Reed-Muller Codes

- Reed-Muller Code $\mathcal{R}(r, m)$ ist definiert für $m \in \mathbb{N}$, $0 \leq r \leq m$.
- Betrachten nur Reed-Muller Codes 1. Ordnung $\mathcal{R}(1, m) = \mathcal{R}(m)$.

Definition Rekursive Darstellung von Reed-Muller Codes

1. $\mathcal{R}(1) = \mathbb{F}_2^2 = \{00, 01, 10, 11\}$.
2. Für $m \geq 1$: $\mathcal{R}(m+1) = \{\mathbf{cc} \mid \mathbf{c} \in \mathcal{R}(m)\} \cup \{\mathbf{c}\bar{\mathbf{c}} \mid \mathbf{c} \in \mathcal{R}(m)\}$.

- $R_1 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ ist eine Generatormatrix für $\mathcal{R}(1)$.
- $\mathcal{R}(2) = \{0000, 0011, 0101, 0110, 1010, 1001, 1111, 1100\}$ mit Generatormatrix

$$R_2 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Parameter der Reed-Muller Codes

Satz Reed-Muller Parameter

$\mathcal{R}(m)$ ist ein linearer $(2^m, 2^{m+1}, 2^{m-1})$-Code. Für alle $\mathbf{c} \in \mathcal{R}(m) \setminus \{\mathbf{0}, \mathbf{1}\}$ gilt $w(\mathbf{c}) = 2^{m-1}$.

IA: $m = 1$

- $\mathcal{R}(1)$ ist ein linearer $(2^1, 2^2, 2^0)$-Code. $01, 10$ besitzen Gewicht 2^0.

IS: $m \to m + 1$

- $n = 2 \cdot 2^m = 2^{m+1}$.
- $\{\mathbf{cc} \mid \mathbf{c} \in \mathcal{R}(m)\}$ und $\{\mathbf{c\bar{c}} \mid \mathbf{c} \in \mathcal{R}(m)\}$ sind disjunkt, d.h. $k = 2 \cdot 2^{m+1} = 2^{m+2}$.
- Sei $\mathbf{c} \in \mathcal{R}(m) \setminus \{\mathbf{0}, \mathbf{1}\}$.
  - Für $\mathbf{cc}$ gilt $w(\mathbf{cc}) = 2w(\mathbf{c}) = 2 \cdot 2^{m-1} = 2^m$.
  - Für $\mathbf{c\bar{c}}$ gilt $w(\mathbf{c\bar{c}}) = w(\mathbf{c}) + w(\bar{\mathbf{c}}) = 2^{m-1} + (2^m - 2^{m-1}) = 2^m$.
- Für $\mathbf{c} = \mathbf{0}$ gilt $\mathbf{c\bar{c}} = \mathbf{01}$ mit $w(\mathbf{01}) = 2^m$.
- Für $\mathbf{c} = \mathbf{1}$ gilt $\mathbf{c\bar{c}} = \mathbf{10}$ mit $w(\mathbf{10}) = 2^m$.

Reed-Muller Generatormatrizen

Satz Generatormatrix für $\mathcal{R}(m)$

Sei R_m eine Generatormatrix für $\mathcal{R}(m)$. Dann ist

$$R_{m+1} = \left(\begin{array}{ccc|ccc} 0 & \dots & 0 & 1 & \dots & 1 \\ \multicolumn{3}{c|}{R_m} & \multicolumn{3}{c}{R_m} \end{array}\right)$$

eine Generatormatrix für $\mathcal{R}(m+1)$.

- **Ann.**: $\exists$ nicht-triviale Linearkombination, die **0** liefert.
- Linearkombination kann nicht nur die erste Zeile enthalten.
- D.h. es gibt eine nicht-triviale Linearkombination der Zeilen $2 \dots m+2$, die den Nullvektor auf der ersten Hälfte liefert. (Widerspruch: R_m ist Generatormatrix für $\mathcal{R}(m)$.)
- Sei C der Code mit Generatormatrix R_{m+1}.
- Für $\mathbf{c} \in \mathcal{R}(m)$ gilt: $\mathbf{cc} \in C$ und $\mathbf{c}\bar{\mathbf{c}} \in C$. D.h. $\mathcal{R}(m+1) \subseteq C$.
- $\dim(C) = m+1 = \dim(\mathcal{R}(m+1))$ und damit $C = \mathcal{R}(m+1)$.

Charakterisierung der Generatormatrizen

Bsp:

$$R_3 = \left(\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right)$$

Streiche Einserzeile aus R_m. Dann

- besitzen die Spaltenvektoren Länge m und
- bestehen aus Binärkodierungen von $0, 1, \ldots, 2^m - 1$.

