$ ,  $m = \Theta(n \log q)$  und  $\beta = \sqrt{m}$ . (Also ist  $\text{SIS}_{n,q,\beta,m}$  lösbar.)
2. Ziehe  $m$  Vektoren  $x_1, \dots, x_m \in \mathbb{R}^n$  gemäß  $\mathcal{N}(0, s^2)$  für  $s \approx \sqrt{n} \cdot \lambda_n(\mathcal{G}(B))$ .  
/\* Hochwahrscheinlich ist  $\|x_i\| \leq \sqrt{n} \cdot s \approx n \cdot \lambda_n(\mathcal{G}(B))$ . Die Vektoren sind also kurz genug, werden aber leider nicht zum Gitter  $\mathcal{G}(B)$  gehören. \*/
3. Reduziere jeden der  $m$  Vektoren  $x_i = \sum_{j=1}^n \alpha_j \cdot b_j$  in die Grundmasche  $P(B)$ , d.h. berechne die  $m$  Vektoren  $y_i = \sum_{j=1}^n (\alpha_j \bmod 1) \cdot b_j$ .  
/\* Smoothing-Lemma: Die Vektoren  $y_i$  liegen beinahe gleichverteilt in  $P(B)$ . Die Vektoren  $x_i - y_i$  gehören zum Gitter  $\mathcal{G}(B)$ , sind aber möglicherweise zu lang! \*/
4. Runde die Vektoren  $y_i$  zu Gitterpunkten  $z_i$  in  $\mathcal{G}^* := \mathcal{G}(\frac{1}{q} \cdot B)$ . Setze dazu zuerst

$$a_i := \lceil q \cdot B^{-1} \cdot y_i \rceil$$

und dann



# Beweis des Hauptsatzes: Finde kurzen Vektor (1/2)

## Finde kurzen Vektor:

1. Sei  $q = \Omega(n^2 \log n)$ ,  $m = \Theta(n \log q)$  und  $\beta = \sqrt{m}$ . (Also ist  $\text{SIS}_{n,q,\beta,m}$  lösbar.)
2. Ziehe  $m$  Vektoren  $x_1, \dots, x_m \in \mathbb{R}^n$  gemäß  $\mathcal{N}(0, s^2)$  für  $s \approx \sqrt{n} \cdot \lambda_n(\mathcal{G}(B))$ .  
/\* Hochwahrscheinlich ist  $\|x_i\| \leq \sqrt{n} \cdot s \approx n \cdot \lambda_n(\mathcal{G}(B))$ . Die Vektoren sind also kurz genug, werden aber leider nicht zum Gitter  $\mathcal{G}(B)$  gehören. \*/
3. Reduziere jeden der  $m$  Vektoren  $x_i = \sum_{j=1}^n \alpha_j \cdot b_j$  in die Grundmasche  $P(B)$ , d.h. berechne die  $m$  Vektoren  $y_i = \sum_{j=1}^n (\alpha_j \bmod 1) \cdot b_j$ .  
/\* Smoothing-Lemma: Die Vektoren  $y_i$  liegen beinahe gleichverteilt in  $P(B)$ . Die Vektoren  $x_i - y_i$  gehören zum Gitter  $\mathcal{G}(B)$ , sind aber möglicherweise zu lang! \*/
4. Runde die Vektoren  $y_i$  zu Gitterpunkten  $z_i$  in  $\mathcal{G}^* := \mathcal{G}(\frac{1}{q} \cdot B)$ . Setze dazu zuerst

$$a_i := \lceil q \cdot B^{-1} \cdot y_i \rceil$$

und dann

$$z_i := \frac{1}{q} \cdot B \cdot a_i.$$

Also ist  $z_i$  ein Gitterpunkt in  $\mathcal{G}^*$ , der nur bei „langer“ Basis  $B$  weit von  $y_i$  entfernt ist.

5. /\* Die Vektoren  $a_i$  sind fast gleichverteilt in  $\mathbb{Z}_q^n$ . Das  $\text{SIS}_{n,q,\beta,m}$ -Orakel funktioniert im Average-Case und „sollte“ für die gleichverteilten  $a_1, \dots, a_m$  funktionieren. \*/

Rufe das  $\text{SIS}_{n,q,\beta,m}$ -Orakel für  $a_1, \dots, a_m$  auf: Das Orakel antwortet mit einer Linearkombination  $\gamma \in \mathbb{Z}^m$ , so dass  $\|\gamma\| \leq \beta$  und

$$\sum_{i=1}^m \gamma_i \cdot a_i \equiv 0 \pmod{q}$$

/\* Es ist  $\sum_i \gamma_i \cdot a_i \equiv 0 \pmod{q}$  und deshalb ist  $\sum_i \gamma_i \cdot \frac{a_i}{q}$  ein ganzzahliger Vektor. Also ist  $\sum_i \gamma_i \cdot z_i = B \cdot \sum_i \gamma_i \cdot \frac{a_i}{q}$  und  $\sum_i \gamma_i \cdot z_i$  gehört zu  $\mathcal{G}(B)$ . \*/

5. /\* Die Vektoren  $a_i$  sind fast gleichverteilt in  $\mathbb{Z}_q^n$ . Das  $\text{SIS}_{n,q,\beta,m}$ -Orakel funktioniert im Average-Case und „sollte“ für die gleichverteilten  $a_1, \dots, a_m$  funktionieren. \*/

Rufe das  $\text{SIS}_{n,q,\beta,m}$ -Orakel für  $a_1, \dots, a_m$  auf: Das Orakel antwortet mit einer Linearkombination  $\gamma \in \mathbb{Z}^m$ , so dass  $\|\gamma\| \leq \beta$  und

$$\sum_{i=1}^m \gamma_i \cdot a_i \equiv 0 \pmod{q}$$

/\* Es ist  $\sum_i \gamma_i \cdot a_i \equiv 0 \pmod{q}$  und deshalb ist  $\sum_i \gamma_i \cdot \frac{a_i}{q}$  ein ganzzahliger Vektor. Also ist  $\sum_i \gamma_i \cdot z_i = B \cdot \sum_i \gamma_i \cdot \frac{a_i}{q}$  und  $\sum_i \gamma_i \cdot z_i$  gehört zu  $\mathcal{G}(B)$ . \*/

6. Der Vektor  $v := \sum_{i=1}^m \gamma_i \cdot (x_i - y_i + z_i)$  gehört zum Gitter  $\mathcal{G}(B)$ , denn alle Vektoren  $x_i - y_i$  wie auch der Vektor  $\sum_{i=1}^m \gamma_i \cdot z_i$  gehören zu  $\mathcal{G}(B)$ .

