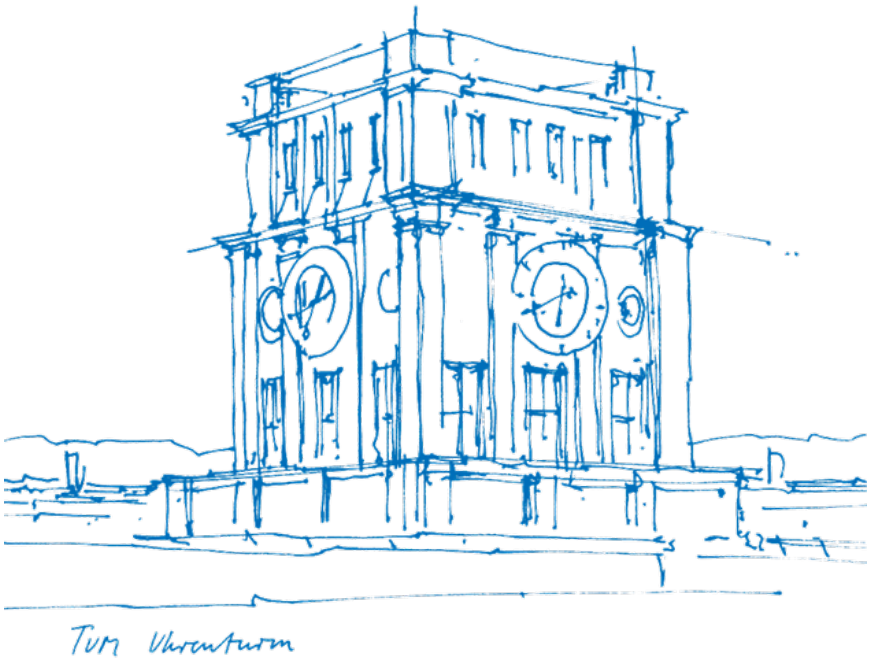
Lehrstuhl für Sicherheit in der Informationstechnik
School of Computation, Information and Technology
Technische Universität München

Parameteroptimierung für Multi-Input Higher-Order Alphabet Quantization

Marius Drechsler



Lehrstuhl für Sicherheit in der Informationstechnik
School of Computation, Information and Technology
Technische Universität München

Parameteroptimierung für Multi-Input Higher-Order Alphabet Quantization

Marius Drechsler

Bericht zur Ableistung der

Ingenieurspraxis

an der School of Computation, Information and Technology der Technischen Universität München.

Prüfer:
Dr. Michael Pehl

Betreuer:
M.Sc. Jonas Ruchti

Eingereicht am:
Munich, 30.05.2025

Inhaltsverzeichnis

1. Motivation und Problemstellung

Physical Unclonable Functions (PUFs) stellen in der Kryptographie einen spannenden Bereich zur Generierung und sicheren Speicherung Schlüsseln da. Durch minimale, nicht reproduzierbare Abweichungen im Fertigungsprozess lässt sich die Einzigartigkeit eines Schaltkreises nutzen, um ein Geheimnis direkt und sicher auf dem Chip zu speichern.

Ein Problem bei der Verwendung der Einzigartigkeiten dieser Schaltkreise ist die verlässliche Rekonstruktion eines Schlüssels. Da diese minimalen Unterschiede nur gemessen werden können, ist das Ergebnis von einem unkontrollierbaren Messfehler behaftet, welcher unter Umständen den neuen generierten Schlüssel verfälscht.

In der Regel lässt sich dieses Problem durch die Verwendung von Fehlerkorrekturcodes beheben. Diese werden üblicherweise nach der Quantisierung — also der Diskretisierung der gemessenen Werte angewendet.

Aufbauend auf die Bachelorarbeit „Towards Efficient Helper Data Algorithms for Mulit-Bit PUF Quantization“ soll hier die Praktikabilität und Umsetzbarkeit einer neuen Methode zur Verbesserung der Bitfehlerrate bei einer PUF Quantisierung analysiert werden.

2. Technischer Hintergrund

Das Betreiben einer PUF beinhaltet zwei verschiedene Arbeitsschritte: **Enrollment** und **Reconstruction**.
Als Enrollment wird die erste Messung des Verhaltens des Schaltkreises bezeichnet. Diese kann direkt in der Fertigungsstätte des Schaltkreises durchgeführt werden. Da bis zu diesem Punkt noch keine andere Messung mit dem Schaltkreis durchgeführt worden ist, können die Ergebnisse aus diesem Schritt als unveränderlichen Referenzwert für das Geheimnis des Schaltkreises angenommen werden. Anschließend wird aus den Messergebnissen mittels eines Quantisierungsprozesses ein geheimer Schlüssel generiert.
Reconstruction bezeichnet jede weitere Messung des Verhaltens des Schaltkreises. Da Messfehler in diesem Schritt nicht ausgeschlossen werden können, ist davon auszugehen, dass das hier gemessene Geheimnis nicht mit dem Referenz-Geheimnis bzw. dem geheimen Schlüssel nach der Enrollment Phase vollständig übereinstimmt. Die Anzahl der Bits, die zwischen diesen beiden Schlüsseln verschieden ist, ist als Bitfehlerrate definiert. Zusätzlich ist davon auszugehen, dass die Messwerte einer PUF normalverteilt und mittelwertfrei sind.