Vergleich von Hamming, Simplex und Reed-Muller Codes

	$\mathcal{H}(m)$	$\mathcal{S}(m)$	$\mathcal{R}(m)$
Codewortlänge	$2^m - 1$	$2^m - 1$	2^m
Anzahl Codeworte	2^{2^m-1-m}	2^m	2^{m+1}
Distanz	3	2^{m-1}	2^{m-1}

Dekodierung von Reed-Muller Codes

- $\mathcal{R}(m)$ kann $\left\lfloor \frac{2^{m-1}-1}{2} \right\rfloor = 2^{m-2} - 1$ Fehler korrigieren.
- Syndrom-Tabelle besitzt $\frac{2^n}{M} = \frac{2^{2^m}}{2^{m+1}} = 2^{2^m-m-1}$ Zeilen.

Bsp: $\mathcal{R}(3)$ ist 1-fehlerkorrigierend.

$$R_3 = \begin{pmatrix} \mathbf{r_1} \\ \mathbf{r_2} \\ \mathbf{r_3} \\ \mathbf{r_4} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Sei $\mathbf{c} = \alpha_1\mathbf{r_1} + \alpha_2\mathbf{r_2} + \alpha_3\mathbf{r_3} + \alpha_4\mathbf{r_4}$. Es gilt

- $c_1+c_5 = \alpha_1(r_{11}+r_{15})+\alpha_2(r_{21}+r_{25})+\alpha_3(r_{31}+r_{35})+\alpha_4(r_{41}+r_{45}) = \alpha_1$
- $c_2+c_6 = \alpha_1(r_{12}+r_{16})+\alpha_2(r_{22}+r_{26})+\alpha_3(r_{32}+r_{36})+\alpha_4(r_{42}+r_{46}) = \alpha_1$
- Ebenso $\alpha_1 = c_3 + c_7 = c_4 + c_8$.

Mehrheitsdekodierung

- Suche für jede Zeile i Spaltenpaar (u, v), so dass sich die Spalten u, v nur in der i-ten Zeile unterscheiden. Liefert Gleichung für α_i.
- Für Zeile 1: $(1,5), (2,6), (3,7), (4,8)$, d.h. im Abstand 4.
- Für Zeile 2: $(1,3), (2,4), (5,7), (6,8)$, d.h. im Abstand 2.
- Für Zeile 3: $(1,2), (3,4), (5,6), (7,8)$, d.h. im Abstand 1.
- Für Zeile 4: nicht möglich.

- Erhalten für $\alpha_1, \alpha_2, \alpha_3$ jeweils 4 Gleichungen in verschiedenen c_i.
- Falls $\mathbf{x} = \mathbf{c} + \mathbf{e_i}$, ist genau 1 von 4 Gleichungen inkorrekt.

Algorithmus Mehrheitsdekodierung Reed-Muller Code $\mathcal{R}(m)$

1. Bestimme $\alpha_1, \ldots, \alpha_m$ per Mehrheitsentscheid.
2. Berechne $\mathbf{e} = \mathbf{x} - \sum_{i=1}^{m} \alpha_i \mathbf{r_i}$.
3. Falls $w(\mathbf{e}) \leq 2^{m-2} - 1$, dekodiere $\mathbf{c} = \mathbf{x} + \mathbf{e}$. (d.h. $\alpha_{m+1} = 0$)
4. Falls $w(\bar{\mathbf{e}}) \leq 2^{m-2} - 1$, dekodiere $\mathbf{c} = \mathbf{x} + \bar{\mathbf{e}}$. (d.h. $\alpha_{m+1} = 1$)

Beispiel Mehrheitsdekodierung

- Verwenden $\mathcal{R}(3)$ und erhalten $\mathbf{x} = 11011100$.
  - $\alpha_1 = x_1 + x_5 = 0$
  - $\alpha_1 = x_2 + x_6 = 0$
  - $\alpha_1 = x_3 + x_7 = 0$
  - $\alpha_1 = x_4 + x_8 = 1$
- Mehrheitsentscheid liefert $\alpha_1 = 0$.
  - $\alpha_2 = x_1 + x_3 = 1$
  - $\alpha_2 = x_2 + x_4 = 0$
  - $\alpha_2 = x_5 + x_7 = 1$
  - $\alpha_2 = x_6 + x_8 = 1$
- Mehrheitsentscheid liefert $\alpha_2 = 1$ und analog $\alpha_3 = 0$.
- $\mathbf{e} = \mathbf{x} - 0 \cdot \mathbf{r_1} - 1 \cdot \mathbf{r_2} - 0 \cdot \mathbf{r_3} = 11011100 - 00110011 = 11101111$.
- $w(\bar{\mathbf{e}}) \leq 1$, d.h. $\mathbf{c} = \mathbf{x} + \bar{\mathbf{e}} = 11001100$.