Für Vektoren  $w_1, \dots, w_N$  und Koeffizienten  $\alpha_1, \dots, \alpha_N$  gilt

$$\left\| \sum_{i=1}^N \alpha_i \cdot w_i \right\| \leq \sqrt{N} \cdot \|\alpha\| \cdot \max_{1 \leq i \leq N} \|w_i\|.$$

Für Vektoren  $w_1, \dots, w_N$  und Koeffizienten  $\alpha_1, \dots, \alpha_N$  gilt

$$\left\| \sum_{i=1}^N \alpha_i \cdot w_i \right\| \leq \sqrt{N} \cdot \|\alpha\| \cdot \max_{1 \leq i \leq N} \|w_i\|.$$

$$\|v\| \leq \left\| \sum_{i=1}^m \gamma_i \cdot x_i \right\| + \left\| \sum_{i=1}^m \gamma_i \cdot (y_i - z_i) \right\|$$

Für Vektoren  $w_1, \dots, w_N$  und Koeffizienten  $\alpha_1, \dots, \alpha_N$  gilt

$$\left\| \sum_{i=1}^N \alpha_i \cdot w_i \right\| \leq \sqrt{N} \cdot \|\alpha\| \cdot \max_{1 \leq i \leq N} \|w_i\|.$$

$$\begin{aligned} \|v\| &\leq \left\| \sum_{i=1}^m \gamma_i \cdot x_i \right\| + \left\| \sum_{i=1}^m \gamma_i \cdot (y_i - z_i) \right\| \\ &\leq \sqrt{m} \cdot \|\gamma\| \cdot \left( \max_i \|x_i\| + \max_i \|y_i - \frac{B}{q} \cdot \lceil qB^{-1}y_i \rceil \right) \end{aligned}$$

Für Vektoren  $w_1, \dots, w_N$  und Koeffizienten  $\alpha_1, \dots, \alpha_N$  gilt

$$\left\| \sum_{i=1}^N \alpha_i \cdot w_i \right\| \leq \sqrt{N} \cdot \|\alpha\| \cdot \max_{1 \leq i \leq N} \|w_i\|.$$

$$\begin{aligned} \|v\| &\leq \left\| \sum_{i=1}^m \gamma_i \cdot x_i \right\| + \left\| \sum_{i=1}^m \gamma_i \cdot (y_i - z_i) \right\| \\ &\leq \sqrt{m} \cdot \|\gamma\| \cdot \left( \max_i \|x_i\| + \max_i \left\| y_i - \frac{B}{q} \cdot \lceil qB^{-1}y_i \rceil \right\| \right) \\ &= \sqrt{m} \cdot \|\gamma\| \cdot \left( \|x\| + \left\| \frac{B}{q} \cdot (qB^{-1}y - \lceil qB^{-1}y \rceil) \right\| \right) \end{aligned}$$

Für Vektoren  $w_1, \dots, w_N$  und Koeffizienten  $\alpha_1, \dots, \alpha_N$  gilt

$$\left\| \sum_{i=1}^N \alpha_i \cdot w_i \right\| \leq \sqrt{N} \cdot \|\alpha\| \cdot \max_{1 \leq i \leq N} \|w_i\|.$$

$$\begin{aligned} \|v\| &\leq \left\| \sum_{i=1}^m \gamma_i \cdot x_i \right\| + \left\| \sum_{i=1}^m \gamma_i \cdot (y_i - z_i) \right\| \\ &\leq \sqrt{m} \cdot \|\gamma\| \cdot \left( \max_i \|x_i\| + \max_i \left\| y_i - \frac{B}{q} \cdot \lceil qB^{-1}y_i \rceil \right\| \right) \\ &= \sqrt{m} \cdot \|\gamma\| \cdot \left( \|x\| + \left\| \frac{B}{q} \cdot (qB^{-1}y - \lceil qB^{-1}y \rceil) \right\| \right) \\ &= \sqrt{m} \cdot \|\gamma\| \cdot \left( \|x\| + \left\| \frac{B}{q} \cdot \underbrace{u}_{\text{mit } u \in [-1, 1]^n} \right\| \right) \end{aligned}$$



Für Vektoren  $w_1, \dots, w_N$  und Koeffizienten  $\alpha_1, \dots, \alpha_N$  gilt

$$\left\| \sum_{i=1}^N \alpha_i \cdot w_i \right\| \leq \sqrt{N} \cdot \|\alpha\| \cdot \max_{1 \leq i \leq N} \|w_i\|.$$

$$\begin{aligned} \|v\| &\leq \left\| \sum_{i=1}^m \gamma_i \cdot x_i \right\| + \left\| \sum_{i=1}^m \gamma_i \cdot (y_i - z_i) \right\| \\ &\leq \sqrt{m} \cdot \|\gamma\| \cdot \left( \max_i \|x_i\| + \max_i \left\| y_i - \frac{B}{q} \cdot \lceil qB^{-1}y_i \rceil \right\| \right) \\ &= \sqrt{m} \cdot \|\gamma\| \cdot \left( \|x\| + \left\| \frac{B}{q} \cdot (qB^{-1}y - \lceil qB^{-1}y \rceil) \right\| \right) \\ &= \sqrt{m} \cdot \|\gamma\| \cdot \left( \|x\| + \left\| \frac{B}{q} \cdot \underbrace{u}_{\text{mit } u \in [-1,1]^n} \right\| \right) \\ &\leq \sqrt{m} \cdot \beta \cdot \left( s\sqrt{n} + \max_{u \in [-1,1]^n} \left\| \frac{B}{q} \cdot u \right\| \right) \end{aligned}$$

Für Vektoren  $w_1, \dots, w_N$  und Koeffizienten  $\alpha_1, \dots, \alpha_N$  gilt

$$\left\| \sum_{i=1}^N \alpha_i \cdot w_i \right\| \leq \sqrt{N} \cdot \|\alpha\| \cdot \max_{1 \leq i \leq N} \|w_i\|.$$

$$\begin{aligned} \|v\| &\leq \left\| \sum_{i=1}^m \gamma_i \cdot x_i \right\| + \left\| \sum_{i=1}^m \gamma_i \cdot (y_i - z_i) \right\| \\ &\leq \sqrt{m} \cdot \|\gamma\| \cdot \left( \max_i \|x_i\| + \max_i \left\| y_i - \frac{B}{q} \cdot \lceil qB^{-1}y_i \rceil \right\| \right) \\ &= \sqrt{m} \cdot \|\gamma\| \cdot \left( \|x\| + \left\| \frac{B}{q} \cdot (qB^{-1}y - \lceil qB^{-1}y \rceil) \right\| \right) \\ &= \sqrt{m} \cdot \|\gamma\| \cdot \left( \|x\| + \left\| \frac{B}{q} \cdot \underbrace{u}_{\text{mit } u \in [-1,1]^n} \right\| \right) \\ &\leq \sqrt{m} \cdot \beta \cdot \left( s\sqrt{n} + \max_{u \in [-1,1]^n} \left\| \frac{B}{q} \cdot u \right\| \right) \leq m \cdot \left( n \cdot \lambda_n(\mathcal{G}(B)) + \frac{n \cdot \max_i \|b_i\|}{q} \right). \end{aligned}$$

$$\|v\| \leq m \cdot n \cdot \left( \lambda_n(\mathcal{G}(B)) + \frac{\max_i \|b_i\|}{q} \right).$$

- Für  $q = \Omega(n^2 \cdot \log_2 n)$  wird der Beitrag  $\max_i \|b_i\|$  der alten Basis stark reduziert.
- Der Approximationsfaktor stimmt asymptotisch überein mit  $m \cdot n = n^2 \cdot \log_2 n$ .

$$\|v\| \leq m \cdot n \cdot \left( \lambda_n(\mathcal{G}(B)) + \frac{\max_i \|b_i\|}{q} \right).$$

- Für  $q = \Omega(n^2 \cdot \log_2 n)$  wird der Beitrag  $\max_i \|b_i\|$  der alten Basis stark reduziert.
- Der Approximationsfaktor stimmt asymptotisch überein mit  $m \cdot n = n^2 \cdot \log_2 n$ .