Die Ausgangslage der Praxis stellt die Bachelorarbeit „Towards Efficient Helper Data Algorithms for Multi-Bit PUF Quantization“ da. Konkret wurden in der Arbeit zwei verschiedene Methoden zur Verbesserung der Bitfehlerrate nach der Reconstruction Phase für Quantisierungen höherer Ordnung analyisert. Die erste Methode beschreibt eine Verallgemeinerung der Two Metric Helper Data method (TMHD) [1]. Mit Hilfe von TMHD werden zwei verschiedene Quantisiererfunktionen definiert. Während der Enrollment Phase wird anschließend entschieden, welche der beiden Funktionen ein verlässlicheres Ergebnis bei wiederholten Messergebnissen hervorrufen wird. Die S-Metric Helper Data method (SMHD) verallgemeinert dieses Konzept auf die Quantisierung mit mehr als einem Bit [2]. Da mit der Publikation von Fischer [2] bereits eine mögliche Implementation von SMHD vorgestellt wurde, bildet die in der Arbeit vorgelegte Implementierung eine Basis um die Performanz der zweiten vorgestellten Methode einordnen zu können.

Im zweiten Teil der Arbeit wurde ein neuer Ansatz zur Verbesserung der Fehlerrate implementiert und genauer analysiert. Die Grundlage der neuen Methode ergibt sich aus der natürlichen Beschaffenheit der Standardnormalverteilung. Da der Erwartungswert einer mittelwertfreien Normalverteilung bei 0 liegt und ein Vorzeichen-basierter 1-bit Quantisierer seine Entscheidungsgrenze ebenfalls bei 0 definiert, sind die Messwerte welche nahe der 0 liegen aufgrund ihrer inhärenten Messschwankungen dazu anfällig, bei wiederholten Messungen und Quantisierungen unterschiedliche Ergebnisse zu verursachen.

Dieses Problem wird in Abbildung 1 grafisch verdeutlicht.

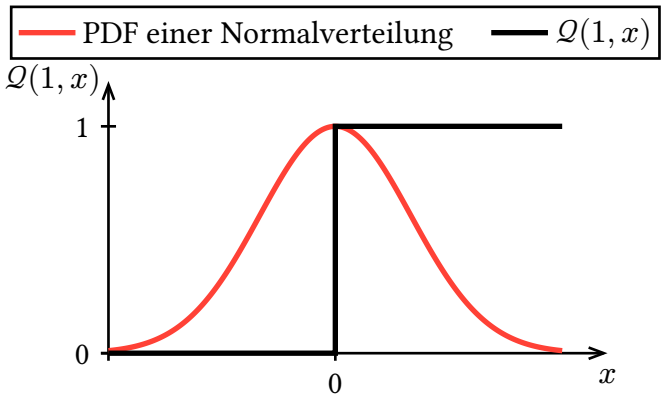


Abbildung 1: 1-bit Quantisierer mit normalverteilten Eingangswerten

Für die Umsetzung der neuen Methode werden gewichtete Summen aus mindestens 3 Eingangswerten – wie Ring-Oszillator-Differenzen – gebildet. Die Vorfaktoren der Summanden sind festgelegt als ± 1, wobei die jeweiligen Vorzeichen als Helperdaten abgespeichert werden.

$$f(\boldsymbol{x}, \boldsymbol{h}) = h_1 x_1 + h_2 x_2 + h_3 x_3 \tag{1}$$

Gleichung 1 zeigt die Struktur einer Funktion mit drei Eingangswerten und ihrer jeweiligen Gewichtung durch h_1, h_2 und h_3. Diese Vorfaktoren sollen nun so gewählt werden, dass die Werte der resultierenden Summen einen möglichst großen Abstand zu ihrer jeweils nächsten Quantisierergrenze haben.

Eine Lösung für den 1-bit Fall der in Abbildung 1 dargestellt wird, ist die betragsmäßige Maximierung der Werte der gewichteten Summen.

