FREIE UNIVERSITÄT
BERLIN

Improved Linear Cryptanalysis on Quantum Computers

Master-Arbeit
zur Erlangung des Grades Master of Science
im Studiengang Informatik

Vorgelegt am 3. November 2023 von

Hannes Hattenbach

Erstgutachter	Zweitgutachter
Prof.Dr.Marian Margraf	

Betreuer: Prof.Dr.Marian Margraf

Fachbereich Mathematik und Informatik

Eidesstattliche Erklärung

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel[1] angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht.

[illegible]

Digitally signed by
Hannes Hattenbach
Date: 2023.11.03
15:58:04 +01'00'

Hannes Hattenbach, Berlin, den 3. November 2023

[1]Einige Sätze und/oder Quellcode wurden mithilfe von GitHub Copilot vervollständigt. Außerdem wurden einige Wörter/Sätze mithilfe von DeepL Übersetzter zwischen dem Deutschen und dem Englischen übersetzt.

Contents

Abstract

This thesis investigates the intersection of linear cryptanalysis and quantum computing in the context of symmetric cryptographic schemes. Focusing on the Malviya algorithm[37], which uses quantum computing to find linear approximations to vectorial Boolean functions, the study examines methods to improve their success probabilities through quantum techniques like amplitude amplification.

The research explores the limitations of the algorithm in handling highly nonlinear functions and introduces measures to gauge function linearity, establishing a link to the corresponding success probabilities. By analyzing various enhancements through quantum methods, this work aims to improve success rates and compares them against classical approaches.

Overall, this thesis offers a comprehensive exploration of leveraging quantum computing for enhancing success probabilities in linear cryptographic analysis, providing insights into the strengths and limitations of quantum-based linear cryptanalysis.

Abstract (German)

Diese Arbeit untersucht die Kombination von linearer Kryptoanalyse und Quantencomputing im Zusammenhang symmetrischen kryptographischen Verfahren. Die Studie konzentriert sich auf den Malviya-Algorithmus [37], welcher Quantencomputing verwendet, um lineare Approximationen für vektorielle boolesche Funktionen zu finden, und untersucht Methoden zur Verbesserung ihrer Erfolgswahrscheinlichkeiten durch Quantentechniken wie Amplitudenverstärkung.

Die Forschungsarbeit untersucht die Grenzen des Algorithmus bei der Handhabung hochgradig nichtlinearer Funktionen und führt Maße ein, um die Linearität der Funktionen abzuschätzen und eine Verbindung zu den entsprechenden Erfolgswahrscheinlichkeiten herzustellen. Durch die Analyse verschiedener Verbesserungen durch Quantenmethoden zielt diese Arbeit darauf ab, die Erfolgsraten des Malviya-Algorithmus zu verbessern, und vergleicht sie mit klassischen Ansätzen.

Insgesamt bietet diese Arbeit eine umfassende Untersuchung des Einsatzes von Quantencomputern zur Verbesserung der Erfolgswahrscheinlichkeiten in der linearen kryptographischen Analyse und gibt Einblicke in die Stärken und Grenzen der quantenbasierten linearen Kryptoanalyse.

[1]Both abstracts were shortened or translated with the help of LLMs.

Chapter 1

Introduction

1.1 Motivation

One of the core concepts of information security, namely data confidentiality, is either solved by stenography, i.e. hiding the data or by using cryptography to encrypt the data. The latter can essentially be divided into two categories, symmetric and asymmetric cryptography. While the security-proofs for most schemes in either category rely on mathematical assumptions that are thought to be true but are not proven, asymmetric schemes have the additional challenge of keeping the private key irretrievable even with access to the closely related public key. It therefor has to rely on arguably more complex mathematical problems than symmetric cryptography. One such problem, which is well established and not been broken by classical computing yet, is the hidden subgroup problem. The common asymmetric cryptography schemes are therefor based on some variation of the hidden subgroup problem, e.g. the discrete logarithm, the elliptic curve discrete logarithm or prime factorization problems. These would specifically be RSA (prime factorization) or ECDSA (elliptic curve discrete logarithm). By only using classic computers these problems are practically (in polynomial time) intractable and therefor these schemes are considered safe against them, the advancement of quantum computers poses a new threat as these problems are now considered to be solvable in quantum polynomial time[1]. This is mainly due to the fact that the quantum Fourier transform can be leveraged to solve the hidden subgroup problem for abelian groups with exponential speedup. This was researched by Shor in 1994 [47] and is now considered to be one of the most important results in quantum computing, mainly for aforementioned reasons.

With the security of the most asymmetric cryptography schemes put at risk by quantum computing, the security of symmetric cryptography schemes is still considered to be safe with the currently best generic approach being circumvented by doubling the key and block sizes. This is caused by the quadratic speedup through Grover's algorithm [26], but more on that in section 5.1.

This sub-exponential speedup is not enough to break the security of symmetric cryptography schemes per se, so other approaches are needed. One of these approaches in classical computing was pioneered by Matsui to break the then standard DES encryption scheme [39]. His approach is now known as linear cryptanalysis and is based on the fact that most vectorial Boolean functions can be approximated to some degree by a linear function. For every linear approximation we gain one bit of information, bisecting the search space for following brute-forcing possible used cipher-texts and or keys. The challenge therein is finding the linear approximations themselves (compare problem [39, P1]), which Matsui did by examining the inner structure of the DES encryption scheme and combining linearities in their S-Boxes. This approach is not necessarily applicable to other encryption schemes, as they are not generally based on S-Boxes. But it is still important for cryptanalysis to research the linearities of cryptographic functions as those would be a weak point in the security of the scheme.

Finding those global approximations however is a computationally hard problem, as the number of

[1]See appendix C.2

possible linear approximations grows exponentially with the number of bits. But with the advent of larger quantum computers there might be a way to use quantum algorithms to solve this problem. One such approach is the one by Malviya and Tiwari [37], which we will discuss in more detail in this thesis. Their solution runs with a constant amount of quantum queries, which is a huge improvement over the classical approach of checking every possible pair of input and output for the function under analysis.

However, their approach seems to have a very bad success probability, so the main contribution of this thesis is to model the success probabilities of their algorithm, improve it using amplitude amplification and comparing it to the classical approach.

1.2 Outline

After presenting some related work, this thesis first introduces the basics of linear cryptanalysis and quantum computing in section 3.3 and section 3.4, respectively as those are the main fields this thesis is based on and are therefor needed as preliminaries to understand the rest of the thesis. These two fields get combined in a single algorithm by Malviya and Tiwari, which leverages quantum computing to find linear approximations of vectorial Boolean functions and is therefor the main starting point of this thesis, this will be discussed and analyzed in section 4.2.

The next part is to define some new measures in section 4.3.2 of how linearly approximatible a function is to then relate this to the success probabilities of applying Malviya algorithm on that function in section 4.3. As it turns out while modeling the success probabilities, their algorithm does not succeed very likely if the function is very non-linear. We will therefor improve these probabilities by using multiple variants of amplitude amplification in chapter 5. The resulting runtimes and probabilities will finally be compared against each other and the classical approach in chapter 6.

Chapter 2

Related Work

The most prominent related work, on which this thesis also builds upon, is the already mentioned paper by Malviya et al. [37] which proposes ideas quite similar to some we had, but already published. These ideas consist primarily of a novel quantum algorithm describing how to find linearities in vectorial Boolean functions. This algorithm will be discussed in great detail in section 4.2 so I will not go into the particulars here. Their paper also already compares to a very trivial classical algorithm but leaves some unclarities. In addition, they have already run their algorithm on actual 8-qubit quantum hardware but measured mostly useless results. This had two reasons, first they used an oracle that was larger than the algorithm itself, result in a very high error rate due to noise, and second the algorithm itself has a low success probability. They still conclude that their algorithm solves finding linearities efficiently with exponential speedup, but factoring in the success probabilities it gets more complicated, this will be discussed in this thesis.

Their paper in turn builds upon the Bernstein-Vazirani Algorithm [12] which, given the induced oracle U_f (compare section 3.4.1.1) of a linear Boolean function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$, i.e. $f(x) := \langle a|x\rangle$ for some fixed $a \in \mathbb{F}_2^n$, finds a in $\mathcal{O}(1)$ oracle calls using the circuit shown in fig. 2.1. In some sense this algorithm is a special case of the algorithm by Malviya et al. as it finds the one correct linear “approximation” to f and uses a similar circuit. If f would not be linear this algorithm would measure an a with a probability corresponding to the “goodness” of the approximation $f(x) \approx \langle a|x\rangle$, which is exactly what the algorithm by Malviya et al. does, which, again, will be analyzed in section 4.2, therefor further analysis of the Bernstein-Vazirani Algorithm is not necessary for this thesis.

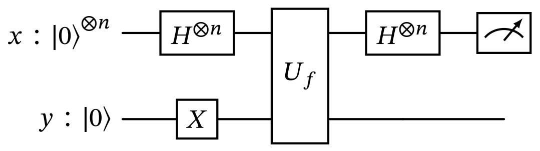


Figure 2.1: Circuit of the Bernstein-Vazirani Algorithm

Another paper, which also builds upon the Bernstein-Vazirani Algorithm is the paper by Li and Yang [34]. In contrast to this thesis, it does not find linear approximations but rather linear structures, which might be closer related to differential cryptanalysis. A linear structure is defined as follows:

Definition 2.0.1 (Linear Structure[34]). *Let $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$, then $a \in \mathbb{F}_2$ is called a* linear structure *of f iff*

$$f(x \oplus a) \oplus f(x) = f(a) \oplus f(0).$$

They provide an algorithm that finds such linear structures and relate its running time to a property of f, called the *relative differential uniformity*. A similar approach will be used in this thesis with the *non-linearity* of f.

Finding a linear structure of f could also be used to half the search space for linear approximations as we could disregard every (second) $x' := x \oplus a$ from our input space.

But this will not be used in this thesis as it increases the complexity of the algorithm a lot without obvious benefits to runtime or success probability.

Another fundamental paper is the introduction of linear cryptanalysis itself by Matsui [40], this has nothing to do with quantum computing yet, but as it is the basis for the whole idea of linear cryptanalysis it is still important to mention it here as some ideas and notations originate from this paper, more details on linear cryptanalysis will be given in section 3.3.

On a similar note, a fundamental quantum building block independent of linear cryptanalysis but used in this thesis, is the amplitude amplification algorithm by Brassard et al. [14] as a generalization of the Grover search algorithm [26]. It will be used to amplify the success probability of the algorithm by Malviya et al. in section 5.2.

Chapter 3

Preliminaries

3.1 Boolean Functions

Since this thesis is primarily about finding linearities in vectorial Boolean functions, it is important to define what a Boolean function is and how it can be represented.

Definition 3.1.1. *$f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is called a Boolean function, $x \in \mathbb{F}_2^n$ is a bitstring while $\mathbb{F}_2$, denoting the finite field of two elements, e.g. represented by $\mathbb{F}_2 = \{0, 1\}$, is called a single bit.*

Definition 3.1.2. *As a generalization of definition 3.1.1 $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ is called a vectorial Boolean function.*

In this thesis we will mostly be working with the idea of cryptographic Boolean functions, but the ideas can be extended to (vectorial) Boolean functions w.l.o.g. as well.

Definition 3.1.3. *Cryptographic Boolean functions, i.e. functions that take in a key $k \in \mathbb{F}_2^{n_k}$ and a plaintext $m \in \mathbb{F}_2^{n_m}$ and output a ciphertext $c \in \mathbb{F}_2^{n_c}$ can be defined as*

$$f : \mathbb{F}_2^{n_k} \times \mathbb{F}_2^{n_m} \mapsto \mathbb{F}_2^{n_c}$$

Preliminiary 3.1.4. *We could also concatenate the two input bitstrings to define our functions as $f'(k \mathbin{+\!\!\!+} m) = f(k, m)$ w.l.o.g. $k \mathbin{+\!\!\!+} m$ denotes the concatenation of the two bitstrings k and m.*

Since this makes it more obvious that key, plaintext and ciphertext can theoretically be sized independently and everything in this thesis would still apply, I will prefer the concatenated notation. But they could be used interchangeably with trivial adjustments.

One interesting measure for Boolean functions, which is later used in this thesis to determine how good a linear approximation is, is the correlation of a Boolean function.

Definition 3.1.5 ([28]). *The Correlation $c(f)$ (or $Corr(f)$) of a Boolean function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is given by how much higher its relative frequency of outputting 0 instead 1 is :*

$$c(f) = 2^{-n} \left(\#\{\xi \in \mathbb{F}_2^n \mid f(\xi) = 0\} - \#\{\xi \in \mathbb{F}_2^n \mid f(\xi) \neq 0\}\right)$$

Besides that we will also need the Walsh-transform of a Boolean function.

Definition 3.1.6 ([38]). *The Walsh-Transform[1] is given by:*

$$\hat{\chi}_f : \mathbb{F}_2^n \to \mathbb{Z}; \eta \mapsto \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \eta, x \rangle \oplus f(x)}$$

[1]Some references use the Fourier transform. The Walsh transform of f is the same as the Fourier transform of $(-1)^f$, i.e. using $[-1, 1]$ instead of $[0, 1]$.

As it is used in many parts of this thesis the Walsh-transform can be extended into vectorial functions as follows:

Definition 3.1.7. *Given $f \in F(\mathbb{F}_2^{n_m} \times \mathbb{F}_2^{n_k}, \mathbb{F}_2^{n_c})$,* [2] *the vectorial Walsh-Transform is defined as:*

$$\hat{\chi}_f : \mathbb{F}_2^n \times \mathbb{F}_2^n, \mathbb{F}_2^n \to \mathbb{Z}; (\alpha, \gamma, \beta) \mapsto \sum_{p,k \in \mathbb{F}_2^m} (-1)^{\langle \alpha \mathbin{+\!\!+} \gamma | p \mathbin{+\!\!+} k \rangle \oplus \langle \beta | g(p,k) \rangle} = \hat{\chi}_{\langle \beta | f \rangle}(\alpha \mathbin{+\!\!+} \gamma).$$

If the Walsh-transform is evaluated everywhere, it is called the Walsh-spectrum of that function.

3.1.1 Linearities

Definition 3.1.8. *A Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is called linear if there exists a vector $v \in \mathbb{F}_2^n$ such that*

$$f(x) = \langle v | x \rangle = \bigoplus_{i=1}^{n} v_i x_i$$

Similarly a vectorial Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is called linear if there exists a matrix $M \in \mathbb{F}_2^{m \times n}$ such that $f(x) = Mx$. Analogously a cryptographic Boolean function $f : \mathbb{F}_2^{n_k} \times \mathbb{F}_2^{n_m} \to \mathbb{F}_2^{n_c}$ is called linear if there exists a matrix $M \in \mathbb{F}_2^{n_c \times n_k + n_m}$ such that $f(k, m) = M(k \mathbin{+\!\!+} m)$.