Ist SIVP innerhalb des Faktors  $\mathcal{O}(n^2 \log n)$  **nicht effizient im Worst-Case** approximierbar, dann ist  $\text{SIS}_{n,q,\beta,m}$  **im Average-Case schwierig** und liefert kollisionsresistentes Hashing, falls

$$\beta = \sqrt{m}, q = \Omega(n^2 \log_2 n), m = \Theta(n \log_2 q).$$

# LWE: Learning with Errors

# Paritätsfunktionen

Eine Funktion  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  heißt eine Paritätsfunktion genau dann, wenn

$$f(x) := \bigoplus_{i \in I} x_i$$

für eine Teilmenge  $I \subseteq \{1, \dots, n\}$ .

(a) Im **Paritätslernen** über  $D := \{0, 1\}^n$  bestimme  $I$  aus klassifizierten Beispielen  $(x_1, f(x_1)), \dots, (x_m, f(x_m))$ .

▶ Paritätslernen ist einfach: Löse ein lineares Gleichungssystem über  $\mathbb{Z}_2$ .

(b) „**Learning Parity with Noise (LPN)**“ ist schwieriger: Bestimme  $f$  aus den verrauschten Beispielen

$$(x_1, f(x_1) \oplus e_1), \dots, (x_m, f(x_m) \oplus e_m),$$

wobei die Bits  $e_i$  unabhängig voneinander mit kleiner Wahrscheinlichkeit für eine Eins ausgewürfelt werden.

# Das Kryptosystem LWE

Die LWE-Verteilung  $\text{LWE}_{n,q,g,D}$

für den Sicherheitsparameter  $n$ , eine Primzahl  $q$ , einen Vektor  $g \in \mathbb{Z}_q^n$  sowie eine Verteilung  $D$  auf  $\mathbb{Z}_q$

erzeugt Beispiele  $(a, b)$ , wobei  $a \in \mathbb{Z}_q^n$  gemäß der Gleichverteilung,  $e \in \mathbb{Z}_q$  gemäß der Verteilung  $D$  ausgewürfelt und  $b$  bestimmt wird durch

$$b := \langle g, a \rangle + e \pmod{q}.$$

# Das Kryptosystem LWE

Die LWE-Verteilung  $\text{LWE}_{n,q,g,D}$

für den Sicherheitsparameter  $n$ , eine Primzahl  $q$ , einen Vektor  $g \in \mathbb{Z}_q^n$  sowie eine Verteilung  $D$  auf  $\mathbb{Z}_q$

erzeugt Beispiele  $(a, b)$ , wobei  $a \in \mathbb{Z}_q^n$  gemäß der Gleichverteilung,  $e \in \mathbb{Z}_q$  gemäß der Verteilung  $D$  ausgewürfelt und  $b$  bestimmt wird durch

$$b := \langle g, a \rangle + e \pmod{q}.$$

Der Vektor  $a \in \mathbb{Z}_q^n$  wie auch das Ergebnis  $b$  ist also bekannt, das Geheimnis  $g \in \mathbb{Z}_q^n$  wie auch der Fehlerterm  $e$  hingegen sind unbekannt.



# LWE: Löse ein verrauschtes Gleichungssystem

Repräsentiere  $m$  – gemäß Verteilung  $\text{LWE}_{n,q,g,D}$  – erzeugte Beispiele

$$(a_i, b_i) \text{ mit } b_i = \langle g, a_i \rangle + e_i \pmod q$$

bei unbekanntem Geheimnis  $g$  durch

$$(A, b^T) \text{ mit } b^T = g^T \cdot A + e^T \pmod q,$$

wobei  $A = (a_1, \dots, a_m)$  und  $e = (e_1, \dots, e_m)$ .

- Die Matrix  $A$  wird zufällig gemäß der Gleichverteilung aus  $\mathbb{Z}_q^{n \times m}$  gezogen
- und der Fehlervektor  $e \in \mathbb{Z}_q^m$  gemäß der Fehlerverteilung  $D$ .

Wie schwierig ist LWE?

# Die LWE-Suchvermutung

Quantenalgorithmen  $F$  (mit Eingabe  $n, q$ ) dürfen ein Orakel  $O_{n,q,g,D}$  befragen, das

- entweder nur Beispiele  $(g, \langle x, a \rangle + e \pmod q)$  gemäß Verteilung  $\text{LWE}_{n,q,g,D}$  erzeugt – wir sprechen vom LWE-Orakel
- oder nur gleichverteilt-zufällige Beispiele  $(a, b) \in \mathbb{Z}_q^{n+1}$  – wir sprechen vom gleichverteilt-zufälligem Orakel  $R$ .

Die **LWE-Suchvermutung**:

Für jeden effizienten Quantenalgorithmus  $F$  ist die Wahrscheinlichkeit

$$\Pr_{g \in R \mathbb{Z}_q^n, (a,b) \sim \text{LWE}_{n,q,g,D}} [ F \text{ bestimmt } g \text{ nach Anfragen an das LWE-Orakel } ]$$

vernachlässigbar.

# Die LWE-Entscheidungsvermutung

Die **LWE-Entscheidungsvermutung**:

Für jeden effizienten Quantenalgorithmus  $F$  ist die Differenz

$$\left| \Pr_{g \in_R \mathbb{Z}_q^n, (a,b) \sim \text{LWE}_{n,q,g,D}} [ F \text{ akzeptiert nach Anfragen an das LWE-Orakel } ] - \Pr [ F \text{ akzeptiert nach Anfragen an das gleichverteilt-zufällige Orakel } R ] \right|$$

vernachlässigbar.

$\implies$  effiziente Algorithmen  $F$  können die LWE-Verteilung nicht von der Gleichverteilung unterscheiden.

# Schwierige Suche bei schwieriger Entscheidung

Die LWE-Entscheidungsvermutung garantiert, dass so gut wie keine Information über  $g$  rekonstruierbar ist. Es gelte  $q = \text{poly}(n)$ .

Aus der LWE-Entscheidungsvermutung folgt die LWE-Suchvermutung.

Zeige insbesondere, dass  $g_1 \stackrel{?}{=} 0$  überprüfbar ist, wenn  $F$  unterscheidet zwischen

- LWE-Beispielen  $(a, b)$  mit  $b = \langle g, a \rangle + e$  (für  $a, g \in \mathbb{Z}_q^n$  und  $e \in \mathbb{Z}_q$ ) sowie
- rein zufälligen Beispielen  $(a, c)$  (für gleichverteilt gewähltes  $c \in \mathbb{Z}_q$ ).

# Schwierige Suche bei schwieriger Entscheidung

Die LWE-Entscheidungsvermutung garantiert, dass so gut wie keine Information über  $g$  rekonstruierbar ist. Es gelte  $q = \text{poly}(n)$ .

Aus der LWE-Entscheidungsvermutung folgt die LWE-Suchvermutung.

Zeige insbesondere, dass  $g_1 \stackrel{?}{=} 0$  überprüfbar ist, wenn  $F$  unterscheidet zwischen

- LWE-Beispielen  $(a, b)$  mit  $b = \langle g, a \rangle + e$  (für  $a, g \in \mathbb{Z}_q^n$  und  $e \in \mathbb{Z}_q$ ) sowie
- rein zufälligen Beispielen  $(a, c)$  (für gleichverteilt gewähltes  $c \in \mathbb{Z}_q$ ).