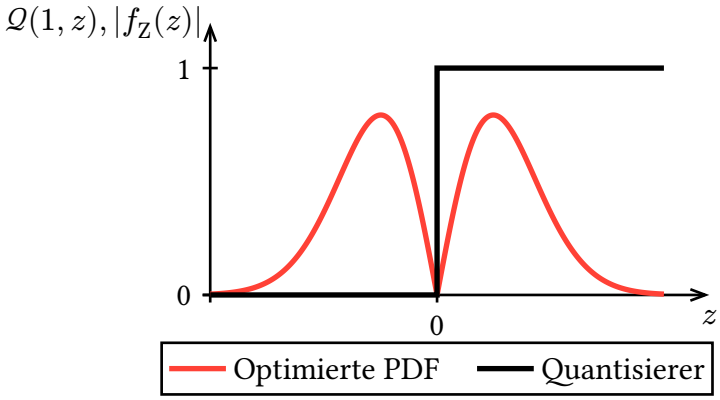


Abbildung 2: Darstellung der optimierten Eingangswerte

Mathematisch lässt sich dies durch die Maximierung des Betrags der Funktion aus Gleichung 1 herleiten:

$$\max_{h_1,h_2,h_3} |f(\boldsymbol{x}, \boldsymbol{h})| \tag{2}$$

Gleichung 2 definiert hiermit die Funktion zur Optimierung der Eingangwerte vor dem Quantisierer für den 1-bit Fall. Jedoch wird die Definition dieser Funktion für eine Vorbereitung der Quantisierung höherer Ordnung um einiges komplexer.

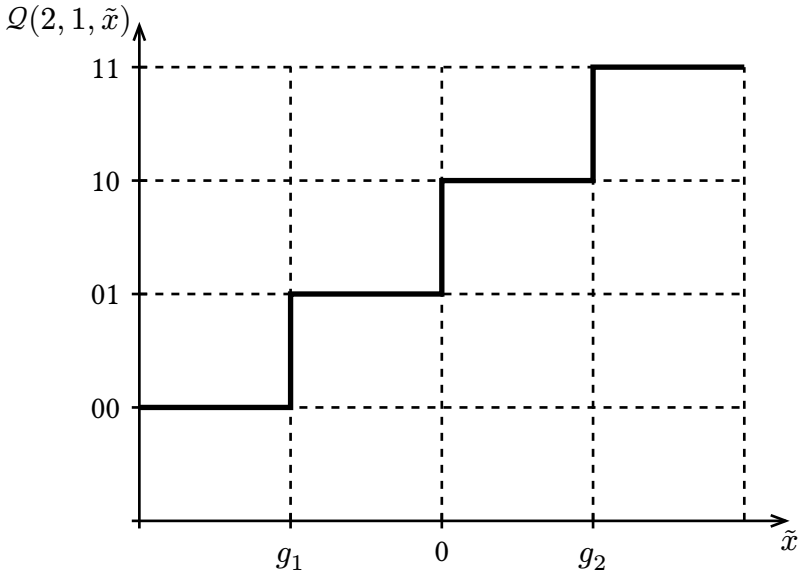


Abbildung 3: 2-bit Quantisierer Funktion

Anstelle einer Quantisierung basierend auf dem Vorzeichen des Eingangswertes, wie in Abbildung 1 ist bei einer Quantisierung höherer Ordnung eine mehrstufige Entscheidungsfunktion mit mehreren Grenzen wie in Abbildung 3 notwendig. Es stellt sich nun die Frage, wie man die Grenzen g_1 und g_2 aus Abbildung 3 wählt, um die Optimierung des 1-bit Falles aus Abbildung 2 auf Fälle höherer Bit-Ordnung zu übertragen.

Die ersten Ansätze der Bachelorarbeit beinhalteten zunächst ein naives Raten der möglichen Grenzen für die Quantisierung basierend auf einer Schätzung der Form der resultierenden Verteilung. Zunächst wurde ein globales Optimierungsverfahren untersucht, bei dem nach einer ersten Optimierung nach der maximalen Distanz zu allen Grenzen, neue Grenzen basierend auf einer empirischen kumulativen Wahrscheinlichkeitsdichtefunktion definiert werden. Dieser Prozess wurde anschließend über mehrere Iterationen hinweg durchgeführt, um ein stabiles Ergebnis Wahrscheinlichkeitsverteilungen zu erhalten.

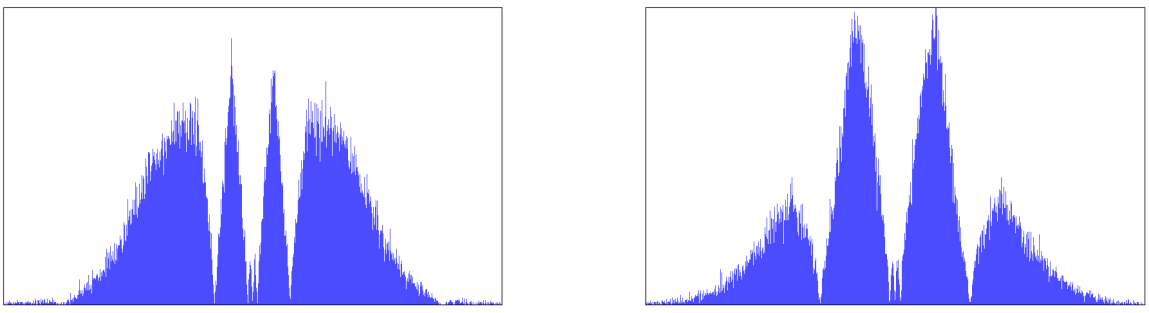

Iteration 1 Iteration 18

Abbildung 4: Wahrscheinlichkeitsverteilungen für verschiedene Iterationen

Abbildung 4 zeigt die Ergebnisse dieses iterativen Prozesses zu verschiedenen Zeitpunkten. Wegen des sehr instabilen Verhaltens der Verteilungen auch über mehrere Iterationen hinweg wurde eine zweite, konvergierende Methode untersucht.