Definition 3.1.9. *In difference to definition 3.1.8, a linearity for a vectorial Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is a pair of vectors $(a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m)$ such that*

$$\langle f(x) | b \rangle = \langle a | x \rangle$$

3.2 Linear Approximations

Some pairs of vectors (a, b) in definition 3.1.9 are not a perfect linearity but come close to it, where the linearity holds for many inputs of f but not for all. These are called *linear approximations*.

3.2.1 Goodness of linear approximations

In this thesis we often reference the *goodness* of a linear approximation. In theory there could be multiple ways to measure this, but we will primarily use a special form of correlation for linear approximation as a measure of goodness here.

The basic definition of correlation is given in definition 3.1.5 and is used to measure how much more often a Boolean function outputs 0 instead of 1.

With the help of this definition and the following definitions we can define the correlation of a linear approximation.

Definition 3.2.1. *Let $\bar{t}_{m_x,m_y}(f, x) := \left(\langle m_x | x \rangle = \langle m_y | f(x) \rangle\right) = \left(\langle m_x | x \rangle \oplus 1 \oplus \langle m_y | f(x) \rangle\right)$ be the function that evaluates whether the linear approximation $m_x \in \mathbb{F}_2^{n_x}, m_y \in \mathbb{F}_2^{n_y}$ for $f \in F(\mathbb{F}_2^{n_x}, \mathbb{F}_2^{n_y})$ holds true at position $x \in \mathbb{F}_2^{n_x}$*

Definition 3.2.1 can be seen as the indicator function for the set of elements that relate according to the linear relation m_x, m_y for f.

Definition 3.2.2. *Also let $t_{m_x,m_y}(f) := \#\left\{x \in \mathbb{F}_2^{n_x} | \bar{t}_{m_x,m_y}(f, x) = 1\right\}$ be the number of times the linear approximation m_x, m_y for f holds true.*
Similarly, let $t^*_{m_x,m_y}(f) := \#\left\{x \in \mathbb{F}_2^{n_x} | \bar{t}_{m_x,m_y}(f, x) = 0\right\} = |\mathbb{F}_2^{n_x}| - t_{m_x,m_y}(f)$ be the number of times the linear approximation m_x, m_y for f does not hold true.

This arguably complicated definition of t and t^* is used to make the following definition of correlation/ goodness of linear approximation easier to understand.

[2]As defined via $F(\mathbb{F}_2^n, \mathbb{F}_2^m)$.

Definition 3.2.3. *The* goodness *of a linear approximation* (x, y) *with* $x \in \mathbb{F}_2^{n_x}, y \in \mathbb{F}_2^{n_y}$ *for any* $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$ *is given by*

$$c(\bar{t}_{x,y}(f)) = \frac{t^*_{x,y}(f) - t_{x,y}(f)}{2^{n_x}}.$$

This aquivalance follows from definition 3.1.5 and definition 3.2.2.

One interesting remark to notice is, that in case of non-vectorial Boolean functions this can simply be mapped to the Walsh transformed of the function as defined in definition 3.1.6 at the point m_x:

Corollary 3.2.3.1. *Let* $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$

$$\forall m_x \in \mathbb{F}_2^n : c(\bar{t}_{m_x,1}(f)) = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^{n_x}} (-1)^{\langle m_x | x \rangle \oplus f(x)} = \frac{1}{2^n} \hat{\chi}_f(m_x)$$

An analogous statement can be made for vectorial Boolean functions, with the help of definition 3.1.7.

In addition to linear approximations we might define *affine approximations* as

$$\langle f(x) | b \rangle \approx \langle a | x \rangle \oplus c \qquad \text{for some } c \in \mathbb{F}_2 \tag{3.1}$$

which would flip the sign of the correlation for $c = 1$, but otherwise would not make much of a difference.

3.2.2 Linear Relations exist in Every Boolean Function

As this thesis is about finding linear approximations for Boolean functions, it is important to show that this is possible for any Boolean function, which is done in this section.

Theorem 3.2.4. *Parsevals theorem states, that for any* $f \in F(\mathbb{F}_2^n, \mathbb{F}_2)$ *[38, p. 60]:*

$$\sum_{\eta \in \mathbb{F}_2^n} \hat{\chi}_f(\eta)^2 = 2^{2n}$$

Let p be a plaintext and k be a symmetric key.

Lemma 3.2.5. *Theorem 3.2.4 directly implies, that:*

$$\forall f \in F(\mathbb{F}_2^n, \mathbb{F}_2) \exists \eta \in \mathbb{F}_2^n : \hat{\chi}_f(\eta) \neq 0$$

Theorem 3.2.6. *Every cryptographic function has non-zero linear approximation with a non-zero correlation or bias, i.e.:*

$$\forall g \in F(\mathbb{F}_2^m \times \mathbb{F}_2^m, \mathbb{F}_2^m) \exists \alpha, \beta, \gamma \in \mathbb{F}_2^m : Corr_{\alpha + \gamma, \beta}(f) \neq 0, \neg (\alpha = \beta = \gamma = 0).$$

Note, that $n_m = n_k = n_c = m$. *This is only for readability and can be generalized to any* n_m, n_k, n_c.

Proof. Similarly to corollary 3.2.3.1 the following holds in accordance with definition 3.1.7:

$$\mathrm{Corr}_{\alpha + \gamma, \beta}(f) = \sum_{p,k \in \mathbb{F}_2^m} (-1)^{\langle \alpha | p \rangle \oplus \langle \gamma | k \rangle \oplus \langle \beta | g(p,k) \rangle} = \hat{\chi}_g(\alpha + \gamma, \beta).$$

The left-hand side of the equation is trivial. The right-hand side is the result of the following steps:

1. Starting with the Walsh-transformation (as given in definition 3.1.6) we choose $x := p + k$ ($\implies m := n/2$) and $f(x) := \langle \beta | g(x) \rangle = \langle \beta | g(p, k) \rangle$

$$\hat{\chi}_f : \mathbb{F}_2^{2m} \to \mathbb{Z}; \eta \mapsto \sum_{p,k \in \mathbb{F}_2^m} (-1)^{\langle \eta | p + k \rangle \oplus \langle \beta | g(p,k) \rangle}$$

2. Splitting η into its first and second m bits ($\eta := \alpha + \gamma$) we get:

$$\hat{\chi}_f : \mathbb{F}_2^m \times \mathbb{F}_2^m \to \mathbb{Z}; \alpha, \gamma \mapsto \sum_{p,k \in \mathbb{F}_2^m} (-1)^{\langle \alpha + \gamma | p + k \rangle \oplus \langle \beta | g(p,k) \rangle}$$

3. Since α, p, γ, k all have the same length and therefor p only maps on α and k only on γ:

$$\hat{\chi}_f : \mathbb{F}_2^m \times \mathbb{F}_2^m \to \mathbb{Z}; \alpha, \gamma \mapsto \sum_{p,k \in \mathbb{F}_2^m} (-1)^{\langle \alpha | p \rangle \oplus \langle \gamma | k \rangle \oplus \langle \beta | g(p,k) \rangle}$$

As a result of that and since we only chose a specific f defined via g and β and renamed x, theorem 3.2.4 still applies and can now be written as:

$$\forall g \in F(\mathbb{F}_2^m \times \mathbb{F}_2^m, \mathbb{F}_2^m), \forall \beta \in \mathbb{F}_2^m : \sum_{\alpha,\gamma \in \mathbb{F}_2^m} \hat{\chi}_f(\alpha + \gamma)^2 = 2^{4m} \tag{3.2}$$

Now applying the same procedure as done in lemma 3.2.5 it follows:

$$\begin{aligned} &\forall g \in F(\mathbb{F}_2^m \times \mathbb{F}_2^m, \mathbb{F}_2^m), \forall \beta \in \mathbb{F}_2^m \exists \alpha, \gamma \in \mathbb{F}_2^m : \chi_f(\alpha + \gamma) \neq 0 \\ \iff &\forall g \in F(\mathbb{F}_2^m \times \mathbb{F}_2^m, \mathbb{F}_2^m), \forall \beta \in \mathbb{F}_2^m \exists \alpha, \gamma \in \mathbb{F}_2^m : \sum_{p,k \in \mathbb{F}_2^m} (-1)^{\langle \alpha | p \rangle \oplus \langle \gamma | k \rangle \oplus \langle \beta | g(p+k) \rangle} \neq 0 \end{aligned}$$

We can now choose any $\beta \neq 0$:

$$\forall g \in F(\mathbb{F}_2^m \times \mathbb{F}_2^m, \mathbb{F}_2^m) \exists \alpha, \beta, \gamma \in \mathbb{F}_2^m : \sum_{p,k \in \mathbb{F}_2^m} (-1)^{\langle \alpha | p \rangle \oplus \langle \gamma | k \rangle \oplus \langle \beta | g(p,k) \rangle} \neq 0, \beta \neq 0$$

□

Although this proofs theorem 3.2.6 it is not very informative on what happens with different output masks β.

Another result of Parsevals theorem shows the following:

Theorem 3.2.7. *$\forall f \in F(\mathbb{F}_2^m \times \mathbb{F}_2^m, \mathbb{F}_2^m)$ it holds, that*

$$\sum_{\alpha,\beta,\gamma \in \mathbb{F}_2^n} \hat{\chi}_f(\alpha + \gamma, \beta)^2 = 2^{5n}.$$

Proof. Direct result of eq. (3.2):

$$\forall \beta \in \mathbb{F}_2^m : \sum_{\alpha,\gamma \in \mathbb{F}_2^m} \hat{\chi}_{\langle \beta | g \rangle}(\alpha + \gamma)^2 = 2^{4m} \Rightarrow \sum_{\alpha,\beta,\gamma \in \mathbb{F}_2^m} \hat{\chi}_{\langle \beta | g \rangle}(\alpha + \gamma)^2 = 2^{5m}.$$

Another alternative proof follows. It relies on the following lemma:

Lemma 3.2.8. *$\forall f \in F(\mathbb{F}_2^n, \mathbb{F}_2)$: f is identically distributed, i.e.*

$$\forall f \in F(\mathbb{F}_2^n, \mathbb{F}_2), \alpha \in \mathbb{F}_2^n : \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x \rangle} = \begin{cases} 0, & if\ \alpha \neq 0 \\ 2^n, & else. \end{cases}$$

Proof. The case $\alpha = 0$ is trivial: $\sum_{x \in \mathbb{F}_2^n} (-1)^0 = 2^n$.
As $\forall \alpha \neq 0\ \exists s : \langle \alpha | s \rangle \neq 0$, for this s it holds that $\forall x : \langle \alpha | x \rangle = 0 \Leftrightarrow \langle \alpha | x \oplus s \rangle = 1$.
As the mapping $x \mapsto x \oplus s$ is a bijection, there are equally many scalar products evaluating to 0 as to 1 and therefor it follows that $\sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x \rangle} = 0$. □

Now, in accordance with definition 3.1.7, we can write:

$$
\begin{aligned}
\sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n} \hat{\chi}_{\langle\beta|f\rangle}(\alpha \mathbin{+\!\!\!+} \gamma)^2 &= \sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n} \left(\sum_{m,k\in\mathbb{F}_2^n} (-1)^{\langle\alpha|m\rangle\oplus\langle\beta|f(m,k)\rangle\oplus\langle\gamma|k\rangle} \right)^2 \\
&= \sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n} \left(\sum_{m_1,k_1\in\mathbb{F}_2^n} (-1)^{\langle\alpha|m\rangle_1\oplus\langle\beta|f(m_1,k_1)\rangle\oplus\langle\gamma|k\rangle_1} \cdot \sum_{m_2,k_2\in\mathbb{F}_2^n} (-1)^{\langle\alpha|m\rangle_2\oplus\langle\beta|f(m_2,k_2)\rangle\oplus\langle\gamma|k\rangle_2} \right) \\
&= \sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n} \left(\sum_{m_1,k_1,m_2,k_2\in\mathbb{F}_2^n} (-1)^{\langle\alpha,m_1\oplus_n m_2\rangle\oplus\langle\beta,f(m_1,k_1)\oplus f(m_2,k_2)\rangle\oplus\langle\gamma,k_1\oplus k_2\rangle} \right) \\
&= \sum_{m_1,k_1,m_2,k_2\in\mathbb{F}_2^n} \left(\sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n} (-1)^{\langle\alpha,m_1\oplus_n m_2\rangle\oplus\langle\beta,f(m_1,k_1)\oplus f(m_2,k_2)\rangle\oplus\langle\gamma,k_1\oplus k_2\rangle} \right) \\
&= \sum_{m_1,k_1,m_2,k_2\in\mathbb{F}_2^n} \left(\sum_{\alpha\in\mathbb{F}_2^n} (-1)^{\langle\alpha,m_1\oplus m_2\rangle} \cdot \sum_{\beta\in\mathbb{F}_2^n} (-1)^{\langle\beta,f(m_1,k_1)\oplus_n f(m_2,k_2)\rangle} \cdot \sum_{\gamma\in\mathbb{F}_2^n} (-1)^{\langle\gamma,k_1\oplus k_2\rangle} \right)
\end{aligned}
$$

As (of lemma 3.2.8) the inner sums are all equal to zero, except for the case where $m_1 = m_2$, $f(m_1, k_1) = f(m_2, k_2)$ and $k_1 = k_2$ each, we can simplify the above to:

$$
\begin{aligned}
\sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n} \hat{\chi}_{\langle\beta|f\rangle}(\alpha \mathbin{+\!\!\!+} \gamma)^2 &= \sum_{m,k,\in\mathbb{F}_2^n} \left(\sum_{\alpha,\beta,\gamma\in\mathbb{F}_2^n} (-1)^{\langle\alpha,0\rangle\oplus\langle\beta,0\rangle\oplus\langle\gamma,0\rangle} \right) \\
&= \sum_{m,k\in\mathbb{F}_2^n} (2^n \cdot 2^n \cdot 2^n) \\
&= 2^{5n}
\end{aligned}
$$

□

Combining theorem 3.2.7 with eq. (3.2) we can see that no matter how choose the output mask β, there will always be a linear approximation with a non-zero bias, and its even rather irrelevant which one we choose.

3.3 Linear Cryptanalysis

As shown in theorem 3.2.6 there are always linear relations in any Boolean function. Linear cryptanalysis is a method to find and exploit these linearities in cryptographic Boolean functions, also called ciphers.

As the name suggests it is a type of cryptanalysis, i.e. a (mathematical) method to try and break a cryptographic system. In this case by analyzing the linear properties of the cipher. The main problem is to find these linearities in otherwise non-linear functions, i.e. solve problem 3.3.1.

Problem 3.3.1. *Given a function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ find a linear approximation $(a, b) \neq 0$ such that $|c(\bar{t}_{a,b}(f))|$ is maximal or at least large.*

First introduced by Matsui in 1993 [40] it was used to break the at its time state-of-the-art Data Encryption Standard (DES)[39].

DES is a symmetric block cipher, that primarily uses a substitution permutation network (SPN), i.e. small blocks of permutation and substitution operations connected together. The primary use of the permutation operations is to change the order of bits during encryption such that every input bit influences as many different output bits as possible. The substitution operations (from now on S-boxes) on the other hand do the actual scrambling by changing a sequence of input bits into a complete different output bit sequence, e.g. through the help of look up tables.