1. Ersetze ein LWE-Beispiel  $(a_i, b_i)$  durch  $(a'_i, b_i)$  für  $a'_i := a_i - r_i \cdot (1, 0, \dots, 0)$ : Die Restklasse  $r_i \in \mathbb{Z}_q$  wird gleichverteilt zufällig gewählt.
2. Setze  $b'_i := \langle a'_i, g \rangle + e$ . Es ist

$$b_i = \langle a_i, g \rangle + e =$$

# Schwierige Suche bei schwieriger Entscheidung

Die LWE-Entscheidungsvermutung garantiert, dass so gut wie keine Information über  $g$  rekonstruierbar ist. Es gelte  $q = \text{poly}(n)$ .

Aus der LWE-Entscheidungsvermutung folgt die LWE-Suchvermutung.

Zeige insbesondere, dass  $g_1 \stackrel{?}{=} 0$  überprüfbar ist, wenn  $F$  unterscheidet zwischen

- LWE-Beispielen  $(a, b)$  mit  $b = \langle g, a \rangle + e$  (für  $a, g \in \mathbb{Z}_q^n$  und  $e \in \mathbb{Z}_q$ ) sowie
- rein zufälligen Beispielen  $(a, c)$  (für gleichverteilt gewähltes  $c \in \mathbb{Z}_q$ ).

1. Ersetze ein LWE-Beispiel  $(a_i, b_i)$  durch  $(a'_i, b_i)$  für  $a'_i := a_i - r_i \cdot (1, 0, \dots, 0)$ : Die Restklasse  $r_i \in \mathbb{Z}_q$  wird gleichverteilt zufällig gewählt.
2. Setze  $b'_i := \langle a'_i, g \rangle + e$ . Es ist

$$b_i = \langle a_i, g \rangle + e = \langle a'_i, g \rangle + g_1 \cdot r_i + e =$$

# Schwierige Suche bei schwieriger Entscheidung

Die LWE-Entscheidungsvermutung garantiert, dass so gut wie keine Information über  $g$  rekonstruierbar ist. Es gelte  $q = \text{poly}(n)$ .

Aus der LWE-Entscheidungsvermutung folgt die LWE-Suchvermutung.

Zeige insbesondere, dass  $g_1 \stackrel{?}{=} 0$  überprüfbar ist, wenn  $F$  unterscheidet zwischen

- LWE-Beispielen  $(a, b)$  mit  $b = \langle g, a \rangle + e$  (für  $a, g \in \mathbb{Z}_q^n$  und  $e \in \mathbb{Z}_q$ ) sowie
- rein zufälligen Beispielen  $(a, c)$  (für gleichverteilt gewähltes  $c \in \mathbb{Z}_q$ ).

1. Ersetze ein LWE-Beispiel  $(a_i, b_i)$  durch  $(a'_i, b_i)$  für  $a'_i := a_i - r_i \cdot (1, 0, \dots, 0)$ : Die Restklasse  $r_i \in \mathbb{Z}_q$  wird gleichverteilt zufällig gewählt.
2. Setze  $b'_i := \langle a'_i, g \rangle + e$ . Es ist

$$b_i = \langle a_i, g \rangle + e = \langle a'_i, g \rangle + g_1 \cdot r_i + e = b'_i + g_1 \cdot r_i.$$

►  $g_1 = 0 \iff$



# Schwierige Suche bei schwieriger Entscheidung

Die LWE-Entscheidungsvermutung garantiert, dass so gut wie keine Information über  $g$  rekonstruierbar ist. Es gelte  $q = \text{poly}(n)$ .

Aus der LWE-Entscheidungsvermutung folgt die LWE-Suchvermutung.

Zeige insbesondere, dass  $g_1 \stackrel{?}{=} 0$  überprüfbar ist, wenn  $F$  unterscheidet zwischen

- LWE-Beispielen  $(a, b)$  mit  $b = \langle g, a \rangle + e$  (für  $a, g \in \mathbb{Z}_q^n$  und  $e \in \mathbb{Z}_q$ ) sowie
- rein zufälligen Beispielen  $(a, c)$  (für gleichverteilt gewähltes  $c \in \mathbb{Z}_q$ ).

1. Ersetze ein LWE-Beispiel  $(a_i, b_i)$  durch  $(a'_i, b_i)$  für  $a'_i := a_i - r_i \cdot (1, 0, \dots, 0)$ : Die Restklasse  $r_i \in \mathbb{Z}_q$  wird gleichverteilt zufällig gewählt.
2. Setze  $b'_i := \langle a'_i, g \rangle + e$ . Es ist

$$b_i = \langle a_i, g \rangle + e = \langle a'_i, g \rangle + g_1 \cdot r_i + e = b'_i + g_1 \cdot r_i.$$

- ▶  $g_1 = 0 \iff b'_i = b_i$  und  $F$  klassifiziert Beispiele  $(a'_i, b_i)$  als LWE-Beispiele.
- ▶  $g_1 \neq 0 \iff$

# Schwierige Suche bei schwieriger Entscheidung

Die LWE-Entscheidungsvermutung garantiert, dass so gut wie keine Information über  $g$  rekonstruierbar ist. Es gelte  $q = \text{poly}(n)$ .

Aus der LWE-Entscheidungsvermutung folgt die LWE-Suchvermutung.

Zeige insbesondere, dass  $g_1 \stackrel{?}{=} 0$  überprüfbar ist, wenn  $F$  unterscheidet zwischen

- LWE-Beispielen  $(a, b)$  mit  $b = \langle g, a \rangle + e$  (für  $a, g \in \mathbb{Z}_q^n$  und  $e \in \mathbb{Z}_q$ ) sowie
- rein zufälligen Beispielen  $(a, c)$  (für gleichverteilt gewähltes  $c \in \mathbb{Z}_q$ ).

1. Ersetze ein LWE-Beispiel  $(a_i, b_i)$  durch  $(a'_i, b_i)$  für  $a'_i := a_i - r_i \cdot (1, 0, \dots, 0)$ : Die Restklasse  $r_i \in \mathbb{Z}_q$  wird gleichverteilt zufällig gewählt.
2. Setze  $b'_i := \langle a'_i, g \rangle + e$ . Es ist

$$b_i = \langle a_i, g \rangle + e = \langle a'_i, g \rangle + g_1 \cdot r_i + e = b'_i + g_1 \cdot r_i.$$

- ▶  $g_1 = 0 \iff b'_i = b_i$  und  $F$  klassifiziert Beispiele  $(a'_i, b_i)$  als LWE-Beispiele.
- ▶  $g_1 \neq 0 \iff b_i - b'_i \in_R \mathbb{Z}_q$  und  $F$  klassifiziert  $(a'_i, b_i)$  als zufällig.

# LWE: Average-Case = Worst-Case

Eine Selbstreduktion: Ist das LWE-Entscheidungsproblem im Average-Case effizient lösbar, dann auch im Worst-Case.

Der effiziente Algorithmus  $F$  löse das LWE-Entscheidungsproblem im **Average-Case**.

(a) Algorithmus  $F$  erhalte die Beispiele  $(a_i, b_i)$ :

Setze  $A := (a_1, \dots, a_i, \dots)$  und  $b = (b_1, \dots, b_i, \dots)$ .

(b) Der neue **Worst-Case**-Algorithmus  $F^*$ :

$F^*$  wählt ein zufälliges Geheimnis  $h \in \mathbb{Z}_q^n$  und übergibt  $(A, b^T + h^T \cdot A)$  an Algorithmus  $F$ . Das Votum von  $F$  wird übernommen.