Anstelle die Gewichtungen zu wählen, dass die resultierende Summe möglichst weit weg von allen Grenzen liegt, sollen die Summen möglichst genau die Mitten zwischen den Grenzen treffen und so implizit möglichst weit weg von den Grenzen liegen. Diese Methode hatte zwar den Vorteil, dass die hervorgehenden Verteilungen zu einer festen Verteilung konvergieren, jedoch zeigte eine spätere Analyse keine signifikante Verbesserung der Bitfehlerrate auf.

Ziel der Ingenieurspraxis ist nun, eine mögliche Lösung für das Problem der Konvergenz dieses Ansatzes zu finden und mit anderen Methoden zur Verbesserung der Bitfehlerrate zu vergleichen.

3. Konzeption und Durchführung

3.1. Literaturrecherche und Konzeption

Zunächst fand eine tiefere Einarbeitung in die existierende Literatur zu alternierenden Optimierungsverfahren statt. Aufgrund der Nähe der Themen hat sich eine Recherche zu den „k-Means Clustering“ und „Expectation-maximization (EM)“ Algorithmen angeboten. Spannende Literatur zu diesen Themen wurde von Bezdek [3] – allgemein zu alternierenden Optimierungsverfahren und spezifischer von Do [4] publiziert.

Vergleiche der Algorithmen mit dem gestellten Problem

Sowohl die hier vorgestellten Methoden, als auch die „k-Means Clustering“ und EM Algorithmen lösen ein Optimierungsproblem mittels eines iterativen Verfahrens. Der EM-Algorithmus befasst sich mit der Schätzung von Parametern in statistischen Modellen, insbesondere wenn der vorgegebene Datensatz unvollständig ist. Währenddessen zielt der „k-Means“ Algorithmus darauf ab, Daten in Cluster zu gruppieren. Besonderes bei letzterem ähnelt die Clusterbildung sehr dem hier gestellten Problem. Ein entscheidender Unterschied ist, dass bei k-Means Datenpunkte nur einer vorgegebenen Anzahl an Clustern zugewiesen werden. Eine Beschränkung, welche Datenpunkte in welches Cluster fallen, wird durch den Algorithmus nicht implementiert, wäre aber für eine Verbesserung der Eingangswerte vor der Quantisierung notwendig, da sonst die Gleichverteilung der quantisierten Symbole nicht garantiert werden kann.

Wenngleich diese Publikationen keinen direkten Weg zur Lösung der Problemstellung der Praxis bieten konnten, stellten sie ein gutes Grundverständnis für diese Art von Problem dar.

3.1.1. Festlegung der verwendeten Toolings

Aufgrund ihrer hohen Effizienz und der umfangreichen Unterstützung für funktionale Programmierung wurde für das Projekt, das die Implementierung der Algorithmen und die Simulation der Bitfehlerrate umfasst, die Programmiersprache Julia ausgewählt. Im weiteren Verlauf der Praxis wurde zusätzlich für die verbesserte Möglichkeit der Visualisierung der Ergebnisse das Pluto Framework – ein Julia-Pendant zu Jupyter Notebooks – mit einbezogen.

3.2. Durchführung und Projektdokumentation

Für eine effiziente und übersichtliche Implementierung der Algorithmen und Simulationen wurden zunächst diverse Hilfsfunktionen in Julia implementiert.

```julia
function create_linearcombinations(inputs, weights, n)
    collect(map(
        set -> begin
            LinearCombinationSet(
                collect(map(
                    weights -> begin
                            LinearCombination(weights, map(v -> [signbit(v)],
                            weights), set, sum(weights .* set))
                    end,
                    weights
                )))
        end,
        Iterators.partition(inputs, n)))
end
```

Listing 1: Generierung von allen möglichen Linearkombinationen

Listing 1 zeigt exemplarisch die Implementierung einer Funktion zur Generierung aller möglichen Linearkombinationen für eine Menge an Eingangswerten.

Anschließend wurde die betragsmäßige Optimierung, welche in Abbildung 1 und Abbildung 2 dargestellt wird, implementiert und getestet.