McEliece Verfahren (1978)

- **Dekodieren eines zufälligen linearen Codes ist NP-hart.**
- Verwende linearen Code C mit effizientem Dekodierverfahren (z.B. sogenannten Goppa-Code).
- Generatormatrix von C bildet den geheimen Schlüssel.
- C wird in äquivalenten linearen Code C' transformiert.

Algorithmus Schlüsselgenerierung McEliece

1. Wähle linearen $[n, k, d]$-Code C mit Generatormatrix G.
2. Wähle zufällige binäre $(k \times k)$-Matrix S mit $\det(S) = 1$.
3. Wähle zufällige binäre $(n \times n)$-Permutationsmatrix P.
4. $G' \leftarrow SGP$

öffentlicher Schlüssel: G', geheimer Schlüssel S, G, P.

McEliece Verschlüsselung

Algorithmus McEliece Verschlüsselung

EINGABE: Plaintext $\mathbf{m} \in \mathbb{F}_2^k$

1. Wähle zufälligen Fehlervektor $\mathbf{e} \in \mathbb{F}_2^n$ mit $w(\mathbf{e}) = \lfloor \frac{d-1}{2} \rfloor$.
2. $\mathbf{c} \leftarrow \mathbf{m}G' + \mathbf{e}$.

AUSGABE: Ciphertext $\mathbf{c} \in \mathbb{F}_2^n$

Vorgeschlagene Parameter:

- $[1024, 512, 101]$-Goppacode C.
- Plaintextlänge: 512 Bit, Chiffretextlänge: 1024 Bit.
- Größe des öffentlichen Schlüssels: 512×1024 Bit.

McEliece Entschlüsselung

Algorithmus McEliece Entschlüsselung

EINGABE: Ciphertext $\mathbf{c} \in \mathbb{F}_2^n$

1. $\mathbf{x} \leftarrow \mathbf{c}P^{-1}$.
2. Dekodiere $\mathbf{x}$ mittels Dekodieralgorithmus für C zu $\mathbf{m}'$.
3. $\mathbf{m} \leftarrow \mathbf{m}'S^{-1}$

AUSGABE: Plaintext $\mathbf{m} \in \mathbb{F}_2^k$

- **Korrektheit**:

  $$\mathbf{x} = \mathbf{c}P^{-1} = (\mathbf{m}G' + \mathbf{e}) \cdot P^{-1} = (\mathbf{m}SGP + \mathbf{e}) \cdot P^{-1} = (\mathbf{m}S)G + \mathbf{e} \cdot P^{-1}.$$

- $\mathbf{e} \cdot P^{-1}$ besitzt Gewicht $w(\mathbf{e}P^{-1}) = w(\mathbf{e}) = \lfloor \frac{d-1}{2} \rfloor$.
- Dekodierung liefert $\mathbf{m}' = \mathbf{m}S$, d.h. $\mathbf{m} = \mathbf{m}'S^{-1}$.

Stern Identifikationsschema

- Dekodieren eines zufälligen linearen Codes ist NP-hart.
- Definiere zufälligen Code mittels zufälliger Parity Check Matrix.

Gegeben: $[n,k]$-Code C mittels $P \in \mathbb{F}_2^{(n-k)\times n}$ und $\mathbf{x} \in \mathbb{F}_2^n$
Gesucht: $\mathbf{e} \in \mathbb{F}_2^n$ mit $\mathbf{x} - \mathbf{e} = \mathbf{c} \in C$, so dass $w(\mathbf{e})$ minimal ist, d.h. gesucht ist $\mathbf{e}$ minimalen Gewichts mit $S(\mathbf{x}) = S(\mathbf{e}) = \mathbf{e}P^t$.

Algorithmus Stern Schlüsselerzeugung

Globale Parameter:

1. Parity Check Matrix $P \in \mathbb{F}_2^{(n-k)\times n}$ mit linear unabhängigen Zeilen.
2. Gewicht $g \in \mathbb{N}$

Jeder Teilnehmer wählt

1. Geheimer Schlüssel: $\mathbf{e} \in \mathbb{F}_2^n$ mit $w(\mathbf{e}) = g$
2. Öffentlicher Schlüssel: $S(\mathbf{e}) = \mathbf{e}P^t$

- Vorgeschlagen: $[n,k] = [512, 256]$ und $g = w(\mathbf{e}) = 56$.
- **Idee Identifikation:** Beweise Besitz von $\mathbf{e}$, ohne $\mathbf{e}$ preiszugeben.