► Es gilt  $b^T = g^T \cdot A + e^T \implies$

$$b^T + h^T \cdot A = g^T \cdot A + e^T + h^T \cdot A = (g^T + h^T) \cdot A + e^T$$

# LWE: Average-Case = Worst-Case

Eine Selbstreduktion: Ist das LWE-Entscheidungsproblem im Average-Case effizient lösbar, dann auch im Worst-Case.

Der effiziente Algorithmus  $F$  löse das LWE-Entscheidungsproblem im **Average-Case**.

(a) Algorithmus  $F$  erhalte die Beispiele  $(a_i, b_i)$ :

Setze  $A := (a_1, \dots, a_i, \dots)$  und  $b = (b_1, \dots, b_i, \dots)$ .

(b) Der neue **Worst-Case**-Algorithmus  $F^*$ :

$F^*$  wählt ein zufälliges Geheimnis  $h \in \mathbb{Z}_q^n$  und übergibt  $(A, b^T + h^T \cdot A)$  an Algorithmus  $F$ . Das Votum von  $F$  wird übernommen.

► Es gilt  $b^T = g^T \cdot A + e^T \implies$

$$b^T + h^T \cdot A = g^T \cdot A + e^T + h^T \cdot A = (g^T + h^T) \cdot A + e^T$$

und  $(A, b^T + h^T \cdot A)$  ist von der LWE-Verteilung erzeugbar.

# LWE: Average-Case = Worst-Case

Eine Selbstreduktion: Ist das LWE-Entscheidungsproblem im Average-Case effizient lösbar, dann auch im Worst-Case.

Der effiziente Algorithmus  $F$  löse das LWE-Entscheidungsproblem im **Average-Case**.

(a) Algorithmus  $F$  erhalte die Beispiele  $(a_i, b_i)$ :

Setze  $A := (a_1, \dots, a_i, \dots)$  und  $b = (b_1, \dots, b_i, \dots)$ .

(b) Der neue **Worst-Case**-Algorithmus  $F^*$ :

$F^*$  wählt ein zufälliges Geheimnis  $h \in \mathbb{Z}_q^n$  und übergibt  $(A, b^T + h^T \cdot A)$  an Algorithmus  $F$ . Das Votum von  $F$  wird übernommen.

▶ Es gilt  $b^T = g^T \cdot A + e^T \implies$

$$b^T + h^T \cdot A = g^T \cdot A + e^T + h^T \cdot A = (g^T + h^T) \cdot A + e^T$$

und  $(A, b^T + h^T \cdot A)$  ist von der LWE-Verteilung erzeugbar.

▶ Das Zufallsorakel bleibt bei Addition von  $h^T \cdot A$  gleichverteilt-zufällig.

Das LWE-Entscheidungsproblem sieht überall gleich aus!

# Der LWE-Hauptsatz

Sei  $D$  die Fehlerverteilung zur Gaußfunktion  $\rho_s(x) := \frac{1}{s} \cdot \exp(-\pi \cdot (x/s)^2)$ , eingeschränkt auf  $x \in \mathbb{Z}$ , mit  $s = \alpha \cdot q \geq 2 \cdot \sqrt{n}$  und  $0 < \alpha < 1$ . Sei  $q = \text{poly}(n)$ .

Für  $\gamma \approx \frac{n}{\alpha}$  gibt es einen effizienten Quanten-Algorithmus, der

SVP $_{\gamma}$  (für *beliebige*  $n$ -dimensionale Gitter)

auf das

LWE-Entscheidungsproblem (für die Parameter  $n$  und  $q$ ) im *Average-Case*

reduziert.

$\implies$  eine effiziente Lösung des LWE-Entscheidungsproblems im Average-Case bedingt eine Lösung von SVP $_{\gamma}$  im Worst-Case!

# LWE: Private-Key-Kryptographie

Die Verteilung  $D$  auf  $\mathbb{Z}_q$  bevorzuge kleine Restklassen, i.e.

$\text{pr}_{e \sim D}[e \geq q/4]$  ist vernachlässigbar.

1. Sei  $n$  der Sicherheitsparameter,  $q$  eine Primzahl.
2. Der *private* Schlüssel  $g \in \mathbb{Z}^n$  wird gemäß der Gleichverteilung auf  $\mathbb{Z}_q$  ausgewürfelt.
3. Würfel  $x \in \mathbb{Z}_q^n$  gemäß der Gleichverteilung und  $e \in \mathbb{Z}_q$  gemäß  $D$ . Das Bit  $\mu$  wird verschlüsselt durch

$$\text{Verschlüsselung}(g, \mu) := (x, b)$$

mit

$$b := \langle x, g \rangle + e +$$



Die Verteilung  $D$  auf  $\mathbb{Z}_q$  bevorzuge kleine Restklassen, i.e.

$\text{pr}_{e \sim D}[e \geq q/4]$  ist vernachlässigbar.

1. Sei  $n$  der Sicherheitsparameter,  $q$  eine Primzahl.
2. Der *private* Schlüssel  $g \in \mathbb{Z}^n$  wird gemäß der Gleichverteilung auf  $\mathbb{Z}_q$  ausgewürfelt.
3. Würfel  $x \in \mathbb{Z}_q^n$  gemäß der Gleichverteilung und  $e \in \mathbb{Z}_q$  gemäß  $D$ . Das Bit  $\mu$  wird verschlüsselt durch

$$\text{Verschlüsselung}(g, \mu) := (x, b)$$

mit

$$b := \langle x, g \rangle + e + \mu \cdot \lceil \frac{q}{2} \rceil \pmod{q}.$$

Die Verteilung  $D$  auf  $\mathbb{Z}_q$  bevorzuge kleine Restklassen, i.e.

$\text{pr}_{e \sim D}[e \geq q/4]$  ist vernachlässigbar.

1. Sei  $n$  der Sicherheitsparameter,  $q$  eine Primzahl.
2. Der *private* Schlüssel  $g \in \mathbb{Z}^n$  wird gemäß der Gleichverteilung auf  $\mathbb{Z}_q$  ausgewürfelt.
3. Würfel  $x \in \mathbb{Z}_q^n$  gemäß der Gleichverteilung und  $e \in \mathbb{Z}_q$  gemäß  $D$ . Das Bit  $\mu$  wird verschlüsselt durch

$$\text{Verschlüsselung}(g, \mu) := (x, b)$$

mit

$$b := \langle x, g \rangle + e + \mu \cdot \lceil \frac{q}{2} \rceil \pmod{q}.$$

4. *Entschlüsselung*( $g, x, b$ ): Dekodiere Geheimtext  $(x, b)$  als das Bit  $\mu'$

$$\mu' := \begin{cases} 0 \end{cases}$$

Die Verteilung  $D$  auf  $\mathbb{Z}_q$  bevorzuge kleine Restklassen, i.e.

$\text{pr}_{e \sim D}[e \geq q/4]$  ist vernachlässigbar.