These S-boxes introduce the non-linearity in the cipher and are therefor essential for confidentiality as it provides the resistance against linear cryptanalysis. Matsui's exploit is based on some linear approximatibility in those S-boxes. In particular the 5th S-box of the DES implementation has approximations with high correlation and is therefor a good starting point for linear cryptanalysis.

The exploit itself is more sophisticated, since it needs to incorporate the permutations as well as multiple rounds and going through the whole SPN. But at its core it is based on finding good linear approximations of these S-boxes. This is possible since the S-boxes of DES only have 6 input bits and 4 output bits. So to find which linear approximations have large enough correlation he simply calculated the correlation of every possible approximation ($a \in \mathbb{F}_2^6, b \in \mathbb{F}_2^4$):

$$c_{(a,b)}(\text{DES-SBOX-5}) \tag{3.3}$$

As this only required 2^{6+4} tries this was easily possible even at his time. Some of these correlations were high enough to allow tracing the connections of the S-Box through the whole cipher and extracting one global linear approximation $\langle\alpha|m\rangle \oplus \langle\gamma|k\rangle \oplus \langle\beta|c\rangle \approx b$ [3] with a high correlation.

Using enough such approximations, each giving up to a bit of information about the key, he was able to extract enough information about the key, to then use an exhaustive search (brute-force) to find the rest of the key.

Though Matsui needed about 2^{43} Plaintext-Ciphertext pairs for his Chosen Plaintext Attack (CPA) on DES, and was therefor impractical at the time, it was essentially his work, linear cryptanalysis, that made DES obsolete. As DES is now easily breakable by a computer in a matter of hours, it was also replaced by the Advanced Encryption Standard (AES) in 2001.

As opposed to DES, AES was designed with Matsui's linear attack in mind and is therefore not as easily breakable by linear cryptanalysis. That is, not by his approach, but maybe it still has some cross-substitution-box linearities that span over the whole cipher or larger parts of it.

As the whole cipher has many more input and output bits, the number of possible linear approximations is much larger, so it is not feasible to calculate all of them with a classical computer, like Matsui did with the S-boxes. In the case of AES there are at least [4] $2^{128+128}$ possible linear approximations. More details on how linearities would be found the classical way can be found in section 4.1. But as this is not feasible, we need to find a way to do this with a quantum computer.

3.4 Quantum Computing

While checking all these possible linear approximations is unfeasible in a classical manner, there might be some speedup possible using quantum computers. Therefor quantum computing will be briefly introduced in this section, but for a more thorough introduction I would recommend the book by Kaye et al. [33], Nielsen et al. [42] or the lecture by Homeister [31] from which this section is mostly derived.

In this thesis the gate based quantum computing model is used. It is similar to the classical circuit model, but with some restrictions and some additional features. The main difference, causing its superiority over the classical model, is its most basic unit of computation, the qubit instead of the bit.

Definition 3.4.1. *A qubit is a quantum mechanical system $|\phi\rangle \in \mathbb{C}^2$ with two distinguishable basis states, usually denoted as $|0\rangle$ and $|1\rangle$. $|\phi\rangle$ can be any linear combination of those basis states, i.e. $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with $\alpha, \beta \in \mathbb{C}$. This is called a superposition of the states $|0\rangle$ and $|1\rangle$, and α and β are called the amplitudes of the basis states $|0\rangle$ and $|1\rangle$ respectively. The probability of measuring $|\Psi\rangle$ in the state $|0\rangle$ is $|\alpha|^2$ and in the state $|1\rangle$ is $|\beta|^2$ according to the born-rule. The sum of those probabilities is always 1, therefor $|\alpha|^2+|\beta|^2 = 1$.*

Similarly to how a computer could operate on data with more than two states (e.g. ternary computers), the generalization of qubits is called a qudit, it is a quantum mechanical system with d distinguishable

[3]This is another notation for eq. 1 in [39].

[4]AES has multiple variations with different key-sizes, but the block size (input and output length) is 128 bits

basis states[5], usually denoted as $|0\rangle, \dots, |d-1\rangle$. A special case of a qudit is when $d = 2^n$ is a power of two, in which case it is called a qubit- or quantum-register of length n.

The basis states of a qubit-register of length n are the 2^n states $|0\rangle_n, \dots, |2^n - 1\rangle_n$, where $|i\rangle_n$ is the binary representation of i. E.g. for $n = 2$ the basis states are $|0\rangle_2 = |00\rangle$, $|1\rangle_2 = |01\rangle$, $|2\rangle_2 = |10\rangle$ and $|3\rangle_2 = |11\rangle$. If the qubits are not entangled with each other, the state of the whole register can be described as a tensor product of the individual qubits, i.e. $|\Psi\rangle = |\Psi_0\rangle \otimes |\Psi_1\rangle \otimes \dots \otimes |\Psi_{n-1}\rangle$. This is called a product state. E.g. for $n = 2$ the state $|\Psi\rangle = |0\rangle \otimes (\alpha |0\rangle + \beta |1\rangle)$ is a product state, but $|\Psi\rangle = \alpha |00\rangle + \beta |11\rangle$ is an entangled state.

A quantum algorithm using the gate model can be described as a sequence of unitary operations, called gates. A gate is a unitary operation that can be applied to (a part of) a quantum register. A gate can also be applied to a single qubit, in which case it is called a single-qubit-gate.

Definition 3.4.2. *A unitary is a linear operator $U : \mathbb{C}^d \to \mathbb{C}^d$ that preserves the inner product, i.e. $\langle U\Psi | U\Phi \rangle = \langle \Psi | \Phi \rangle$ for all $|\Psi\rangle, |\Phi\rangle \in \mathbb{C}^d$. Equivalently, a unitary is a matrix $U \in \mathbb{C}^{d\times d}$ that fulfills $U^\dagger U = UU^\dagger = I$.*

3.4.1 Input and Output

Similarly to the classical Turing machine (compare appendix C.1), a quantum algorithm needs an input and produces an output. The input is usually assumed to be $|0\rangle^{\otimes n}$, where n is the number of qubits in the input register. But it can also be any other state. The output is usually assumed any state, denoted $|\Psi\rangle$.

3.4.1.1 Oracles

In addition to the input- and output- "tape", many quantum algorithms are given an oracle as a parameter. An oracle is a unitary operation that transforms a state $|\Psi\rangle$ into a state $|\Psi'\rangle$. The oracle is usually assumed to be a black box, i.e. it is not specified how exactly the state is transformed. This black box interpretation is interesting, because often the task of a quantum algorithm is to find out a certain property of an oracle. Such an oracle can then be considered as a kind of meta-parameter of the algorithm and can therefore be regarded as part of the input. Often the oracle is given as the induced quantum version of a classical function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$ as $U_f : |x, y\rangle \mapsto |x, y \oplus f(x)\rangle$ and in case of $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ sometimes also as $V_f : |x\rangle \mapsto (-1)^{f(x)} |x\rangle$.

3.4.1.2 Measurement

A quantum algorithm itself can again be interpreted as a unitary black box and passed as an oracle or subroutine to a larger algorithm. However, to obtain the output of a quantum algorithm in a classical form, it must be measured. Often quantum algorithms are designed in such a way that the resulting state is in such a superposition, that it is sufficient to obtain one of the basis states of this superposition. In this case a *measurement with respect to the standard basis* is sufficient:

$$|\Psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \xmapsto{\text{measurement}} |i\rangle \text{ with probability } |\alpha_i|^2 \text{ ; observe } i[6].$$

In some simpler quantum algorithms, even $|\alpha_i|^2 = 1$ holds for one i, i.e. the resulting state is a basis state up to a complex phase factor $e^{i\varphi}$ for $\varphi \in (-\pi, \pi]$ and the measurement results in i with certainty. Often a more complex measurement is necessary, but as its not needed in this thesis it will be omitted.

[5] A state, given in ket notation might also be called its wave function.

[6] i can be interpreted as both a bit string and a natural number. Accordingly, an index is denoted as $i \in \mathbb{F}_2^n$, which reflects the content of the individual qubits, as well as equivalently as $i \in \{0, \dots, 2^n - 1\}$, which represents the corresponding natural number represented by the bit string, or the index of the state.

What is needed however is the fact that while classical circuits can be mapped onto quantum hardware and therefor classic bits can be copied, this is not the case for general, non-orthogonal quantum bits. This is called the *No-Cloning-Theorem*.

Theorem 3.4.3 (No-Cloning-Theorem [31, p. 82]). *There is no unitary operation U, that fulfills the following equation for every $|\Psi\rangle$:*

$$U|\Psi\rangle|0\rangle = |\Psi\rangle|\Psi\rangle.$$

3.4.2 Quantum Complexity

There a many complexity classes appended in appendix C.2, classifying multiple types of problems by how hard they are to solve. E.g. how long a QTM has to run to solve a problem. Unfortunately, these classes are very broad, therefor this section introduces some more fine-grained complexity measures.

3.4.2.1 Oracle-model

Describing the complexity of quantum algorithms through the QTM also misses one major part of quantum computing: Quantum algorithms are often described to access quantum oracles (as defined in section 3.4.1.1) as subroutines to, in most cases, find some hidden information about this oracle. E.g. in Grover's database search algorithm, the database is given as a quantum oracle that flips the phase of the state that represents the solution to the search problem.

An interesting measure therefor is how often an algorithm needs to call/ access that oracle to extract that hidden information.

In the oracle model every operation and combination of unitaries between two consecutive oracle calls is considered to be a single unitary operation, or an oracle itself. Therefor any quantum algorithm can be described as a sequence of oracle calls and unitary operations between them[7]. So the complexity of an algorithm can solely be described as the number of calls to the given oracle it requires.

3.4.2.2 Gate-Model

Considering everything between two oracle calls as constant is already quite useful, but an even more fine-grained measure that also works if no oracles are used is the gate model. In the gate model every unitary operation is considered to be composed of k-local gates, and the complexity of an algorithm is measured by the number of local gates it uses.

Definition 3.4.4. *A gate is k-local if it acts on at most k qubits.*

Any k-local gate is congruent to a k-local unitary operator/ matrix U and decomposing a unitary into k-local gates is also possible but might be exponentially expensive. Therefor in the gate model the unitaries are already given as a combination k-local gates. The complexity of any unitary can then be given as follows:

Definition 3.4.5. *The diamond norm $||\mathcal{U}||_\diamond$ for a quantum channel $\mathcal{U}$ of size n is defined as*

$$\sup_\rho ||(\mathcal{U} \otimes I)\rho||_1$$

where ρ is a state on $2n$ qubits (given as the density matrix) and $||\cdot||_1$ is the trace norm.

Definition 3.4.6. *a unitary U has ϵ-complexity of at most d, denoted $C_\epsilon(U)$, if there exists a circuit C of depth d consisting only of k-local gates that approximates U with error ϵ: $\frac{1}{2}||\mathcal{U} - \mathcal{C}||_\diamond \leq \epsilon$ where $\mathcal{U}(\rho) = U\rho U^\dagger$ and $\mathcal{C}(\rho) = C\rho C^\dagger$ represent the quantum channels [8] corresponding with these unitary*

[7] The resulting algorithm could be considered as a single oracle itself, but that would not be useful as we are interested in how often the oracle given as an input is called.

[8] The unitary channel given by $\mathcal{U}(\rho) = U\rho U^\dagger$ applied to the density matrix ρ of a state is congruent the unitary operator $\mathcal{U}$ applied to the wave function of a state. If the wavefunction is given as $|\Psi\rangle$ the corresponding density matrix would be $\rho = |\Psi\rangle\langle\Psi|$.

transformations. [13] [9]

Building such a circuit is done by chaining gates behind each other using the matrix product and parallel to each other using the tensor product[10]. This is possible because of the following lemma:

Lemma 3.4.7. *Let $A, C \in K^{m\times m}$ and $B, D \in K^{n\times n}$.*

$$(A \otimes B) \cdot (C \otimes D) = (A \cdot C) \otimes (B \cdot D) \tag{3.4}$$

The unitary of the combined gate increases exponentially with the number of qubits it is applied to. Therefor k has to be chosen small, usually $k = 2$ or $k = 3$. It can be differentiated between the circuit size and the circuit depth as well as the circuit width. The circuit size is the number of gates in the circuit, the circuit depth is the longest path from input to output and the circuit width is the number of qubits that are operated on in parallel, visualized in fig. 3.1.

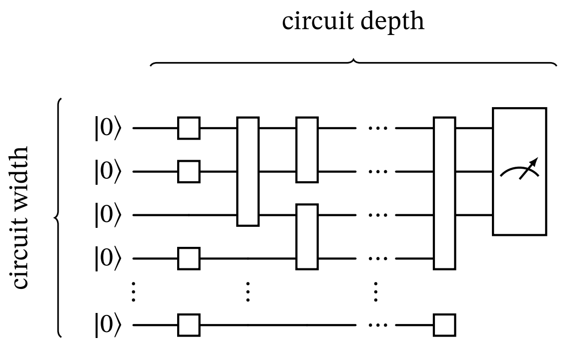


Figure 3.1: Schematic circuit for a quantum algorithm in the gate-model. The empty boxes represent arbitrary gates. Created by Valentin Pickel.

[9]Intuitively the diamond norm gives the inverse of the optimal error (by choosing the optimal input state) of distinguishing between these two channels with the help of an auxiliary register. More information on this can be learned in [18].

[10]If a gate G gets applied to n consecutive qudits each, it can be denoted as $G^{\otimes n}$.

Chapter 4

A Quantum Algorithm for Exposing Linear Approximations in Cryptographic Boolean Functions

4.1 Classical approach

As proven with theorem 3.2.6 every Boolean function has linear relations. Finding them is not a trivial task.

In a classical setting linear cryptanalysis traditionally requires a more manual approach, by looking at the exact structure of a cipher. In most cases that have been susceptible to linear cryptanalysis, the cipher is constructed via a SPN whose only non-linear part are the rather small S-boxes [40, 39]. Finding linear approximations for these is a rather simple task, as they are small enough to be exhaustively searched. The linearities are then combined to a linear approximation for the whole cipher as touched upon in section 3.3.

This approach however misses on potential linear approximations that only emerge by a combination of multiple S-Boxes and by looking at the whole cipher as one single entity. Doing this however is exponentially expensive in the size (block size) of the cipher and therefor unfeasible with a reasonably complex cipher.

The exponential complexity arises not only from the fact that the number of possible linear approximations for a cipher grows exponentially with the block size of the cipher, but also that calculating how good a single linear approximation is, is an exponential task in the block size of the cipher itself.

The naive classical algorithm to extract the best possible linear relation from any function $f : \mathbb{F}_2^{n_m} \times \mathbb{F}_2^{n_k} \mapsto \mathbb{F}_2^{n_c}$ is to simply try all possible linear relations and calculate their "goodness", this algorithm is described in algorithm 1.