3.2.1. Rekursiver Ansatz

Als nächste Möglichkeit für den Multi-Bit Fall, ist ein rekursiver Ansatz des Problems implementiert worden. Hierfür werden die Eingangswerte zunächst mit der betragsmäßigen Optimierung verarbeitet um so eine „optimale“ Verteilung für den 1-bit Fall zu konstruieren. Anschließend wird die Verteilung in zwei symmetrische Unterverteilungen aufgeteilt, und jeweils deren Mittelwert bestimmt. Daraufhin werden für jede Summe der jeweiligen Unterverteilungen zusätzliche fraktionierte Gewichtungen auf die bereits bestehenden Gewichtungen aufaddiert. Aufbauend auf die Mittelwertbestimmung werden zusätzliche Grenzen definiert, anhand deren die aufkommenden neuen Summen mit fraktionierten Gewichtungen optimal gewählt werden. Basierend auf der Anzahl m an Bits die aus einer Summe extrahiert werden sollen, wird dieses Verfahren m-Mal mit allen entstehenden Unterverteilungen durchgeführt.

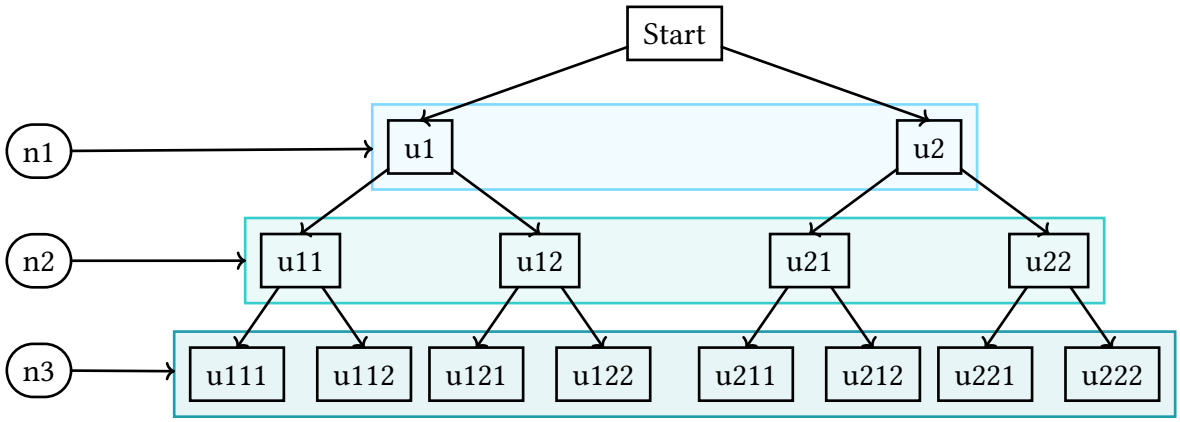


Abbildung 5: Darstellung des rekursiven Algorithmus

Abbildung 5 zeigt das grundsätzliche Schema für den Rekursiven Algorithmus auf. Zu Beginn werden die anfänglichen Eingangswerte nach der Methode zur Betragsoptimierung verarbeitet. Anschließend wird die Verteilung in zwei symmetrische Unterverteilungen aufgeteilt. Für jede neue Unterverteilung jeweils eine neue Quantisierergrenze über den Median dieser Unterverteilung definiert. **n1** beschreibt das Skalar, welches in allen möglichen Kombinationen auf die Helperdaten der Linearkombinationen von **u1** addiert bzw. subtrahiert wird. Aus den neu gefundenen möglichen Linearkombinationen wird nun diejenige gewählt, welche möglichst weit weg von den neu definierten Grenzen liegt. Dieses Verfahren wird nun iterativ auf jeder weitere Unterverteilung angewendet bis die Anzahl der Verteilungen $\log_2(m)$ entspricht, wobei m die Anzahl der zu quantisierenden Bits definiert.

Ein erstes positives Ergebnis hier war die schnelle Konvergenz der Verteilung und die Gleichverteilung der quantisierten Symbole, da in jeden Grenzbereich möglichst gleich viele Summen gelegt worden sind. Abbildung 6 zeigt das Ergebnis des rekursiven Ansatzes für die Quantisierung von 2 Bit.

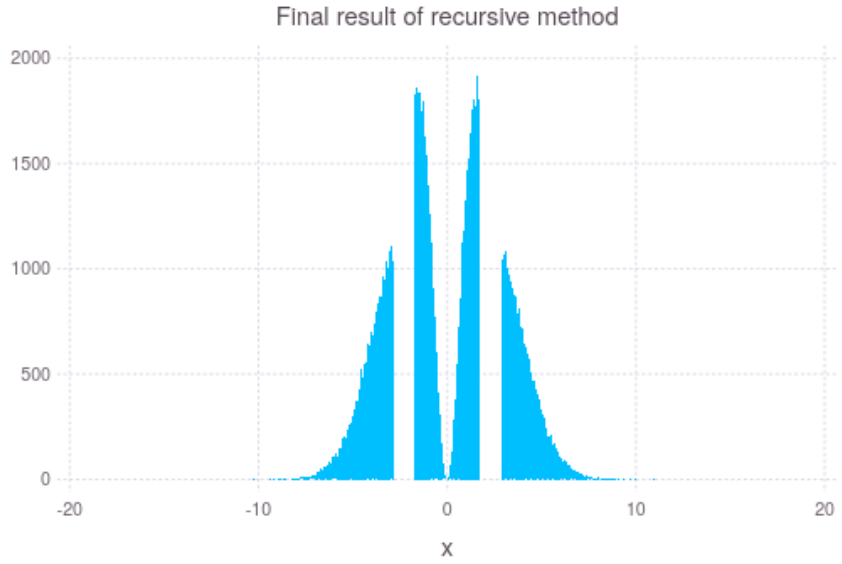


Abbildung 6: Verteilung der Eingangswerte nach dem rekursiven Ansatz

Jedoch stellte sich nach der Analyse der verwendeten Helperdatenvektoren heraus, dass durch die Zuweisung der Helperdaten Informationen über den Schlüssel ableitbar sind.