1. Sei  $n$  der Sicherheitsparameter,  $q$  eine Primzahl.
2. Der *private* Schlüssel  $g \in \mathbb{Z}^n$  wird gemäß der Gleichverteilung auf  $\mathbb{Z}_q$  ausgewürfelt.
3. Würfel  $x \in \mathbb{Z}_q^n$  gemäß der Gleichverteilung und  $e \in \mathbb{Z}_q$  gemäß  $D$ . Das Bit  $\mu$  wird verschlüsselt durch

$$\text{Verschlüsselung}(g, \mu) := (x, b)$$

mit

$$b := \langle x, g \rangle + e + \mu \cdot \lceil \frac{q}{2} \rceil \pmod{q}.$$

4. *Entschlüsselung*( $g, x, b$ ): Dekodiere Geheimtext  $(x, b)$  als das Bit  $\mu'$

$$\mu' := \begin{cases} 0 & |b - \langle x, g \rangle| < q/4, \\ 1 & \text{sonst.} \end{cases}$$

IND-CPA: Was passiert bei Angriffen mit frei wählbarem Klartext?

Die LWE-Entscheidungsvermutung möge gelten. Sei  $A$  ein effizienter Quanten-Algorithmus, der Anfragen an das LWE-Orakel LWE richten darf. Dann ist

$$\left| \Pr_{g \in_R \mathbb{Z}_q^n, \mu \in \{0,1\}} [A \text{ dekodiert erfolgreich}] - \frac{1}{2} \right|$$

vernachlässigbar.

IND-CPA: Was passiert bei Angriffen mit frei wählbarem Klartext?

Die LWE-Entscheidungsvermutung möge gelten. Sei  $A$  ein effizienter Quanten-Algorithmus, der Anfragen an das LWE-Orakel LWE richten darf. Dann ist

$$\left| \Pr_{g \in_R \mathbb{Z}_q^n, \mu \in \{0,1\}} [A \text{ dekodiert erfolgreich}] - \frac{1}{2} \right|$$

vernachlässigbar.

- LWE-Entscheidungsvermutung: LWE- und Gleichverteilung auf  $\mathbb{Z}_q^{n+1}$  sind nur mit vernachlässigbarem Vorteil unterscheidbar.

IND-CPA: Was passiert bei Angriffen mit frei wählbarem Klartext?

Die LWE-Entscheidungsvermutung möge gelten. Sei  $A$  ein effizienter Quanten-Algorithmus, der Anfragen an das LWE-Orakel LWE richten darf. Dann ist

$$\left| \Pr_{g \in_R \mathbb{Z}_q^n, \mu \in \{0,1\}} [A \text{ dekodiert erfolgreich}] - \frac{1}{2} \right|$$

vernachlässigbar.

- LWE-Entscheidungsvermutung: LWE- und Gleichverteilung auf  $\mathbb{Z}_q^{n+1}$  sind nur mit vernachlässigbarem Vorteil unterscheidbar.
- LWE-Verschlüsselungen unterscheiden sich von einem LWE-Beispiel nur durch die mögliche Addition von  $\lceil q/2 \rceil$ .

IND-CPA: Was passiert bei Angriffen mit frei wählbarem Klartext?

Die LWE-Entscheidungsvermutung möge gelten. Sei  $A$  ein effizienter Quanten-Algorithmus, der Anfragen an das LWE-Orakel LWE richten darf. Dann ist

$$\left| \Pr_{g \in_R \mathbb{Z}_q^n, \mu \in \{0,1\}} [A \text{ dekodiert erfolgreich}] - \frac{1}{2} \right|$$

vernachlässigbar.

- LWE-Entscheidungsvermutung: LWE- und Gleichverteilung auf  $\mathbb{Z}_q^{n+1}$  sind nur mit vernachlässigbarem Vorteil unterscheidbar.
- LWE-Verschlüsselungen unterscheiden sich von einem LWE-Beispiel nur durch die mögliche Addition von  $\lceil q/2 \rceil$ .
- Beispiele mit  $\mu = 0$  und Beispiele mit  $\mu = 1$  sehen für einen Angreifer  $F$  wie rein zufällige Beispiele aus: Der Vorteil von  $F$  im Dekodieren ist vernachlässigbar.  $\square$

# LWE: Public-Key-Kryptographie



- (a) Das Geheimnis  $g$  muss nach Störung durch den Fehlervektor  $e$  rekonstruierbar bleiben.
- ▶ Fordere, dass  $D$  hochwahrscheinlich nur Fehlervektoren  $e$  mit hinreichend kleiner  $L_2$ -Norm erzeugt.
  - ▶ Insbesondere:  $D$  weise Restklassen außerhalb von

$$\{-\sqrt{q/(8n+4)}, \dots, \sqrt{q/(8n+4)}\}$$

nur vernachlässigbare Wahrscheinlichkeit zu.

- (a) Das Geheimnis  $g$  muss nach Störung durch den Fehlervektor  $e$  rekonstruierbar bleiben.
- ▶ Fordere, dass  $D$  hochwahrscheinlich nur Fehlervektoren  $e$  mit hinreichend kleiner  $L_2$ -Norm erzeugt.
  - ▶ Insbesondere:  $D$  weise Restklassen außerhalb von

$$\{-\sqrt{q/(8n+4)}, \dots, \sqrt{q/(8n+4)}\}$$

nur vernachlässigbare Wahrscheinlichkeit zu.

- (b) Übungsaufgabe: Die LWE-Entscheidungsvermutung bleibt richtig, wenn  $g$  durch  $D$  erzeugt wird.

1. Sei  $n$  der **Sicherheitsparameter**. Der **private** Schlüssel  $g \in \mathbb{Z}^n$  wird gemäß der Fehlerverteilung  $D$  auf  $\mathbb{Z}_q$  ausgewürfelt. Der **öffentliche** Schlüssel ist das Paar

$$(A, \underbrace{g^T \cdot A + e^T}_{:= b \in \mathbb{Z}_q^n}).$$

Ziehe die  $n \times n$ -Matrix  $A$  gleichverteilt über  $\mathbb{Z}_q$  und  $e \in \mathbb{Z}_q^n$  gemäß  $D$ .

1. Sei  $n$  der **Sicherheitsparameter**. Der **private** Schlüssel  $g \in \mathbb{Z}^n$  wird gemäß der Fehlerverteilung  $D$  auf  $\mathbb{Z}_q$  ausgewürfelt. Der **öffentliche** Schlüssel ist das Paar

$$(A, \underbrace{g^T \cdot A + e^T}_{:= b \in \mathbb{Z}_q^n}).$$

Ziehe die  $n \times n$ -Matrix  $A$  gleichverteilt über  $\mathbb{Z}_q$  und  $e \in \mathbb{Z}_q^n$  gemäß  $D$ .