As the two nested loops make apparent, this algorithm has a runtime of $\mathcal{O}\left(2^{2n_k+2n_m+n_c}\right)$. This is exponential in the block size of the cipher, but as we will see this is not the best possible runtime.

The asymptotically fastest, albeit more memory intensive and still exponential algorithm, utilizes the fast Walsh (or Fourier) transform. The fast Walsh Hadamard transform (FWHT) is very useful, as it can calculate the full Walsh spectrum of any $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ in $\mathcal{O}(n2^n)$, a python implementation is shown in listing 4.1. There might be a more integrated way of using the fast transform that already includes the output mask, but that is not known to me and might be a topic for future research. A more direct approach to use the fast transform is to calculate the Walsh transform for every possible output mask β as described in algorithm 2.

Algorithm 1: Naive classical algorithm to extract the best linear approximation.

```
input : $f : \mathbb{F}_2^{n_m} \times \mathbb{F}_2^{n_k} \mapsto \mathbb{F}_2^{n_c}$
$g^\top \leftarrow 0$;                                          /* current best goodness */
$\alpha^\top, \beta^\top, \gamma^\top \leftarrow 0^{\otimes n_m}, 0^{\otimes n_c}, 0^{\otimes n_k}$;     /* current best linear approximation */
for $\alpha \in \mathbb{F}_2^{n_m}, \beta \in \mathbb{F}_2^{n_c}, \gamma \in \mathbb{F}_2^{n_k} \setminus \{0\}$ do
    $g \leftarrow 0$ ;                          /* goodness of current linear approximation */
    for $k \in \mathbb{F}_2^{n_k}, m \in \mathbb{F}_2^{n_m}$ do
        $c \leftarrow f(k \Vert m)$;
        $t \leftarrow \langle\alpha|k\rangle \oplus \langle\beta|c\rangle \oplus \langle\gamma|m\rangle$;  /* is this approxmation true for this input? */
        if $t$ then
            $g \leftarrow g - 1$;
        end
        else
            $g \leftarrow g + 1$;
        end
    end
    if $g > g^\top$ then
        $g^\top \leftarrow g$;
        $\alpha^\top, \beta^\top, \gamma^\top \leftarrow \alpha, \beta, \gamma$;
    end
end
output: $\alpha^\top, \beta^\top, \gamma^\top$ and $g^\top$
```

```python
def fwht(a) -> None:
    """In-place Fast Walsh-Hadamard Transform of array a."""
    h = 1
    while h < len(a):
        # perform FWHT
        for i in range(0, len(a), h * 2):
            for j in range(i, i + h):
                x = a[j]
                y = a[j + h]
                a[j] = x + y
                a[j + h] = x - y
        # normalize and increment
        a /= 2
        h *= 2
```

Listing 4.1: Python code for the classic fast walsh transform [6].

Algorithm 2: Algorithm to extract the best linear approximation using the FWHT.

```
input  : $f : \mathbb{F}_2^{n_m+n_k} \mapsto \mathbb{F}_2^{n_c}$
$g^\top \leftarrow 0$;                                          /* current best goodness */
$\alpha^\top, \beta^\top, \gamma^\top \leftarrow 0^{\otimes n_m}, 0^{\otimes n_c}, 0^{\otimes n_k}$;      /* current best linear approximation */
for $\beta \in \mathbb{F}_2^{n_c} \setminus \{0\}$ do
    $G \leftarrow \mathrm{FWHT}_{\langle\beta|f\rangle}$ ;                    /* dict[a] of goodness for each input mask */
    for $\gamma \in \mathbb{F}_2^{n_k}, \alpha \in \mathbb{F}_2^{n_m}$ do
        if $G[\alpha \mathbin{+\!\!\!+} \gamma] > g^\top$ then
            $g^\top \leftarrow G[\alpha \mathbin{+\!\!\!+} \gamma]$;
            $\alpha^\top, \beta^\top, \gamma^\top \leftarrow \alpha, \beta, \gamma$;
        end
    end
end
output: $\alpha^\top, \beta^\top, \gamma^\top$ and $g^\top$
```

[a]Or blackbox function, this calculation needs $\mathcal{O}(n2^n)$ steps with $n = n_k + n_m$.

As becomes apparent by looking at the nested loops, this algorithm has a runtime of

$$\mathcal{O}\left(2^{n_c} \cdot \left(2^{n_k+n_m} + \mathcal{O}_{\mathrm{FWHT}^{n_k+n_m}}\right)\right)$$

where $\mathcal{O}_{\mathrm{FWHT}^n}$ denotes the runtime of the FWHT calculation for calculating the Walsh spectrum of a n-bit Boolean function. Therefor this approach achieves a nearly quadratic speedup over the naive algorithm with runtime

$$\mathcal{O}_{\text{algorithm 2}} \in \mathcal{O}\left((n_k + n_m)2^{n_k+n_m+n_c}\right). \tag{4.1}$$

4.2 Quantum Algorithm for Finding Linear Relations

As the classical approach was not t0o promising considering its runtime, we will now look at a quantum approach for finding linear relations. The approach covered here was first published by [37]. Their algorithm, extended to have two input registers, is shown in algorithm 3.

Algorithm 3: Malviya algorithm: modified from [37]

```
input  : Induced oracle $U_f$ for $f \in F(\mathbb{F}_2^{n_m} \times \mathbb{F}_2^{n_k}, \mathbb{F}_2^{n_c})$
$|m \mathbin{+\!\!\!+} k\rangle |c\rangle \leftarrow |0\rangle^{\otimes n_m} |0\rangle^{\otimes n_k} |0\rangle^{\otimes n_c}$;
$|m \mathbin{+\!\!\!+} k\rangle |c\rangle \leftarrow H^{\otimes(n_m+n_k)}(|m \mathbin{+\!\!\!+} k\rangle) |c\rangle$;
$|m \mathbin{+\!\!\!+} k\rangle |c\rangle \leftarrow U_f(|m \mathbin{+\!\!\!+} k\rangle |c\rangle)$;
$|m \mathbin{+\!\!\!+} k\rangle |c\rangle \leftarrow H^{\otimes(n_m+n_k+n_c)}(|m \mathbin{+\!\!\!+} k\rangle |c\rangle)$;
output: Linear (affine) relation $(\alpha, \beta, \gamma)$ for $f$
```

The circuit model of this algorithm can be found in fig. 4.1. The corresponding unitary would be $H^{n_m+n_k+n_c} U_f (H^{n_m+n_k} \otimes I)$.

Theorem 4.2.1. *Let $f : \mathbb{F}_2^{n_m} \times \mathbb{F}_2^{n_k} \mapsto \mathbb{F}_2^{n_c}$ be a cryptographic Boolean function. Running algorithm 3 on f will return any linear approximation (α, γ, β) with probability*

$$p_{\alpha,\gamma,\beta} = \frac{|c(\bar{t}_{\alpha \mathbin{+\!\!\!+} \gamma, \beta}(f))|^2}{2^{n_c}}.$$

Proof. This theorem can be proven by analyzing[1] the steps of the algorithm:

[1]The same algorithm has been independently found by Christoph Graebnitz, most of the analysis (including LaTeX source code) has been derived from his work.

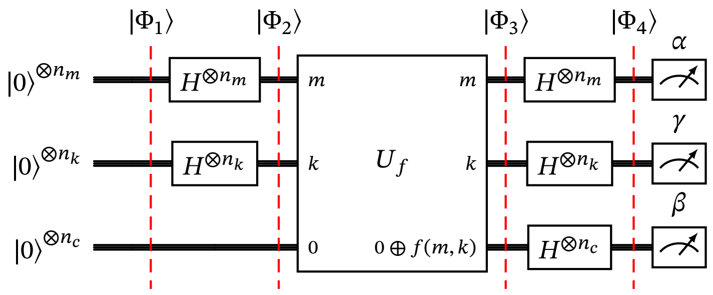


Figure 4.1: The circuit model depiction of the Malviya algorithm. The marked states correspond to the steps of the algorithm analyzed in the proof of following theorem 4.2.1.

1. $|\Phi_1\rangle = |0\rangle^{\otimes n_m} |0\rangle^{\otimes n_k} |0\rangle^{\otimes n_c}$

2. $|\Phi_2\rangle = (H^{\otimes(n_m+n_k)} |0\rangle^{\otimes n_m} |0\rangle^{\otimes n_k}) \otimes (I^{\otimes n_c} |0\rangle^{\otimes n_c}) = \frac{1}{\sqrt{2^{n_m+n_k}}} \sum_{m+k\in\mathbb{F}_2^{n_m+n_k}} |m + k\rangle |0\rangle^{\otimes n_c}$

3. $|\Phi_3\rangle = U_f(|\Phi_2\rangle) = \frac{1}{\sqrt{2^{n_m+n_k}}} \sum_{m+k\in\mathbb{F}_2^{n_m+n_k}} |m + k, f(m,k)\rangle$

4. After applying $H_{n_m+n_k+n_c}$ to $|\Phi_3\rangle$ the register $|\Phi_4\rangle$ is in the state

$$|\Phi_4\rangle = H_{n_m+n_k+n_c} \cdot \frac{1}{\sqrt{2^{n_m+n_k}}} \sum_{m+k\in\mathbb{F}_2^{n_m+n_k}} |m + k, f(m + k)\rangle \tag{4.2}$$

$$= \frac{1}{\sqrt{2^{n_m+n_k}}} \sum_{m+k\in\mathbb{F}_2^{n_m+n_k}} H_{n_m+n_k+n_c} \cdot (|m + k, f(m + k)\rangle) \tag{4.3}$$

$$= \frac{1}{\sqrt{2^{2*(n_m+n_k)+n_c}}} \sum_{m+k\in\mathbb{F}_2^{n_m+n_k}} \sum_{\alpha\in\mathbb{F}_2^{n_m},\gamma\in\mathbb{F}_2^{n_k},\beta\in\mathbb{F}_2^{n_c}} (-1)^{\langle\eta|m+k+f(m+k)\rangle} |\eta\rangle \tag{4.4}$$

Let $\alpha \in \mathbb{F}_2^{n_k}, \gamma \in \mathbb{F}_2^{n_m}, \beta \in \mathbb{F}_2^{n_c}$ s.t. $\eta = \alpha + \gamma + \beta$, written as $\alpha\gamma\beta$ for readability.
Also let the normalization factor $l = 1/\sqrt{2^{2*(n_m+n_k)+n_c}}$
Therefore eq. (4.4) can be written as

$$|\Phi_4\rangle = l * \sum_{m+k\in\mathbb{F}_2^{n_m+n_k}} \sum_{\alpha\in\mathbb{F}_2^{n_m},\gamma\in\mathbb{F}_2^{n_k},\beta\in\mathbb{F}_2^{n_c}} (-1)^{\langle\alpha+\gamma+\beta|m+k+f(m+k)\rangle} |\alpha\gamma\beta\rangle \tag{4.5}$$

$$= l * \sum_{m+k\in\mathbb{F}_2^{n_m+n_k}} \sum_{\alpha\in\mathbb{F}_2^{n_m},\gamma\in\mathbb{F}_2^{n_k},\beta\in\mathbb{F}_2^{n_c}} (-1)^{\langle\alpha|m\rangle\oplus\langle\gamma|k\rangle\oplus\langle\beta|f(m,k)\rangle} |\alpha\gamma\beta\rangle \tag{4.6}$$

using definition 3.2.2 eq. (4.6) can be written as

$$l * \sum_{\alpha\in\mathbb{F}_2^{n_m},\gamma\in\mathbb{F}_2^{n_k},\beta\in\mathbb{F}_2^{n_c}} \left(t^*_{\alpha+\gamma,\beta}(f) - t_{\alpha+\gamma,\beta}(f)\right) |\alpha\gamma\beta\rangle$$

which can then be further simplified using definition 3.2.3 with $n_x = n_m + n_k$ and $n_y = n_c$ to:

$$\sum_{\alpha\in\mathbb{F}_2^{n_m},\gamma\in\mathbb{F}_2^{n_k},\beta\in\mathbb{F}_2^{n_c}} \frac{t^*_{\alpha+\gamma,\beta}(f) - t_{\alpha+\gamma,\beta}(f)}{\sqrt{2^{2*(n_m+n_k)+n_c}}} = \sum_{\alpha\in\mathbb{F}_2^{n_m},\gamma\in\mathbb{F}_2^{n_k},\beta\in\mathbb{F}_2^{n_c}} \frac{t^*_{\alpha+\gamma,\beta}(f) - t_{\alpha+\gamma,\beta}(f)}{2^{n_m+n_k} * \sqrt{2^{n_c}}} \tag{4.7}$$

$$= \sum_{\alpha\in\mathbb{F}_2^{n_m},\gamma\in\mathbb{F}_2^{n_k},\beta\in\mathbb{F}_2^{n_c}} \frac{c(\bar{t}_{\alpha+\gamma,\beta}(f))}{\sqrt{2^{n_c}}} \tag{4.8}$$

5. Measuring $|\Phi_4\rangle$ will therefor return any linear relation $\alpha\gamma\beta$ with probability

$$p_{\alpha,\gamma,\beta} = \left(\frac{c(\bar{t}_{\alpha\doubleplus\gamma,\beta}(f))}{\sqrt{2^{n_c}}}\right)^2 = \frac{|c(\bar{t}_{\alpha\doubleplus\gamma,\beta}(f))|^2}{2^{n_c}} \tag{4.9}$$

□

Corollary 4.2.1.1. *The probability of measuring a linear relation (α, β, γ) whose correlation is large is more likely.*

While the success probability might not be very high (discussed in next section) the algorithm is still very interesting, as its *complexity* or *runtime* is in $\mathcal{O}(1)$ using the oracle model as it always only requires a single call to the oracle U_f.
In the gate model we can see that it has a width of $n_m + n_k + n_c$ qubits and a depth of $2 + d_{U_f}$ (the two consecutive Hadamard gates) and therefore a circuit size of $\mathcal{O}\left(n_m + n_k + n_c + \mathcal{O}_{U_f}\right)$ where d_{U_f} and $\mathcal{O}_{U_f}$ denote the depth and complexity of the oracle respectively.

4.3 Success rate of Malviya algorithm

As shown in theorem 4.2.1 the probability of measuring a single approximation (α, γ, β) is $p_{\alpha,\gamma,\beta} = \frac{|c(\bar{t}_{\alpha\doubleplus\gamma,\beta}(f))|^2}{2^{n_c}}$, but what exactly that means and how it relates to the success of the algorithm will be discussed in this section.

4.3.1 Trivial Solution

Having a look at the trivial solution (all masks equal zero, $\alpha \doubleplus \gamma = 0^{n_m+n_k}, \beta = 0^{n_c}$) we can see that this approximation would be always true, e.g. have correlation 1 ($c(\bar{t}_{0,0}(f)) = 1$).