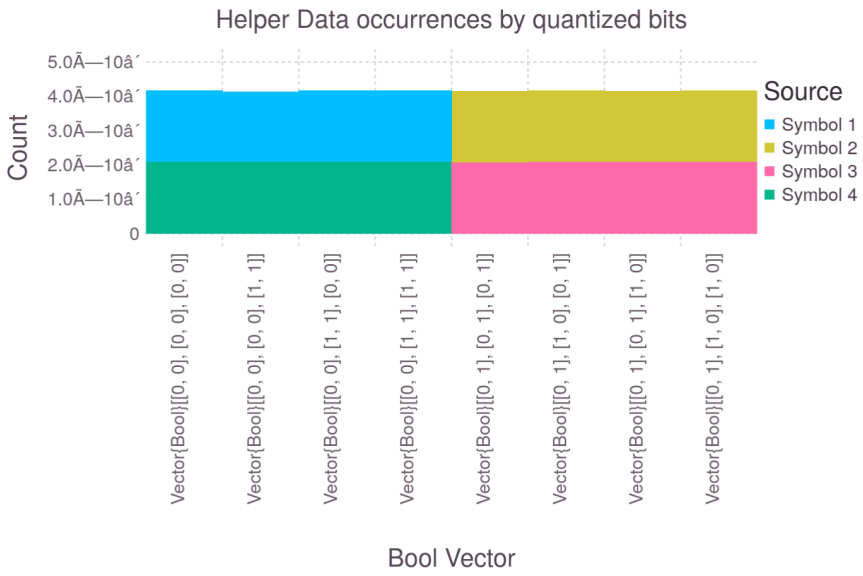


Abbildung 7: Verteilung der Helperdatenvektoren für jedes Bitsymbol

Das Histogramm in Abbildung 7 zeigt dieses Problem auf. Damit über die Helperdaten keine Informationen über den Schlüssel bekannt werden, muss jeder verwendete Helperdatenvektor von jedem Symbol gleich häufig verwendet werden. Mit diesem Ansatz werden von je zwei Symbolen jedoch nur vier von acht möglichen Helperdatenvektoren verwendet.

3.2.2. Vorgabe des Codeworts

Eine weitere getestete Methode bestand aus dem vorgeben des zu verwendeten Codewords bzw. Schlüssels. Hierfür werden die Grenzen der Quantisiererfunktion über die kumulative Verteilungsfunktion der Eingangswerte bestimmt. Anschließend wird jene Summe mit Helperdaten gewählt, welche die Summe zu ihrem vorgegebenen Codewort quantisieren.

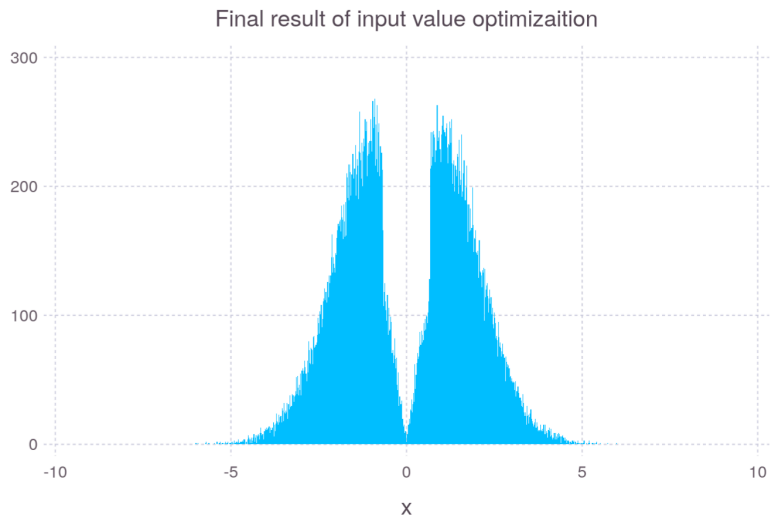


Abbildung 8: Verteilung nach der Verarbeitung mit vorgegebenem Codewort

Leider stellte sich die Vorgabe, jene Linearkombination zu wählen welche am besten ein vorgegebenes Codewort approximiert nicht als praktikabel heraus, da – wie in Abbildung 8 zu sehen ist – bei einer Verarbeitung für 2 Bit keine vier voneinander unterscheidbaren Unterverteilungen ergeben. Nahe der 0 lässt sich lediglich eine kleine Abweichung vom 1-bit Fall feststellen, welche aber nicht signifikant genug ist, um die Bitfehlerrate zu minimieren.