2. Der Sender verschlüsselt ein Bit  $\mu \in \{0, 1\}$  wie folgt:
- ▶ Würfel die Vektoren  $r, r' \in \mathbb{Z}_q^n$  sowie die Restklasse  $r'' \in \mathbb{Z}_q$  gemäß  $D$  aus.
  - ▶ Die Verschlüsselung ist

$$Enc(\mu) := \underbrace{(A \cdot r + r' \pmod q)}_{:= \text{die Präambel } a \in \mathbb{Z}_q^n}, \underbrace{b^T \cdot r + r'' + \mu \cdot \lceil \frac{q}{2} \rceil \pmod q}_{:= \text{die Payload } \beta \in \mathbb{Z}_q}$$

3. Der Geheimtext  $(a, \beta)$  wird genau dann als Bit 0 dekodiert, wenn

$$|\beta - g^T \cdot a| < \frac{q}{4}.$$

$$Enc(\mu) := \underbrace{(A \cdot r + r' \pmod q)}_{:= \text{die Pramibel } a \in \mathbb{Z}_q^n}, \underbrace{(b^T \cdot r + r'' + \mu \cdot \lceil \frac{q}{2} \rceil \pmod q)}_{:= \text{die Payload } \beta \in \mathbb{Z}_q}$$

Der Geheimtext  $(a, \beta)$  wird genau dann als Bit 0 dekodiert, wenn

$$|\beta - g^T \cdot a| < \frac{q}{4}.$$

Ist das richtig?

$$\beta - \mu \cdot \lceil \frac{q}{2} \rceil - g^T \cdot a = (b^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r')$$

$$Enc(\mu) := \underbrace{(A \cdot r + r' \pmod q)}_{:= \text{die Pramibel } a \in \mathbb{Z}_q^n}, \underbrace{b^T \cdot r + r'' + \mu \cdot \lceil \frac{q}{2} \rceil \pmod q}_{:= \text{die Payload } \beta \in \mathbb{Z}_q}$$

Der Geheimtext  $(a, \beta)$  wird genau dann als Bit 0 dekodiert, wenn

$$|\beta - g^T \cdot a| < \frac{q}{4}.$$

Ist das richtig?

$$\begin{aligned} \beta - \mu \cdot \lceil \frac{q}{2} \rceil - g^T \cdot a &= (b^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r') \\ &= (g^T \cdot A \cdot r + e^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r') \end{aligned}$$

$$Enc(\mu) := \underbrace{(A \cdot r + r' \pmod q)}_{:= \text{die Pramibel } a \in \mathbb{Z}_q^n}, \underbrace{b^T \cdot r + r'' + \mu \cdot \lceil \frac{q}{2} \rceil \pmod q}_{:= \text{die Payload } \beta \in \mathbb{Z}_q}$$

Der Geheimtext  $(a, \beta)$  wird genau dann als Bit 0 dekodiert, wenn

$$|\beta - g^T \cdot a| < \frac{q}{4}.$$

Ist das richtig?

$$\begin{aligned} \beta - \mu \cdot \lceil \frac{q}{2} \rceil - g^T \cdot a &= (b^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r') \\ &= (g^T \cdot A \cdot r + e^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r') \\ &= e^T \cdot r + r'' - g^T \cdot r'. \end{aligned}$$

$$Enc(\mu) := \underbrace{(A \cdot r + r' \pmod{q})}_{:= \text{die Pramibel } a \in \mathbb{Z}_q^n}, \underbrace{(b^T \cdot r + r'' + \mu \cdot \lceil \frac{q}{2} \rceil \pmod{q})}_{:= \text{die Payload } \beta \in \mathbb{Z}_q}$$

Der Geheimtext  $(a, \beta)$  wird genau dann als Bit 0 dekodiert, wenn

$$|\beta - g^T \cdot a| < \frac{q}{4}.$$

Ist das richtig?

$$\begin{aligned} \beta - \mu \cdot \lceil \frac{q}{2} \rceil - g^T \cdot a &= (b^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r') \\ &= (g^T \cdot A \cdot r + e^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r') \\ &= e^T \cdot r + r'' - g^T \cdot r'. \end{aligned}$$

$$\implies |\beta - \mu \cdot \lceil \frac{q}{2} \rceil - g^T \cdot a| = |e^T \cdot r + r'' - g^T \cdot r'| \leq$$



$$Enc(\mu) := \underbrace{(A \cdot r + r' \pmod q)}_{:= \text{die Pramibel } a \in \mathbb{Z}_q^n}, \underbrace{b^T \cdot r + r'' + \mu \cdot \lceil \frac{q}{2} \rceil \pmod q}_{:= \text{die Payload } \beta \in \mathbb{Z}_q}$$

Der Geheimtext  $(a, \beta)$  wird genau dann als Bit 0 dekodiert, wenn

$$|\beta - g^T \cdot a| < \frac{q}{4}.$$

Ist das richtig?

$$\begin{aligned} \beta - \mu \cdot \lceil \frac{q}{2} \rceil - g^T \cdot a &= (b^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r') \\ &= (g^T \cdot A \cdot r + e^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r') \\ &= e^T \cdot r + r'' - g^T \cdot r'. \end{aligned}$$

$$\implies |\beta - \mu \cdot \lceil \frac{q}{2} \rceil - g^T \cdot a| = |e^T \cdot r + r'' - g^T \cdot r'| \leq \frac{q}{8n+4} \cdot (2n+1) =$$

$$Enc(\mu) := \underbrace{(A \cdot r + r' \pmod{q})}_{:= \text{die Pramibel } a \in \mathbb{Z}_q^n}, \underbrace{b^T \cdot r + r'' + \mu \cdot \lceil \frac{q}{2} \rceil \pmod{q}}_{:= \text{die Payload } \beta \in \mathbb{Z}_q}$$

Der Geheimtext  $(a, \beta)$  wird genau dann als Bit 0 dekodiert, wenn

$$|\beta - g^T \cdot a| < \frac{q}{4}.$$

Ist das richtig?

$$\begin{aligned} \beta - \mu \cdot \lceil \frac{q}{2} \rceil - g^T \cdot a &= (b^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r') \\ &= (g^T \cdot A \cdot r + e^T \cdot r + r'') - (g^T \cdot A \cdot r + g^T \cdot r') \\ &= e^T \cdot r + r'' - g^T \cdot r'. \end{aligned}$$

$$\implies |\beta - \mu \cdot \lceil \frac{q}{2} \rceil - g^T \cdot a| = |e^T \cdot r + r'' - g^T \cdot r'| \leq \frac{q}{8n+4} \cdot (2n+1) = \frac{q}{4}$$

IND-CPA: Was passiert bei Angriffen mit frei wählbarem Klartext?

Die LWE-Entscheidungsvermutung möge gelten. Sei  $A$  ein effizienter Quantenalgorithmus, der Anfragen an das LWE-Orakel richten darf. Dann ist

$$\left| \Pr_{g \in_D \mathbb{Z}_q^n, \mu \in \{0,1\}} [A \text{ dekodiert Verschlüsselung}(A, b, \mu) \text{ erfolgreich}] - \frac{1}{2} \right|$$

vernachlässigbar.

Beweis: Siehe Skript.

# Voll-Homomorphe Verschlüsselung

Berechne  $f(x_1, \dots, x_n)$  auf öffentlichen Rechnern:

- Die Rechner erhalten nur Zugriff auf die Verschlüsselungen  $v(x_1), \dots, v(x_n)$  und müssen die Verschlüsselung  $v(f(x_1, \dots, x_n))$  des Funktionswerts bestimmen.
- **Forderung:** Es gibt eine Funktion  $g$  mit vergleichbarer Auswertungskomplexität, so dass

$$\underbrace{v(f(x_1, \dots, x_m)) = g(v(x_1), \dots, v(x_m))}_{\text{Voll-homomorphe Verschlüsselung}}$$

# Voll-Homomorphe Verschlüsselung

Berechne  $f(x_1, \dots, x_n)$  auf öffentlichen Rechnern:

- Die Rechner erhalten nur Zugriff auf die Verschlüsselungen  $v(x_1), \dots, v(x_n)$  und müssen die Verschlüsselung  $v(f(x_1, \dots, x_n))$  des Funktionswerts bestimmen.
- **Forderung:** Es gibt eine Funktion  $g$  mit vergleichbarer Auswertungskomplexität, so dass

$$\underbrace{v(f(x_1, \dots, x_m)) = g(v(x_1), \dots, v(x_m))}_{\text{Voll-homomorphe Verschlüsselung}}$$