Corollary 4.3.0.1. *The trivial, as well as other perfect i.e. always true (or always false) approximations, will be measured with a chance of* $1/2^{n_c}$

Proof. Perfect approximations will have a correlation of 1 or -1, which means that $|c(\bar{t}_{\alpha\doubleplus\gamma,\beta}(f))| = 1$. The rest follows from theorem 4.2.1. □

Non-Trivial Solutions

More complex approximations, which are neither always true nor always false are more interesting, but also harder to analyze. To analyze this we will first need to introduce some measures of how linear a function is to relate the success probability of the algorithm to those measures.

4.3.2 Linearity of Boolean Functions

We now want to define a measure on how "linearly approximateable" a function is.

To do that some further knowledge about Boolean functions is provided in the following.

Theorem 4.3.1. *The squared sum of the "goodness" i.e. the square of the correlation of every linear approximation for any Boolean mapping also sums to the size of its codomain:*

$$\forall f \in F(\mathbb{F}_2^{n_x}, \mathbb{F}_2^{n_y}) : \sum_{x\in\mathbb{F}_2^{n_x}, y\in\mathbb{F}_2^{n_y}} |c(\bar{t}_{x,y}(f))|^2 = 2^{n_y}$$

[1]Most graphics in this section where generated with my own code, provided at https://github.com/HannesGitH/MA-public/blob/main/success_probabilities.ipynb.

Proof. Since algorithm 3 is a valid quantum algorithm its success-probabilities sum to 1. Therefor for its result (4.9) it holds:

$$\sum_{\alpha\in\mathbb{F}_2^{n_m},\gamma\in\mathbb{F}_2^{n_k},\beta\in\mathbb{F}_2^{n_c}} \frac{|c(\bar{t}_{\alpha+\gamma,\beta}(f))|^2}{2^{n_c}} = 1$$

□

Corollary 4.3.1.1. *This also provides proof for theorem 3.2.4*

Proof. by corollary 3.2.3.1 □

Let $\mathcal{A}_{n,m}$ denote the affine subset of $F(\mathbb{F}_2^n, \mathbb{F}_2^m)$.

Definition 4.3.2. *The non-linearity of a Boolean function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is defined as the minimal Hamming-Distance between its truth table and any other linear functions truth table [10, definition 5][2]:*

$$\bar{\mathcal{L}}(f) := \min_{g\in\mathcal{A}_{n,1}} \#\{x \in \mathbb{F}_2^n : f(x) \neq g(x)\}$$

This could be expanded to a measure of the non-linearity of a vectorial Boolean function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$ by:

Corollary 4.3.2.1. *The non-linearity of a vectorial Boolean function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$ is defined as the minimal Hamming-Distance between its truth table and any other linear functions truth table:*

$$\bar{\mathcal{L}}(f) := \min_{g\in\mathcal{A}_{n_x,1}} \#\{x \in \mathbb{F}_2^n : f(x) \neq g(x)\}$$

Theorem 4.3.3. *We could also measure the non-linearity of $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ by:*

$$\bar{\mathcal{L}}(f) = \min_{g\in\mathcal{A}_{n,1}} 2^{n-1}\left(c(f = g) + 1\right)$$

Proof. Let $f, g \in F(\mathbb{F}_2^n, \mathbb{F}_2)$

$$\begin{aligned} 2^n \cdot c(f = g) &= \#\{\xi \in \mathbb{F}_2^n \,|\, f(\xi) \neq g(\xi)\} - \#\{\xi \in \mathbb{F}_2^n \,|\, f(\xi) = g(\xi)\} && (4.10)\\ &= \#\{\xi \in \mathbb{F}_2^n \,|\, f(\xi) \neq g(\xi)\} - 2^n + \#\{\xi \in \mathbb{F}_2^n \,|\, f(\xi) \neq g(\xi)\} && (4.11)\\ \Leftrightarrow 2^{n-1}\left(c(f = g) + 1\right) &= \#\{\xi \in \mathbb{F}_2^n \,|\, f(\xi) \neq g(\xi)\} && (4.12)\end{aligned}$$

□

Corollary 4.3.3.1. *We could also measure the non-linearity of $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ by:*

$$\min_{\alpha\in\mathbb{F}_2^n} 2^{n-1}\left(c(t_{\alpha,1}(f)) + 1\right)$$

With the minor difference that this only regards linear (not affine) functions.

Proof. Let α be the coefficients of linear function g. As of definition 3.2.1:

$$\bar{t}_{\alpha,1}(f, x) = (\langle\alpha|x\rangle \Leftrightarrow f(x)) = (g(x) \Leftrightarrow f(x)) \qquad (4.13)$$

□

[2]This is also congruent to f being $\epsilon :\geq \frac{\bar{\mathcal{L}}(f)}{2^n}$ near to being linear, as used in [24]. Or equivalently f is $\epsilon := \frac{\bar{\mathcal{L}}(f)-1}{2^n}$ far from being linear.

As the Hamming variant can be extended into a measure for vectorial Boolean functions, so can the correlation variant. One such idea would be to use

$$\bar{\mathcal{L}}(f) := \frac{1}{2^{n-1}} \min_{m_x \in \mathbb{F}_2^n} c(\bar{t}_{m_x,(1^{\otimes n_y})}(f)) + 1$$

with $1^{\otimes n_y}$ being the all-ones vector of length n_y, but that would disregard all output masks, so we rather define it as follows:

Definition 4.3.4. *The non-linearity of a vectorial Boolean function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$ is defined as:*

$$\bar{\mathcal{L}}(f) := \frac{1}{2^{n-1}} \min_{\substack{m_x,m_y \in \mathbb{F}_2^n \\ m_y \neq 0}} c(\bar{t}_{m_x,m_y}(f)) + 1$$

The input mask is allowed to be zero s.t. the overall distribution of f can be taken into account. Meanwhile, the output mask must not be zero, as that would only yield trivial results[3].

Anyway, the non-linearity of a function is not of interest for this thesis, but rather its "linear-approximatibility".

To calculate that we could use the inverse of the non-linearity, as the smaller the non-linearity, the more linear the function is: $1/\bar{\mathcal{L}}$.
Or another approach would be to use the maximum instead of the minimum, although the trivial solution should be disregarded, as it is always the maximum:

$$\mathcal{L}_{\max}(f) := \max_{\substack{m_x,m_y \in \mathbb{F}_2^n \\ m_y \neq 0}} |c(\bar{t}_{m_x,m_y}(f))|. \tag{4.14}$$

This would make the most sense in a classical setting, where only one approximation will be outputted and used. But as soon as we try linear cryptanalysis and want to have multiple linear approximations to get multiple bits of information, or a quantum (or nondeterministic) version is used where multiple approximations might be measured, we want to take multiple approximations into account.

As we already have a measure of how good a single approximation is, we could simply sum up the approximations with a specific monotonically increasing activation function $\mathbb{a} : \mathbb{R} \mapsto \mathbb{R}$ and use that as a measure of how good the function can be linearly approximated:

$$\mathcal{L}_{\mathbb{a}}(f) := \sum_{\substack{m_x,m_y \in \mathbb{F}_2^n \\ m_y \neq 0}} \mathbb{a}\left(|c(\bar{t}_{m_x,m_y}(f))|\right). \tag{4.15}$$

This seems like an interesting approach, but it is not very clear what the best activation function would be. One such idea, with $\mathbb{a}(x) = x^2$, for example would not work at all, since that results in a constant value of 2^{2n} for any function f, as shown in theorem 3.2.7.

As a result of that observation the following hypothesis is made:

Hypothesis 4.3.5. *Any activation function $\mathbb{a}$ used in eq. (4.15) should have a second derivate that is greater than the one of x^2 in the relevant area (i.e. $x \in [0, 1]$). It should also be monotonically increasing and have a value of 0 at 0.*

To test hypothesis 4.3.5 we will include the activation function $\mathbb{a}(x) = \sqrt{x}$ in most analyses.

One argument strengthening hypothesis 4.3.5, stated in [38] as a direct result of Parsevals theorem, is that the closer a specific correlation goes to zero the higher the other values squares get and vice versa.

[3]This reduces the sum by 2^{n_x} possible masks. But all of them with $m_x \neq 0$ will have correlation zero and therefor play no role. Only the trivial mask $m_x = m_y = 0$ will always be true and therefor has to be disregarded.

Therefor we cannot simply look for most high correlations, but rather for the most uneven distribution (furthest from bent functions) of correlations, i.e. a few very high correlations and many very low correlations.
I therefor propose the following measure for the “approximatibility” of a Boolean function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$:

$$\mathcal{L}_{\wedge 4}(f) := \sum_{\substack{m_x, m_y \in \mathbb{F}_2^n \\ m_y \neq 0}} |c(\bar{t}_{m_x,m_y}(f))|^4$$

As a quick preliminary check that this does not result in a constant value for all functions, a quick analysis is made. For readability reasons $m_y = 1$:

$$\sum_{\alpha=0}^{2^n} |c(\bar{t}_{\alpha,1}(f))| = \sum_{\alpha \in \mathbb{F}_2^n} \left(\sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x \rangle \oplus f(x)} \right)^4 \tag{4.16}$$

$$= \sum_{\alpha \in \mathbb{F}_2^n} \left(\sum_{x_1 \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x_1 \rangle \oplus f(x_1)} * \sum_{x_2 \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x_2 \rangle \oplus f(x_2)} * \sum_{x_3 \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x_3 \rangle \oplus f(x_3)} * \sum_{x_4 \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x_4 \rangle \oplus f(x_4)} \right) \tag{4.17}$$

$$= \sum_{x_1,x_2,x_3,x_4 \in \mathbb{F}_2^n} (-1)^{f(x_1) \oplus f(x_2) \oplus f(x_3) \oplus f(x_4)} \sum_{\alpha \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x_1 \rangle \oplus \langle \alpha | x_2 \rangle \oplus \langle \alpha | x_3 \rangle \oplus \langle \alpha | x_4 \rangle} \tag{4.18}$$

$$= \sum_{x_1,x_2,x_3,x_4 \in \mathbb{F}_2^n} (-1)^{f(x_1) \oplus f(x_2) \oplus f(x_3) \oplus f(x_4)} \sum_{\alpha \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x_1 \oplus x_2 \oplus x_3 \oplus x_4 \rangle} \tag{4.19}$$

From lemma 3.2.8 follows

$$\sum_{\alpha \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x_1 \oplus x_2 \oplus x_3 \oplus x_4 \rangle} = \begin{cases} 2^n, & if\ x_1 = x_2 \oplus x_3 \oplus x_4 \\ 0, & else \end{cases}$$

therefor

$$\sum_{\alpha=0}^{2^n} |c(\bar{t}_{\alpha,1}(f))| = \sum_{\alpha \in \mathbb{F}_2^n} \left(\sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha | x \rangle \oplus f(x)} \right)^4 \tag{4.20}$$

$$= \sum_{x_1,x_2,x_3,x_4 \in \mathbb{F}_2^n} (-1)^{f(x_2 \oplus x_3 \oplus x_4) \oplus f(x_2) \oplus f(x_3) \oplus f(x_4)} 2^n. \tag{4.21}$$

Now, if f is linear this results in $\sum_{x_1,x_2,x_3,x_4 \in \mathbb{F}_2^n} (-1)^{f(x_2) \oplus f(x_3) \oplus f(x_4) \oplus f(x_2) \oplus f(x_3) \oplus f(x_4)} 2^n = 2^{4n}$. But if that is not the case then this value gets lower with less “linear approximatible” fs.

One might also consider an exponential activation function like $\mathbb{a}(x) = 2^x$ but as seen in fig. 4.2 this also has a lower second derivate in the relevant area (i.e. $x \in [0, 1]$) than x^2 and therefor is not suited as well.

Another, arguably even simpler idea would be to use a binary threshold (that simply outputs whether input X is greater or equal than a specified threshold τ) as an activation function.
Let

$$\delta_\tau(x) = \begin{cases} 1 & x \geq \tau \\ 0 & \text{else} \end{cases}$$

We can now use δ as our activation function $\mathbb{a}$ to simply count how many approximations with a certain “goodness” (correlation) exist for a given function:

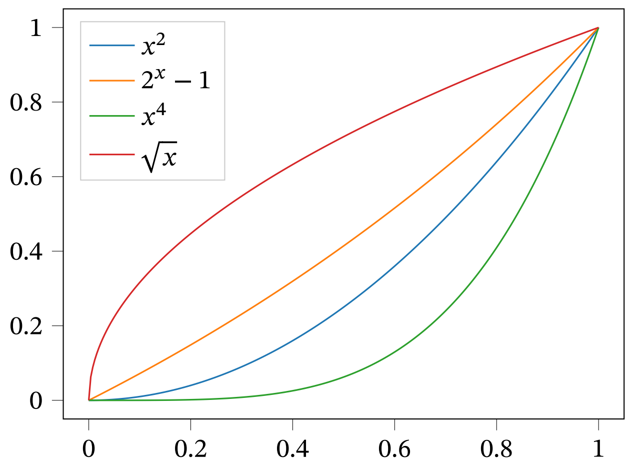


Figure 4.2: Comparison of different activation functions. As x^2 results in a constant measure for any function all functions above promote non-linear functions while functions below promote functions with higher linearities.

Definition 4.3.6. *let*

$$\mathcal{L}_{\delta_\tau}(f) := \sum_{m_x \in \mathbb{F}_2^{n_x}, m_y \in \mathbb{F}_2^{n_y}} \delta_\tau\left(\left|c(\bar{t}_{m_x,m_y}(f))\right|\right)$$

be a measure of the linear approximateability of any vectorial Boolean function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$

$\mathcal{L}_{\delta_\tau}, \mathcal{L}_{\wedge 4}, \mathcal{L}_{\max}, \frac{1}{\bar{\mathcal{L}}}$ might all be reasonable measures of linear approximatibility depending on the context used, but they also all have the common disadvantage of being exponential in the number of calculations needed to compute them.

But I could not find any good measure of linear approximatibility computable in P (polynomial time), and can think of a way how that could be classically possible. Other measures e.g. by [10] are exponentially hard to compute as well.

4.3.2.1 Normalized measures

Another disadvantage is, that they all have a different scaling, resulting in different values even for the same linear, i.e. maximal approximatible function. That is why we define the following somewhat normalized versions (compare with end of section 4.3.4.1) for any $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$:

$$
\begin{aligned}
\mathcal{L}^*_{\max} = \mathcal{L}_{\max} &= \max_{\substack{m_x\in\mathbb{F}_2^{n_x}, m_y\in\mathbb{F}_2^{n_y}\\ m_y\neq 0}} |c(\bar{t}_{m_x,m_y}(f))| \\
\frac{1}{\bar{\mathcal{L}}}^* = \frac{2^{1-n_x}}{\bar{\mathcal{L}}} &= \min_{m_x\in\mathbb{F}_2^{n}} c(\bar{t}_{m_x,(1\otimes)}(f)) + 1 \\
\mathcal{L}^*_{\delta_\tau} = \mathcal{L}_{\delta_\tau}/2^{n_y} &= \frac{1}{2^{n_y}} \sum_{\substack{m_x\in\mathbb{F}_2^{n_x}, m_y\in\mathbb{F}_2^{n_y}\\ m_y\neq 0}} \delta_\tau\left(\left|c(\bar{t}_{m_x,m_y}(f))\right|\right) \\
\mathcal{L}^*_{\wedge 4} = \mathcal{L}_{\wedge 4}/2^{n_y} &= \frac{1}{2^{n_y}} \sum_{\substack{m_x\in\mathbb{F}_2^{n_x}, m_y\in\mathbb{F}_2^{n_y}\\ m_y\neq 0}} \left|c(\bar{t}_{m_x,m_y}(f))\right|^4 \\
\mathcal{L}^*_{\wedge\frac{1}{2}} = \mathcal{L}_{\wedge\frac{1}{2}}/2^{n_y} &= \frac{1}{2^{n_y}} \sum_{\substack{m_x\in\mathbb{F}_2^{n_x}, m_y\in\mathbb{F}_2^{n_y}\\ m_y\neq 0}} \left|c(\bar{t}_{m_x,m_y}(f))\right|^{\frac{1}{2}}
\end{aligned}
$$

With those measures we can now try to analyze the success probability of the algorithm for different kinds of functions.