3.2.3. Brute-Force-Ansatz

Als letzten möglichen Lösungsansatz wurde ein Brute-Force-Ansatz untersucht. Um die optimalen Grenzen für eine Quantisierung höherer Ordnung zu erhalten, wurden für verschiedene Quantisierergrenzen die Verteilungen der Quantisierten Codewörter analysiert. Im Detail wurde für eine große Menge an möglichen Grenzen die Distanzmaximierung der Linearkombinationen durchgeführt. Direkt im Anschluss wurde über Pearson's Chi-square Test die Gleichverteilung der Quantisierten Symbole überprüft und nach einem Maximum des Ergebnisses des Tests gesucht. Abbildung 9 zeigt das Ergebnis der Verarbeitung dieser Grenzen für einen 3-bit Fall.

Da diese Brute-Force Operation sehr rechenaufwendig ist, wurden die bereits in Julia implementierten Lösungen für parallel Computing eingesetzt und die Berechnung der idealen Grenzen auf einem Computer mit hoher Rechenkapazität ausgelagert.

Damit das parallele Rechnen eine signifikante Verbesserung in der Rechengeschwindigkeit erzielt, gab es einige Punkte zu beachten:

- Für einen festgelegten Datensatz ändern sich die möglichen gewichteten Summen nicht während der Ausführung des Algorithmus', also können diese vorab berechnet und gespeichert werden.
- Das Verwenden der „pmap" Funktion zur parallelen Ausführung des Optimierungsalgorithmus hat den Rechenprozess um ca. 700ms verlangsamt.
  Eine effizientere Lösung besteht darin, bei der Ausführung des Julia Skripts die Anzahl an Threads vorzudefinieren und die jeweiligen Funktionen mit dem „@everywhere" Flag zu markieren, damit sie von den verschiedenen Threads aufgerufen werden können.

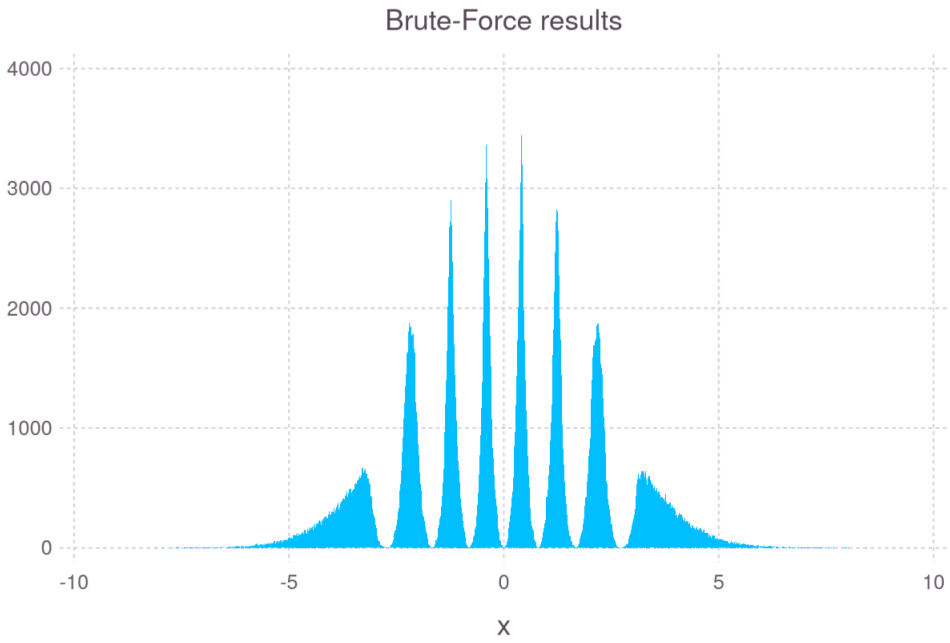


Abbildung 9: Resultat nach Verwendung der durch den Brute-Force Ansatz gefundenen Grenzen

Auch die Betrachtung des Histogramms der Verteilung der Helperdaten zeigt befriedigende Ergebnisse auf. Wie in Abbildung 10 zu sehen ist, wird jeder Helperdatenvektor von jedem Symbol gleich häufig verwendet.