- 1 Die RSA-Verschlüsselung  $v(x) := x^e \pmod N$  verschlüsselt die Multiplikation:

$$v(x \cdot y) = (x \cdot y)^e \pmod N = x^e \cdot y^e \pmod N = v(x) \cdot v(y) \pmod N.$$

# Voll-Homomorphe Verschlüsselung

Berechne  $f(x_1, \dots, x_n)$  auf öffentlichen Rechnern:

- Die Rechner erhalten nur Zugriff auf die Verschlüsselungen  $v(x_1), \dots, v(x_n)$  und müssen die Verschlüsselung  $v(f(x_1, \dots, x_n))$  des Funktionswerts bestimmen.
- **Forderung:** Es gibt eine Funktion  $g$  mit vergleichbarer Auswertungskomplexität, so dass

$$\underbrace{v(f(x_1, \dots, x_m)) = g(v(x_1), \dots, v(x_m))}_{\text{Voll-homomorphe Verschlüsselung}}$$

- 1 Die RSA-Verschlüsselung  $v(x) := x^e \pmod N$  verschlüsselt die Multiplikation:

$$v(x \cdot y) = (x \cdot y)^e \pmod N = x^e \cdot y^e \pmod N = v(x) \cdot v(y) \pmod N.$$

- 2 Craig Gentry zeigt in 2009:

Arithmetische Schaltkreise mit  $t$  Gattern sind auf den verschlüsselten Daten in Zeit  $\mathcal{O}(t \cdot \text{polylog}(t \cdot n))$  auswertbar. ( $n = \text{Sicherheitsparameter.}$ )

Die Sicherheit der Berechnung folgt aus der LWE-Entscheidungsvermutung.

# Zusammenfassung

# Konsequenzen der Kryptographie für die Komplexitätstheorie

- (a) Wenn One-Way-Funktionen existieren, dann gilt  $P \neq NP$ .
- (b) Falls Pseudo-Random-Funktionen großer Komplexität existieren, dann gibt es keine natürlichen Beweise für den Nachweis von  $P \neq NP$ .
- (c) Sehr gute Pseudo-Random-Generatoren verringern die Lücke zwischen  $P$  und  $BPP$ .

Aber gibt es One-Way-Funktionen oder Trapdoor-Funktionen? Gilt  $P = BPP$ ?



# In welcher Welt leben wir?

1. *Algorithmica*: Es gilt  $P = NP$ .

- ▶ Enorme Kraft konventioneller Rechner,
- ▶ aber keine Kryptographie.

# In welcher Welt leben wir?

1. *Algorithmica*: Es gilt  $P = NP$ .

- ▶ Enorme Kraft konventioneller Rechner,
- ▶ aber keine Kryptographie.

2. *Heuristica*: Es gilt  $P \neq NP$ , aber  $\text{Avg}P = \text{Avg}NP$ .

- ▶ Heuristiken „funktionieren“, denn ihr Versagen wird nicht bemerkt.
- ▶ Die Kryptographie bleibt trivialisiert.

# In welcher Welt leben wir?

1. *Algorithmica*: Es gilt  $P = NP$ .
  - ▶ Enorme Kraft konventioneller Rechner,
  - ▶ aber keine Kryptographie.
2. *Heuristica*: Es gilt  $P \neq NP$ , aber  $AvgP = AvgNP$ .
  - ▶ Heuristiken „funktionieren“, denn ihr Versagen wird nicht bemerkt.
  - ▶ Die Kryptographie bleibt trivialisiert.
3. *Pessiland*:  $AvgP \neq AvgNP$ , aber es gibt keine One-Way-Funktionen.
  - ▶ Die Wunder der Heuristiken bleiben genauso aus
  - ▶ wie wichtige kryptographische Anwendungen.

# In welcher Welt leben wir?

1. *Algorithmica*: Es gilt  $P = NP$ .
  - ▶ Enorme Kraft konventioneller Rechner,
  - ▶ aber keine Kryptographie.
2. *Heuristica*: Es gilt  $P \neq NP$ , aber  $AvgP = AvgNP$ .
  - ▶ Heuristiken „funktionieren“, denn ihr Versagen wird nicht bemerkt.
  - ▶ Die Kryptographie bleibt trivialisiert.
3. *Pessiland*:  $AvgP \neq AvgNP$ , aber es gibt keine One-Way-Funktionen.
  - ▶ Die Wunder der Heuristiken bleiben genauso aus
  - ▶ wie wichtige kryptographische Anwendungen.
4. *Minicrypt*: Es gibt One-Way-Funktionen, aber keine Public-Key-Kryptographie!
  - ▶ Es gibt Pseudo-Random-Generatoren und Pseudo-Random-Funktionen,
  - ▶ Es gilt  $AvgP \neq AvgNP$ , da One-Way-Funktionen existieren.

# In welcher Welt leben wir?

1. *Algorithmica*: Es gilt  $P = NP$ .
  - ▶ Enorme Kraft konventioneller Rechner,
  - ▶ aber keine Kryptographie.
2. *Heuristica*: Es gilt  $P \neq NP$ , aber  $AvgP = AvgNP$ .
  - ▶ Heuristiken „funktionieren“, denn ihr Versagen wird nicht bemerkt.
  - ▶ Die Kryptographie bleibt trivialisiert.
3. *Pessiland*:  $AvgP \neq AvgNP$ , aber es gibt keine One-Way-Funktionen.
  - ▶ Die Wunder der Heuristiken bleiben genauso aus
  - ▶ wie wichtige kryptographische Anwendungen.
4. *Minicrypt*: Es gibt One-Way-Funktionen, aber keine Public-Key-Kryptographie!
  - ▶ Es gibt Pseudo-Random-Generatoren und Pseudo-Random-Funktionen,
  - ▶ Es gilt  $AvgP \neq AvgNP$ , da One-Way-Funktionen existieren.
5. *Cryptomania*: Gitterprobleme sind exponentiell hart im Mittel, der Traum der Kryptographie wird wahr.

## 1 Zentrale Konzepte

- ▶ One-Way-Funktionen
  - ★ Pseudo-Random-Generatoren, Pseudo-Random-Funktionen, Kollisionsresistente Hashfunktionen
- ▶ Trapdoor-Funktionen
  - ★ Digitale Signaturen

## 2 Quanten-Algorithmen können effizient faktorisieren, beißen sich aber an Gitterproblemen die Zähne aus.

## 3 Gitter-Kryptographie

- ▶ Methoden:
  - ★ **Short-Integer-Solution:** In  $\text{SIS}_{n,q,\beta,m}$  wird ein Vektor  $z$  für eine zufällige  $n \times m$  Matrix  $A$  (mit  $n \ll m$ ) gesucht, so dass  $A \cdot z = 0 \pmod q$  und  $\|z\| \leq \beta$ .
  - ★ **Learning-With-Errors:** Für eine zufällige  $n \times m$ -Matrix  $A$  und die Verschlüsselung  $b^T = g^T \cdot A + e^T \pmod q$  bestimme  $g$ .
- ▶ Führe Average-Case-Härte auf Worst-Case-Härte zurück.
- ▶ Voll-homomorphe Verschlüsselung.