4.3.3 Their function

In their paper Malviya et al. [37] do not give a success probability for their algorithm. They do, however, present some empirical results for one arbitrary 4-to4-bit vectorial Boolean function.

We will have a look at their results first and then try to generalize them to any function. Having a closer look at their function we can apply some measures created in section 4.3.2. As their function is quite small we can calculate the correlation of *every* possible mask/ approximation. Resulting distributions are shown in fig. 4.3 and fig. 4.4.

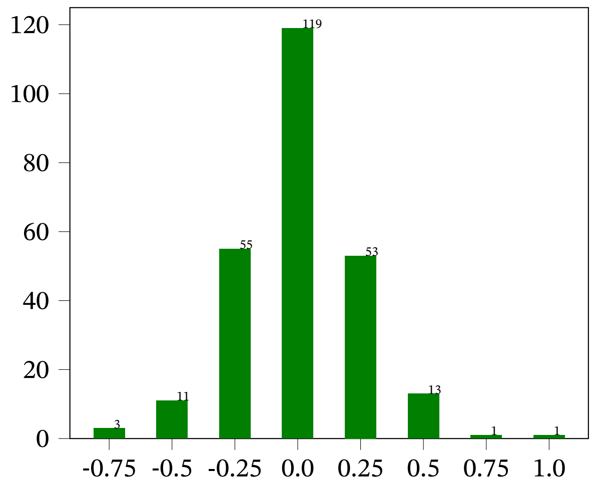


Figure 4.3: Count of how many different masks (m_x, m_y) have a certain correlation $c(\bar{t}_{m_x,m_y}(f))$, for the f provided by Malviya et al. That this approaches a normal distribution is not a coincidence and will be discussed in section 4.3.4.2.

These figures, especially fig. 4.5 should give a good intuition how f behaves and how that results in the success probability of the algorithm.

As we can see, the optimally bad approximations (i.e. correlation zero) will not be measured, which is a good thing. However, the probability of measuring a good approximation (i.e. correlation one) is

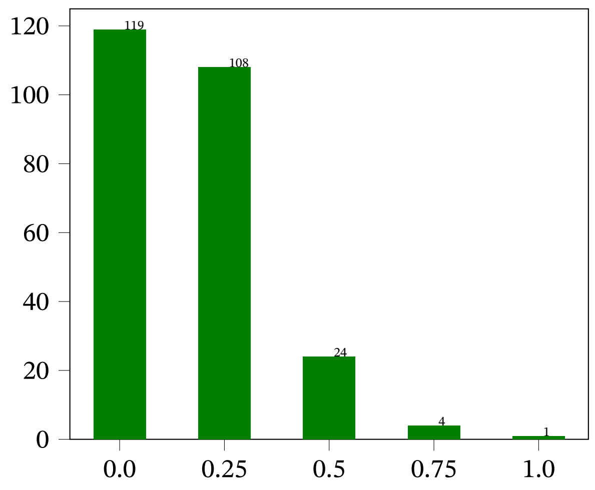


Figure 4.4: Count of how many different masks (m_x, m_y) have a certain "goodness", i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for the f provided by Malviya et al.

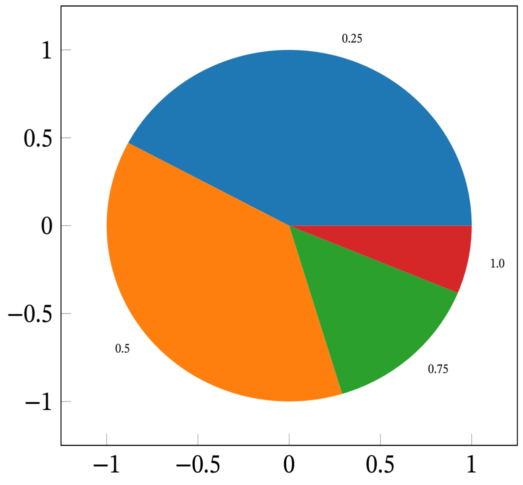


Figure 4.5: Probability of measuring a mask with a certain goodness, i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for the f provided by Malviya et al. The probabilities are calculated in accordance with eq. (4.25), and are 0, $\frac{27}{16\cdot 4}$, $\frac{24}{16\cdot 4}$, $\frac{9}{16\cdot 4}$, $\frac{4}{16\cdot 4}$ in the same order as in fig. 4.4 respectively.

only $\frac{9}{16\cdot 4}$, which is quite low for the low complexity of f. Also, the trivial approximation (i.e. masks zero) gets measured with probability $\frac{1}{16}$, which is not ideal either, as this approximation is always true and thus not interesting, this probability is exactly as expected (compare section 4.3.1) though.

With that information we can also calculate some measures of how linear, as defined in section 4.3.2, f is.

Iteration size	Trivial	Highly	Least
1024	60.8 (5.94%)	150.4 (14.69%)	436.4 (42.62%)
2048	130.6 (6.38%)	292.6 (14.29%)	865.8 (42.28%)
3072	201.6 (6.56%)	418.2 (13.61%)	1281.8 (41.73%)
4096	247.6 (6.04%)	586.8 (14.33%)	1708.8 (41.72%)
5120	314.8 (6.15%)	717.2 (14.01%)	2153.2 (42.05%)
6144	390.4 (6.35%)	856.8 (13.95%)	2601 (42.33%)
7168	447.4 (6.24%)	1024 (14.29%)	3003 (41.89%)
8192	520.8 (6.36%)	1163.6 (14.20%)	3446.8 (42.08%)
Average %	6.25%	14.17%	42.09%

Table 4.1: Simulation results for the algorithm by Malviya et al. with different iteration sizes[37].

$$\begin{aligned}
\mathcal{L}^*_{\max} & & &= 0.75 \\
\frac{1}{\bar{\mathcal{L}}}^{*} &= c(\bar{t}_{1000,1111}(f)) + 1 = \frac{8}{-0.75+1} & &= 4 \\
\mathcal{L}^*_{\delta_{0.7}} & & &= \frac{4}{16} \\
\mathcal{L}^*_{\delta_{0.3}} & & &= \frac{28}{16} \\
\mathcal{L}^*_{\wedge\frac{1}{2}} &= \frac{1}{16}\Big(108 \cdot \sqrt{0.25} + 24 \cdot \sqrt{0.5} + 4 \cdot \sqrt{0.75} + 1 \cdot \sqrt{1}\Big) & &\approx 4.7 \\
\mathcal{L}^*_{\wedge 4} &= \frac{1}{16}\Big(108 \cdot 0.25^4 + 24 \cdot 0.5^4 + 4 \cdot 0.75^4 + 1 \cdot 1^4\Big) & &= \frac{67}{256} \\
\mathcal{L}_{\mathbb{p}_{0.7}}{}^{4} &= 4 \cdot (\frac{3}{4}\frac{1}{2^4})^2 & &= \frac{9}{1024}
\end{aligned}$$

With some intuitive understanding of the function f we can look at the results by Malviya et al. (table 4.1) again. We see that measuring the trivial approximation is exactly as likely as calculated. The "highly" linear approximations are certainly meant to be the ones with absolute correlation 0.75 as $\frac{9}{64} \approx 0.1417$, and with the same reasoning the "least" linear approximations are the ones with absolute correlation 0.25. The missing 37.49% are therefor the approximations with absolute correlation 0.5. As all uncertainty in their results only comes from the random measurement in the end (they probably could have skipped that, and just look at the resulting distribution before measuring, resulting in the same probabilities as my theoretical analysis) and not from any decohering effects, the very bad approximations (with correlation 0) are not measured at all. This would not be the case in a real quantum computer, where decoherence would result in a non-zero probability of measuring the trivial approximation, as can be seen in their practical results, where those bad approximations are measured with a probability of $> \frac{1}{2}$.

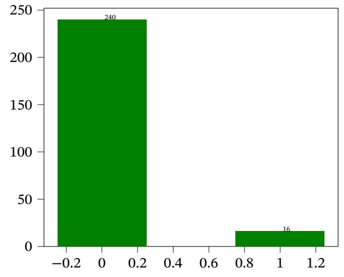
4.3.4 Other special cases

We can now try to generalize the results from section 4.3.3 to any function f. We will start by analyzing a few special cases and then try to generalize those results.

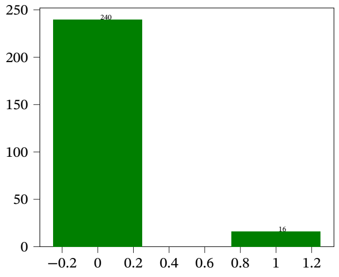
4.3.4.1 Linear (affine) function

If f is linear, i.e. $f(x) = Mx$ for some binary matrix M, then the algorithm is guaranteed to measure one perfect approximation. This can be seen in fig. 4.6c and fig. 4.7c for the affine case.

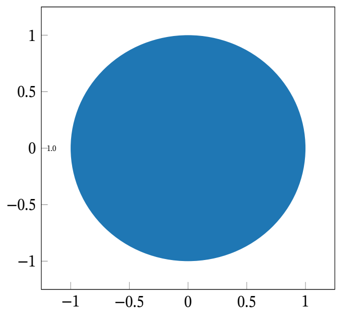
This result can easily be explained if we have a closer look at what a vectorial Boolean function actually is:



(a) Count of how many different masks (m_x, m_y) have a certain correlation $c(\bar{t}_{m_x,m_y}(f))$, for a linear 4-to-4-bit function.

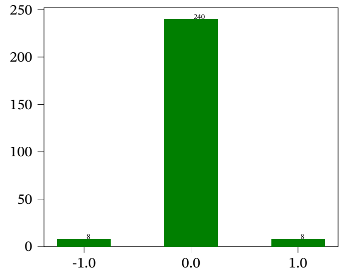


(b) Count of how many different masks (m_x, m_y) have a certain "goodness", i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for a linear 4-to-4-bit function.

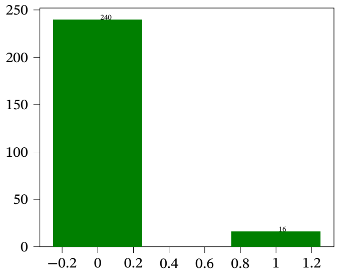


(c) Probability of measuring a mask with a certain goodness, i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for a linear 4-to-4-bit function.

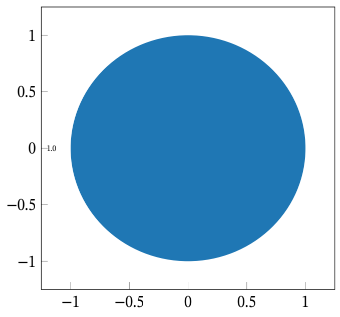
Figure 4.6: Analysis of a linear 4-to-4-bit function.



(a) Count of how many different masks (m_x, m_y) have a certain correlation $c(\bar{t}_{m_x,m_y}(f))$, for an affine 4-to-4-bit function.



(b) Count of how many different masks (m_x, m_y) have a certain "goodness", i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for an affine 4-to-4-bit function.



(c) Probability of measuring a mask with a certain goodness, i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for an affine 4-to-4-bit function.

Figure 4.7: Analysis of an affine 4-to-4-bit function.

Lemma 4.3.7. *Any vectorial Boolean function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$ can be constructed by combining n_y linear Boolean functions f_i with n_x input bits each, i.e. $f(x) = \Vert_{i=1}^{n_y} f_i(x_i)$.*

This means that every output bit can be considered as its own linear function, and thus there are n_y single bit linear functions that can be combined into 2^{n_y} different vectorial linear functions (of which one is the trivial one). Each of them has a perfect approximation, as it itself is perfectly linear, and thus the algorithm will measure one of them with probability 1.

In fig. 4.6 $n_y = 4$ and therefor $2^{n_y} = 16$ of the $2^{n_x+n_y} = 256$ approximations are perfect, while the rest (as proven by theorem 3.2.4) have correlation 0 and therefor will not be measured.
The only difference between fig. 4.7(a) and fig. 4.6(a) is, that the single bit functions, of which the affine function is constructed of, are affine instead of linear. This means that some linear approximations might be inverted resulting in a negative (still perfect) correlation.

This result is in accordance with theorem 3.2.7 and can even be generalized to:

Lemma 4.3.8. *Any function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$ that has exactly 2^{n_y} perfect approximations (e.g. all affine functions) results in Malviya algorithm measuring one of those approximations.*

To better generalize this we will calculate the "approximatibility"-measures for these affine functions:

$$\begin{aligned}
\mathcal{L}_{\max} &= 1 \\
\frac{1}{\bar{\mathcal{L}}} = \frac{2^{n-1}}{0+1} &= 2^{n_x-1} \\
\mathcal{L}_{\delta_{0.7}} &= 2^{n_y}-1 \\
\mathcal{L}_{\delta_{0.3}} &= 2^{n_y}-1 \\
\mathcal{L}_{\wedge x} = 2^{n_y} - 1 * 1^x + 0 &= 2^{n_y}-1 \\
\mathcal{L}_{\mathbb{P}_\tau} &= 1 - \frac{1}{2^{n_y}}
\end{aligned}$$

Where the -1 is a result of disregarding the trivial approximation. Also note that $\frac{1}{\bar{\mathcal{L}}}$ here only regards linear (not affine functions) since affine functions can get even higher values, depending on their exact setup, up to infinity.
As we can see they are not normalized, but we might want to do that to compare them to each other. That is why there are normalized versions of them introduced in section 4.3.2.1.

These results also apply to asymmetric functions, i.e. functions where $n_x \neq n_y$, results are shown in fig. D.1 and fig. D.2.

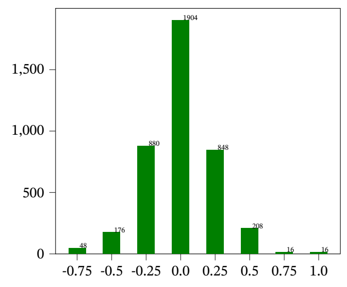
The first major observation here is, that choosing the right measure, or rather the right τ for the measure, is very important. This does not mean the measure is not suitable for the task, but rather that the measure has to be chosen carefully, for example the f chosen by Malviya et al. has much better results with $\tau = 0.3$ as there are quite a few mediocre approximations, while linear functions only have a few very good ones, so choosing a too low τ can be a pitfall.