Identifikationsverfahren

Algorithmus Stern Identifikation

Prover: Wähle zufällige $\mathbf{y} \in \mathbb{F}_2^n$ und Permutation $\sigma : \mathbb{F}_2^n \to \mathbb{F}_2^n$.
Hinterlege beim Verifier (sogenanntes Commitment)

$$c_0 = \sigma(\mathbf{y} + \mathbf{e}), c_1 = \sigma(\mathbf{y}) \text{ und } c_2 = (\sigma, \mathbf{y}P^t).$$

Verifier: Wähle zufälliges $b \in \{0, 1, 2\}$ für Prüfung von $c_i, i \neq b$.

Prover: Falls $b = 0$: Sende $\mathbf{y}$,σ und öffne c_1, c_2.
Falls $b = 1$: Sende $\mathbf{y} + \mathbf{e}, \sigma$ und öffne c_0, c_2.
Falls $b = 2$: Sende $\sigma(\mathbf{y}), \sigma(\mathbf{e})$ und öffne c_0, c_1.

Verifier: Falls $b = 0$: Prüfe Korrektheit von c_1 und c_2.
Falls $b = 1$: Prüfe c_0 und $c_2 = (\sigma, (\mathbf{y} + \mathbf{e})P^t - \mathbf{e}P^t)$.
Falls $b = 2$: Prüfe $c_0 = \sigma(\mathbf{y}) + \sigma(\mathbf{e}), c_1, w(\sigma(\mathbf{e})) = w(\mathbf{e})$.

Korrektheit: Nur Besitzer von **e** bestehen Protokoll.

- **Completeness:** Falls Prover **e** besitzt, akzeptiert Verifier.
- **Soundness:** Falls Prover **e** nicht besitzt, besteht er das Protokoll mit Ws höchstens $\frac{2}{3}$.
- **Strategie 1**: Prover wählt σ, **y** und $\tilde{\mathbf{e}}$ mit Gewicht $w(\tilde{\mathbf{e}})$.
  - Prover besteht nur $b = 1$ nicht, da hier $\tilde{\mathbf{e}}P^t \neq \mathbf{e}P^t$.
- **Strategie 2**: Prover wählt σ, **y** und $\mathbf{y} + \tilde{\mathbf{e}}$ mit $\tilde{\mathbf{e}}P^t = \mathbf{e}P^t$.
  - Prover besteht nur $b = 2$ nicht, da hier $w(\tilde{\mathbf{e}}) \neq w(\mathbf{e})$.
- Prover wählt **Strategie 1** und **Strategie 2** jeweils mit Ws $\frac{1}{2}$.

$$\begin{aligned}&\mathrm{Ws}(\text{P besteht Protokoll})\\ &= \mathrm{Ws}(b \neq 1) \cdot \mathrm{Ws}(\textbf{Strategie 1}) + \mathrm{Ws}(b \neq 2) \cdot \mathrm{Ws}(\textbf{Strategie 2})\\ &= \tfrac{2}{3}\left(\tfrac{1}{2} + \tfrac{1}{2}\right) = \tfrac{2}{3}.\end{aligned}$$

Fakt (Beweis ist nicht-trivial)

Jeder Angreifer mit $\mathrm{Ws} > \frac{2}{3}$ liefert Berechnung von **e**.

- Intuitiv: Prover kann nur $b = 1$ und 2 bestehen, falls er **e** kennt.

Zeroknowledge Eigenschaft

- **Zeroknowledge:** Verifier lernt nichts über **e**.
- Verifier lernt für
  - $b = 0$: Zufälliges $\mathbf{y} \in \mathbb{F}_2^n$, unabhängig von **e**.
  - $b = 1$: Zufälliges $\mathbf{y} + \mathbf{e} \in \mathbb{F}_2^n$, da $\mathbf{y} \in \mathbb{F}_2^n$ zufällig ist.
    (D.h. **y** ist One-Time Pad für **e**.)
  - $b = 2$: Zufälliges $\sigma(\mathbf{e}) \in \mathbb{F}_2^n$ mit Gewicht $w(\mathbf{e})$.
- Formaler Zeroknowledge Beweis verwendet Simulator für Prover, ohne dabei **e** zu kennen.