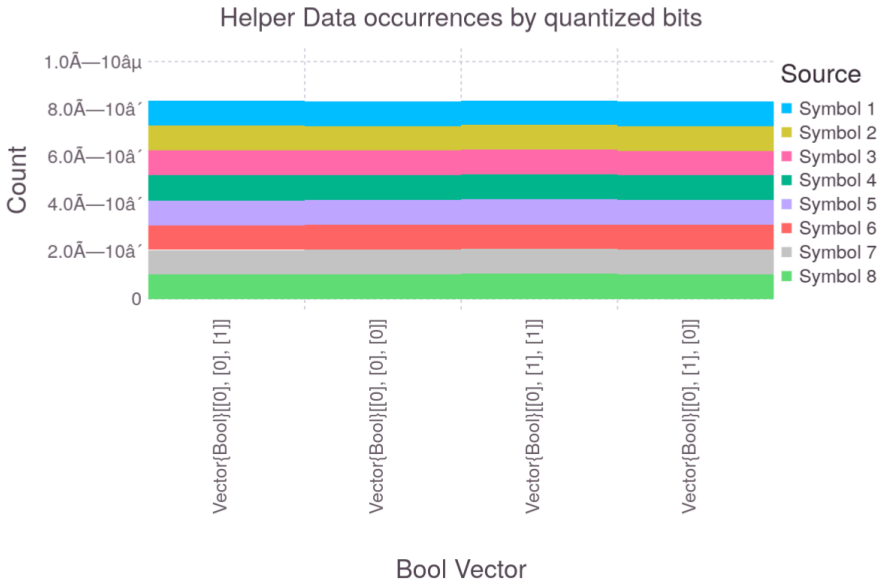


Abbildung 10: Resultat nach Verwendung der durch den Brute-Force Ansatz gefundenen Grenzen

Außerdem ist über diese Methode eine signifikante Verbesserung de Bitfehlerrate im Bezug auf eine Quantisierung ohne Vorverarbeitung und Fehlerkorrekturcode vorzuweisen, weshalb sich hier gute Chancen auf ein zukünftiges Nutzbares Verfahren abbilden.

3.2.4. Portieren der BCH-Code Python Bibliothek

Für die weitere Integration und vollständige Implementierung zu einem allgemeinen Simulationsprogramm, das die gesamte Fehlerkorrektur bis zum Schlüssel abdeckt, ist ein BCH-Fehlerkorrektur-Code als Python-Codebasis vorgegeben worden. Damit dieser auf die bereits in Julia programmierten Implementierungen angewendet werden kann, wurde diese Codebasis vollständig von Python nach Julia portiert. Hierzu waren kleinere Zwischenrecherchen im Bezug auf die Differenzen und Ähnlichkeiten der beiden Sprachen notwendig, um eine ordnungsgemäße Portierung durchführen zu können.

Aufgrund des kurz darauf folgenden Endes der Anstellung im Rahmen der Ingenieurspraxis war es leider nicht mehr möglich den portierten BCH-Code im Zusammenhang mit der gefundenen Lösung zur Optimierung der Grenzen einzusetzen.

4. Ergebnisse & Zusammenfassung

Insgesamt sind die meisten geplanten Ziele zum Erfolg der Ingenieurspraxis erreicht worden. Da mit einer der getesteten Methoden zur Optimierung der Grenzen eine reproduzierbare Lösung gefunden wurde, stellen sich auch spannende zukünftige Projekte in diesem Gebiet auf. Insgesamt könnte der Brute-Force Ansatz noch weiter optimiert werden, etwa mit einer iterativen Erhöhung der Genauigkeit der Grenzen um so auch bei sehr hohen Symbolbreiten genaue Grenzen bestimmen zu können. Da die optimalen Grenzen für den 2- und 4-bit Fall mit einer mittelwertfreien Standardnormalverteilung bestimmt worden sind, lassen sich die hier bereits gefundenen Grenzen leicht auf weitere Normalverteilungen mit anderen Parametern übertragen. Außerdem besteht die Möglichkeit des Zusammenstellens einer Datenbank mit den numerisch optimalen Grenzen für dieses Problem für verschiedene Symbolbreiten. Obwohl letztendlich keine vollständige Integration des portierten BCH-Codes stattgefunden hat, bietet die nun portierte Bibliothek eine solide Grundlage für zukünftige Integration auch in anderen Projekten.

Abkürzungsverzeichnis

***PUF* – physical unclonable function** 4, 5, 5

***SMHD* – S-Metric Helper Data method** 5, 5

***TMHD* – Two Metric Helper Data method** 5, 5

Bibliographie

[1] J.-L. Danger, S. Guilley, und A. Schaub, „Two-metric helper data for highly robust and secure delay PUFs“, in *2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI)*, 2019, S. 184–188.

[2] R. F. Fischer, „Helper Data Schemes for Coded Modulation and Shaping in Physical Unclonable Functions“, *arXiv preprint arXiv:2402.18980*, 2024.

[3] J. C. Bezdek und R. J. Hathaway, „Some Notes on Alternating Optimization“, in *Advances in Soft Computing — AFSS 2002*, N. R. Pal und M. Sugeno, Hrsg., Berlin, Heidelberg: Springer, 2002, S. 288–300. doi: 10.1007/3-540-45631-7_39.

[4] C. B. Do und S. Batzoglou, „What is the expectation maximization algorithm?“, *Nature Biotechnology*, Bd. 26, Nr. 8, S. 897–899, Aug. 2008, doi: 10.1038/nbt1406.