Partially linear

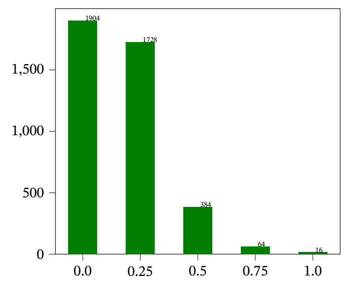
A possible generalization of those linear functions is to look at functions that are only partially linear, defined as follows:

Definition 4.3.9. *A $\frac{k}{n_y}$-partially vectorial Boolean function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$ is constructed of k single-bit linear Boolean functions and $n_y - k$ arbitrary single-bit Boolean function f_i with n_x input bits each.*

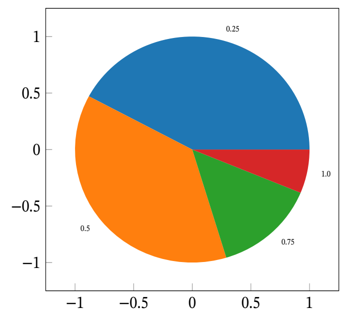
Definition 4.3.9 is a generalization of lemma 4.3.7.



(a) Count of how many different masks (m_x, m_y) have a certain correlation $c(\bar{t}_{m_x,m_y}(f))$.



(b) Count of how many different masks (m_x, m_y) have a certain "goodness", i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$.



(c) Probability of measuring a mask with a certain goodness.

Figure 4.8: Analysis of a partially linear 4-to-4-bit function, in this case the concatenation of the identity (4 to 4 bit linear function) and the function used by Malviya et al. (4 to 4 bit non-linear function). In comparison to the figures in section 4.3.3 it becomes apparent that it is exactly the same distribution, just multiplied by a factor of $2^4 = 16$ (the additional linear output bits).

It is easy to see that this function has 2^k perfect approximations, i.e. $\mathcal{L}^*_{\delta_1}(f) = \frac{2^k}{2^{n_y}}$. Namely, those that ignore the $n_y - k$ arbitrary output bits and only look at all possible combinations of the k linear ones. Adding a single non-linear output bit with some specific distribution will make this distribution hold true no matter how the masks of the linear bit are chosen, as the linear part will either always (or never) or exactly half of the time hold true, not influencing the non-linear part.
This directly extends into any amount of non-linear output bits.
Looking at an example, the concatenation of the identity (4 to 4 bit linear function) and the function used by Malviya et al. (4 to 4 bit non-linear function),
i.e. in python notation `lambda x : x | (f_malviya(x)<<4)`, results in a function with exactly the same properties as the function used by Malviya et al., just multiplied by the possible combinations of the linear output bits (in this case $2^4 = 16$) as can be seen in fig. 4.8.

Bent function

On the complete opposite site of the spectrum, in some ways exactly between affine (in this case affine but not linear) and linear functions, are bent functions.

Definition 4.3.10 ([38, 45, 1]). *$f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is called bent iff*

$$\forall \alpha \in \mathbb{F}_2 : \hat{\chi}_f(\alpha) = \pm 2^{\frac{n}{2}} = \pm\sqrt{2^n}$$

Corollary 4.3.10.1. *If $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is bent then*

$$\forall \alpha \in \mathbb{F}_2 : c(\bar{t}_\alpha(f)) = \pm 2^{-\frac{n}{2}} = \pm\frac{1}{\sqrt{2^n}}$$

Proof. directly follows from definition 4.3.10 and corollary 3.2.3.1. □

This could be considered the worst case for the Malviya algorithm, as it has the most evenly distributed correlations. The measures would be as follows:

$$
\begin{aligned}
&\mathcal{L}^*_{\max} && = 2^{-\frac{n}{2}} \\
&\frac{1}{\bar{\mathcal{L}}}^* && = 1 - 2^{-\frac{n}{2}} \\
&\mathcal{L}^*_{\delta_{\epsilon^-}} = \mathcal{L}_{\mathbb{p}_{\epsilon^-}} && = 1 - \frac{1}{2^n} \\
&\mathcal{L}^*_{\delta_{\epsilon^+}} = \mathcal{L}_{\mathbb{p}_{\epsilon^+}} && = 0 \\
&\mathcal{L}^*_{\wedge\frac{1}{2}} = \frac{1}{2^n}\sum_{\alpha=0}^{2^n} \frac{1}{\sqrt{2^n}^{\frac{1}{2}}} = \frac{1}{\sqrt[4]{2^n}} \\
&\mathcal{L}^*_{\wedge 4} = \frac{1}{2^n}\sum_{\alpha=0}^{2^n} \frac{1}{\sqrt{2^n}^{4}} = \frac{1}{2^{2n}}
\end{aligned}
$$

Where $\epsilon^{\pm}(x) := \lim_{h\to 0} x \pm h$ is the smallest[5] number that is bigger than x (ϵ^+) or the biggest number smaller than x (ϵ^-).

A graphic here is omitted, as every correlation (except the trivial one) is equal and therefor the probability of measuring any of them is equal.
One could argue that the Malviya algorithm can not be successful in this case, as there is no good approximation. Except if success will be defined via $\tau \leq \epsilon^-$, but that seems unreasonable.
So if the function is bent, this setup of linear cryptanalysis can not help.

Thus far we only considered single bit functions, but we can generalize this to any n_x-to-n_y-bit vectorial Boolean function f by looking at the single bit functions f_i that f is constructed of.

Definition 4.3.11. *A vectorial Boolean function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$ is called bent iff all of its single bit functions f_i (as deconstructed via lemma 4.3.7) are bent [51].*

This definition would allow two subfunctions f_i and f_j to be identical, i.e. $\forall x \in \mathbb{F}_2^{n_x} : f_i(x) = f_j(x)$. Therefor two output bits of f would always be equal, resulting in the approximation that has exactly those bits set to have correlation 1, which would give us one bit of information. A design of a cryptographic function would want to avoid that, and therefor we will use a slightly different definition (similar to [41]), that is more restrictive but also more useful for our purposes, it relies on the same idea of extending definitions via the correlation as used in definition 4.3.4 and therefor disallows output bits to correlate[6] with each other:

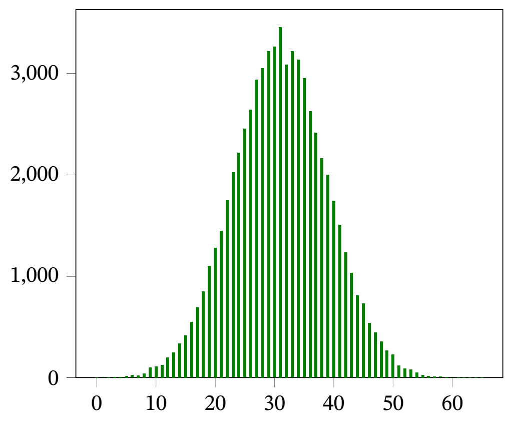
Definition 4.3.12. *A vectorial Boolean function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$ is called bent iff*

$$\forall m_x \in \mathbb{F}_2^{n_x}, m_y \in \mathbb{F}_2^{n_y}, n_y \neq 0 : c(\bar{t}_{m_x,m_y}(f)) = \pm\frac{1}{\sqrt{2^{n_x+n_y}}}$$

Compared to the non-vectorial (single bit) variant, the distribution and therefor success-rate of the Malviya algorithm does not change.

[5] Does not have to be.

[6] Except exactly with correlation $\pm\frac{1}{\sqrt{2^{n_x+n_y}}}$


(a) Count of how many different masks (m_x, m_y) have a certain correlation $c(\bar{t}_{m_x,m_y}(f))$, for a random 8-to-8-bit function.

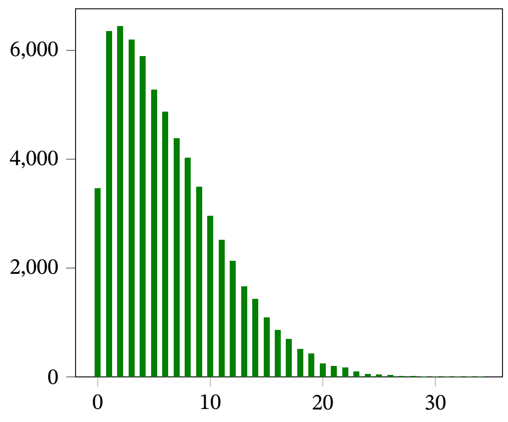

(b) Count of how many different masks (m_x, m_y) have a certain "goodness", i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for a random 8-to-8-bit function.

Figure 4.9: Analysis of a random 8-to-8-bit function. The probabilities have been omitted as they were quite crowded and not very interesting.

$$\begin{aligned}
\mathcal{L}^*_{\max} && &= 2^{-\frac{n_x+n_y}{2}} \\
\frac{1}{\bar{\mathcal{L}}}^{*} && &= 1 - 2^{-\frac{n_x+n_y}{2}} \\
\mathcal{L}^*_{\delta_{\epsilon-}} &= \mathcal{L}_{\mathbb{p}_{\epsilon-}} & &= 1 - \frac{1}{2^{n_x+n_y}} \\
\mathcal{L}^*_{\delta_{\epsilon+}} &= \mathcal{L}_{\mathbb{p}_{\epsilon+}} & &= 0
\end{aligned}$$

$$\mathcal{L}^*_{\wedge\frac{1}{2}} = \frac{1}{2^{n_y}} \sum_{\substack{m_x \in \mathbb{F}_2^{n_x}, m_y \in \mathbb{F}_2^{n_y} \\ n_y \neq 0}} \left(\frac{1}{\sqrt{2^{n_x+n_y}}}\right)^{\frac{1}{2}} = \frac{2^{n_y-1} \cdot 2^{n_x}}{2^{n-y}} \frac{1}{\sqrt[4]{2^{n_x+n_y}}} = \frac{\sqrt{n_x}}{2\sqrt{m_x}}$$

$$\mathcal{L}^*_{\wedge 4} = \frac{1}{2^{n_y}} \sum_{\substack{m_x \in \mathbb{F}_2^{n_x}, m_y \in \mathbb{F}_2^{n_y} \\ n_y \neq 0}} \left(\frac{1}{\sqrt{2^{n_x+n_y}}}\right)^{4} = \frac{2^{n_y-1} \cdot 2^{n_x}}{2^{n-y}} \frac{1}{2^{2(n_x+n_y)}} = \frac{1}{2^{2n_y+n_x+1}}$$

4.3.4.2 Random function: normally distributed correlations

As many cryptographic schemes have *plausible deniability* as a goal, they have to be *indistinguishable from random noise*. Many schemes also reach that as a side effect[44]. Therefor analyzing functions that are indistinguishable from random noise is especially interesting.
As it is quite hard to construct a bijective function that is indistinguishable from random noise however, we will use actual random noise instead. This of course could not be used as an actual cryptographic function as it is not reversible, but it will result in similar results using the Malviya algorithm.

The largest function I could reasonably brute-force all correlation values for is an 8-to-8-bit function, which results in the distribution of correlations shown in fig. 4.9.

As becomes apparent, especially in fig. 4.9a, the correlations seem to be normally distributed in the limit. This is not very unexpected, considering that we essentially have a sum of many binary random variables. It is a very interesting observation nonetheless, as it allows us to generalize the success rates of the Malviya algorithm to any scheme that achieves plausible deniability.

When fitting a normal distribution to the correlations for our 8-to-8-bit random function as shown in

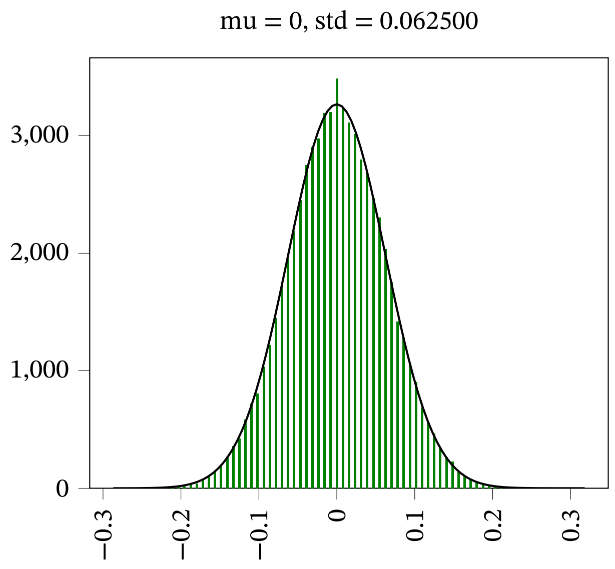


Figure 4.10: Fitting a normal distribution to the correlations of a random 8-to-8-bit function. The fitted function is in accordance with eq. (4.22).

fig. 4.10, we get the following fit:

$$f := 2^{8+1} \cdot \mathcal{N}\left(0, \left(\frac{1}{8+8}\right)^2\right) \tag{4.22}$$

Here $\mathcal{N}\left(\mu, \sigma^2\right)$ denotes the normal distribution with mean μ and variance σ^2:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Choosing high input and output lengths for our random function is important, as the normal distribution only approximates the actual distribution of correlations for large enough input and output lengths, so to validate our hypothesis from fig. 4.3 choosing $n_x = n_y = 8$ is a good choice.
Unfortunately this does not allow distinguishing which length was responsible for which parameter in eq. (4.22). Therefor a quick empirical test[7] with $n_x = 7$ and $n_y = 5$ was done, shown in fig. 4.11 which resulted in the following fit:

$$f := 2^{7+1} \cdot \mathcal{N}\left(0, \left(\frac{1}{7+5}\right)^2\right) \tag{4.23}$$

Allowing us to generalize[8] the fit to

$$f := 2^{n_x+1} \cdot \mathcal{N}\left(0, \left(\frac{1}{n_x+n_y}\right)^2\right). \tag{4.24}$$

This would be a very useful result as it allows us to efficiently calculate how many approximations with a given correlation and therefor goodness exist in a random function of a given size. This makes

[7] Many more tests with n_x and n_y ranging between 3 and 8 were made to confirm the result, but the graphics are not included in this thesis, but can be simply be generated via the provided script.

[8] This is only a hypothesis, but the empirical evidence is strong. Future work could probably analytically verify (or deny) that result.

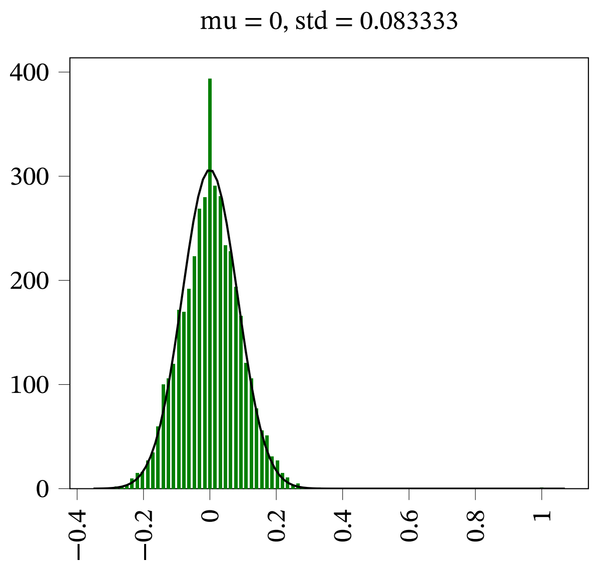


Figure 4.11: Fitting a normal distribution to the correlations of a random 7-to-5-bit function. The fitted function is in accordance with eq. (4.23).

it possible to give a much more efficient formula for the linearity of $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$:

$$\begin{aligned}
\mathcal{L}_{\mathbb{a}}(f) &\approx \sum_{c=-2^{n_x-2}}^{2^{n_x-2}} \left(\mathbb{a}\left(\frac{|c|}{2^{n_x-1}}\right)\right) \cdot \left(2^{n_x+1} \cdot \frac{1}{\frac{1}{n_x+n_y}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\frac{c}{2^{n_x-1}}-0}{\frac{1}{n_x+n_y}}\right)^2}\right) \\
&= \sum_{c=-2^{n_x-2}}^{2^{n_x-2}} \mathbb{a}\left(\frac{|c|}{2^{n_x-1}}\right) \cdot 2^{n_x+1} \cdot \frac{n_x+n_y}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c\cdot(n_x+n_y)}{2^{n_x-1}}\right)^2} \\
&= 2\sum_{c=0}^{2^{n_x-2}} \mathbb{a}\left(\frac{c}{2^{n_x-1}}\right) \cdot 2^{n_x+1} \cdot \frac{n_x+n_y}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c\cdot(n_x+n_y)}{2^{n_x-1}}\right)^2} \\
&= 2 \cdot 2^{n_x-1} \sum_{c=0}^{2^{n_x-2}} \mathbb{a}\left(\frac{c}{2^{n_x-1}}\right) \cdot 2^{n_x+1} \cdot \frac{n_x+n_y}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c\cdot(n_x+n_y)}{2^{n_x-1}}\right)^2} \cdot \frac{1}{2^{n_x-1}} \\
&\approx 2^{n_x} \int_0^1 \mathbb{a}(c) \cdot 2^{n_x+1} \cdot \frac{n_x+n_y}{\sqrt{2\pi}} e^{-\frac{1}{2}(c\cdot(n_x+n_y))^2}\, dc \\
&= 2^{2n_x+1} \cdot \frac{n_x+n_y}{\sqrt{2\pi}} \int_0^1 \mathbb{a}(c) \cdot e^{-\frac{1}{2}(c\cdot(n_x+n_y))^2}\, dc
\end{aligned}$$

This allows us to directly calculate the success probability[9] of the Malviya algorithm for any random function $f : \mathbb{F}_2^{n_x} \mapsto \mathbb{F}_2^{n_y}$:

$$2^{2n_x-n_y+1} \cdot \frac{n_x+n_y}{\sqrt{2\pi}} \int_\tau^1 c^2 \cdot e^{-\frac{1}{2}(c\cdot(n_x+n_y))^2}\, dc \approx \mathcal{L}_{\mathbb{p}_\tau}(f)$$

This can be calculated using wolframalpha. I will omit the output here.
Unfortunately this integral can not be calculated analytically, but it can be approximated numerically. However, doing this for $n_x = n_y = 128$ and $\tau = 0$, I would expect the result to equal 1, which is not the case. This means the hypothesis proposed in eq. (4.24) is wrong[10]. A Finding the correct formula will

[9]The notation $\mathcal{L}_{\mathbb{p}_\tau}(f)$ will be introduced in the following section.
[10]Instead, replacing $2^{2n_x-n_y+1}$ with $2 \cdot (n_x+n_y)^2$ yields the expected results, but this might be a coincidence.

be very interesting future work as it later allows to approximate runtime of the Malviya algorithm for schemes that require to allow *plausible deniability* as a cryptographic criterion.

4.3.5 In general

As seen in eq. (4.9) we measure any linear relation $\alpha\gamma\beta$ with probability

$$p_{\alpha,\gamma,\beta} = \frac{|c(\bar{t}_{\alpha+\gamma,\beta}(f))|^2}{2^{n_c}} \tag{4.25}$$

Definition 4.3.13. *We call an algorithm successful iff we measure any non-trivial (non-zero) approximation that holds true (or false) with probability* $\frac{1}{2} \pm \tau$

Let $\mathcal{S}$ be the set of all linear approximations that result in a successful algorithm if measured

$$\mathcal{S} = \left\{ (\alpha \parallel \gamma \parallel \beta) \,\middle|\, \beta \neq 0, |P(\langle(\alpha \parallel \gamma \parallel \beta)|(m \parallel k \parallel c)\rangle = 1) - 1/2| \geq \tau \right\}$$

We can now relate that to the "approximatibility" of f as defined in section 4.3.2.

Lemma 4.3.14. *For any* $f : \mathbb{F}_2^{n_m} \times \mathbb{F}_2^{n_k} \mapsto \mathbb{F}_2^{n_c}$ *and* $f'(a + b) := f(a, b)$ *it holds that*

$$|\mathcal{S}| = \mathcal{L}_{\delta_{\tau*2^{n_c}}}(f')$$

Note that we used $\mathcal{L}$ *not the normalized* $\mathcal{L}^*$*.*

Proof.

$$\begin{aligned}
&\left|P(\langle(\alpha \parallel \gamma \parallel \beta)|(m \parallel k \parallel c)\rangle = 1) - \frac{1}{2}\right| \geq \tau \\
\Leftrightarrow &\left|\frac{|c(\bar{t}_{\alpha+\gamma,\beta}(f'))|}{2^{n_c}} + \frac{1}{2} - \frac{1}{2}\right| \geq \tau \\
\Leftrightarrow &\frac{|c(\bar{t}_{\alpha+\gamma,\beta}(f'))|}{2^{n_c}} \geq \tau \\
\Leftrightarrow &|c(\bar{t}_{\alpha+\gamma,\beta}(f'))| \geq 2^{n_c}\tau \\
\Leftrightarrow &\delta_{\tau*2^{n_c}}\left(|c(\bar{t}_{\alpha+\gamma,\beta}(f'))|\right) = 1
\end{aligned}$$

□

Because better approximations get measured more likely, the success probability of Malviya algorithm is lower bounded by

$$P(''success'') \geq \frac{\mathcal{L}_{\delta_{\tau*2^{n_c}}}(f')}{2^{n_m+n_k+n_c}}.$$

A better, more complicated bound is given by the following theorem:

Theorem 4.3.15. *The success probability of Malviya algorithm is bounded by*

$$P(''success'') \geq \sum_{(\alpha\parallel\gamma\parallel\beta)\in\mathcal{S}} p_{\alpha,\gamma,\beta}$$

I.e. the sum of all probabilities of measuring a linear approximation that results in a successful algorithm. This success-rate can be directly converted into a measure of the "approximateability" of f (similar to those defined in section 4.3.2) by using

$$\mathbb{a}(x) = \mathbb{p}_\tau(x) := \begin{cases} 0 & \text{if } x < \tau \\ \frac{x^2}{2^{n_y}} & \text{else} \end{cases}$$

Corollary 4.3.15.1. *The success probability of Malviya algorithm is given by*

$$P(''success'') = \mathcal{L}_{\mathbb{p}_\tau}(f') = \sum_{\substack{m_x, m_y \in \mathbb{F}_2^n \\ m_y \neq 0}} \mathbb{p}_\tau \left(|c(\bar{t}_{m_x, m_y}(f))| \right)$$

Chapter 5

Applying Quantum Search

As this success probability is not good enough for most purposes, we will now look at how to increase it. The most obvious way to do this is to simply repeat the algorithm multiple times and take the best result. An obstacle using this approach is though, that it is not easy to verify or compare which result was best (compare appendix A.1). Besides, this is neither very efficient as its success probability only increases linearly with the amount of repetitions, nor does it guarantee a good result as it is still probabilistic.

5.1 Grover Algorithm for Quantum Search

In some sense, we search for a specific input (masks) that results in a good approximation. The search problem is famously solved by Grovers algorithm[1] [26] which achieves a quadratic speedup compared to the classical case. The classical case needs $\mathcal{O}(N)$ queries to find the solution, while Grover only needs $\mathcal{O}\left(\sqrt{N}\right)$ queries. This cannot be achieved faster for black-boxed quantum oracles as shown by the following lower bound.

5.1.1 Lower Bound for Quantum Search

As Grover's algorithm finds a marked state in a superposition of all possible states, it needs some way of knowing which state is marked. This is done by applying an oracle that marks this state by inverting its amplitude, compare with V_f in section 3.4.1.1. This oracle will simply be called *the oracle* in this section.

Theorem 5.1.1 (Lower Bound for Quantum Search [11]). *Any quantum algorithm that solves the search problem with certainty needs at least $\Omega(\sqrt{N})$ applications of the oracle.*

There are multiple different proofs for this theorem, e.g. [22] and [42, pp. 269-271].

We will not go into detail on the proof in this thesis, but as a sketch the proof consist of two parts:

1. The first part shows that to distinguish between N elements with certainty, it has to hold that $\sum_{\bar{x}\in\{0,1\}^n} \| |\psi_k^{\bar{x}}\rangle - |\psi_k\rangle\|^2 \equiv \Omega(N)$, where $|\psi_k^{\bar{x}}\rangle$ denotes the state after running an algorithm using k applications of the oracle (with$\bar{x}$ being the marked element) and $|\psi_k\rangle$ denotes the same algorithm, but where the oracle has been replaced by the identity, i.e. no element is marked.
2. The second part shows that this difference is at most $4k^2$.

[1]As Grovers Algorithm is only a special case, that is not applicable in this thesis, of Amplitude Amplification, which will be described in section 5.2, a description of the Grover algorithm will be omitted here.

Remarks on the lower bound regarding the complexity model

This however uses the oracle model (compare section 3.4.2.1) and therefor misses some possible considerations. As quoted from [23, p.145]:

> Remark: It is important to emphasize that concerning the scenario when the database is virtual (e.g. a given function defines it and we do not need to build special hardware to access the database) we are interested rather in the number of elementary gates implementing the counting than in the number of applied Grover operators. As we learned at the end of Section 6.2.4 the complexity can be expressed with $\mathcal{O}\left(\log^3(N)\right)$ gates, which is a significantly favorable metrics for complexity!

This however only holds for specific cases where U_f can be efficiently exponentiated, which is the case for trivial and known ${U_f}$[2], but neither in general nor if U_f is given as a black-box oracle.

If the implementation of U_f is given this might be useful[3] as it allows to optimize the circuit for the specific U_f. It might be the case though, that Grover oracles which mark only a single element cannot be squared sub-linearly, or more generally the more elements are marked the more gates can be eliminated. This hypothesis was not researched in this thesis, but might be a topic for future research.

But a sublinear squaring of U_f would be considered an a priori promise to f as it requires some special properties that can then be exploited, if no promises are given any quantum algorithm can only achieve a polynomial speedup compared to the classical case[9].

5.2 Amplitude Amplification

Grover's algorithm can only be used to find a single marked element, but as we are looking for multiple good approximations, we need a more general approach. Fortunately this exists and is called *Amplitude Amplification* [14]. It is a generalization of Grover's algorithm and can be used to amplify the amplitude of any state in a superposition. To be more precise, it can solve the following problem in $\mathcal{O}\left(\sqrt{N}\right)$ oracle calls[31, p. 146]:

Problem 5.2.1 (Boosting). *Given a measurement-free quantum algorithm represented by a unitary operator $A \in \mathbb{C}^{2^n \times 2^n}$ as an oracle and a Boolean function $\rho : \mathbb{F}_2^n \to \mathbb{F}_2$ partitioning the canonical basis vectors of $\mathbb{C}^{2^n}$. Let $|\Psi\rangle := A|0\rangle$ be the output state after running the Algorithm, $P_i := \sum_{k\in\rho^{-1}(i)} |k\rangle\langle k|$ for $i \in \mathbb{F}_2$ the projectors corresponding to these partitions and $a := \langle\psi_{good}|\psi_{good}\rangle := \langle\Psi| P_1^\dagger P_1 |\Psi\rangle \in [0,1]$ the success probability of A. We are looking for an algorithm that boosts that probability to* 1.

Solving this problem the Amplitude Amplification algorithm uses the following efficent unitarity operations:

Definition 5.2.1. *The operators $Q, R_{2^n}, V_\rho \in \mathbb{C}^{2^n\times 2^n}$ are defined for any $x \in \{0, 1, ..., 2^n - 1\}$ and unitary operator $A \in \mathbb{C}^{2^n\times 2^n}$ over*

$$Q := -A^{-1}R_{2^n}AV_\rho,\ R_{2^n}|x\rangle := \begin{cases} -|x\rangle & x = 0 \\ |x\rangle & x \neq 0 \end{cases} \text{ and } V_\rho|x\rangle := (-1)^{\rho(x)}|x\rangle. \tag{5.1}$$

Where $-A^{-1}R_{2^n}A$ is the generalization of the Grover diffusion operator D, that does not depend on the current search problem and V_ρ is the interesting part, marking which states are "good", by flipping their amplitude. Creating V_ρ is the main challenge and the focus of section 5.4.

The diffusion operation, and therefor A, on the other hand is interesting as well, it plays the following role in the effect of Q [14]:

[2]Using a square and multiply algorithm for efficient exponentiation. This however requires U^2 to have the same complexity (regarding amount of gates) as U, or at least have a sub-linear relation, which is usually not the case.
[3]Although the optimized `qiskit.transpile` function does not yield any benefits in my small empirical study.

$$Q\,|\psi_{\text{good}}\rangle = (1-2a)\,|\psi_{\text{good}}\rangle - (2a)\,|\psi_{\text{bad}}\rangle$$
$$Q\,|\psi_{\text{bad}}\rangle = 2(1-a)\,|\psi_{\text{good}}\rangle + (1-2a)\,|\psi_{\text{bad}}\rangle$$

where $|\psi_{\text{good}}\rangle$ and $|\psi_{\text{bad}}\rangle$ are the good and bad states in accordance to eq. (5.3) respectively and a is the success probability of A, i.e. measuring a "good" state, as also used in problem 5.2.1.

Resulting in the following equation when applying Q multiple times [15]:

$$Q^j A\,|0\rangle = a_j^{\text{good}}\,|\psi_{\text{good}}\rangle + a_j^{\text{bad}}\,|\psi_{\text{bad}}\rangle$$

with

$$a_j^{\text{good}} = \frac{1}{\sqrt{a}}\sin\big((2j+1)\arcsin(\sqrt{a})\big) \qquad \text{and therefor} \qquad a_j^{\text{bad}} = \frac{1}{\sqrt{1-a}}\cos\big((2j+1)\arcsin(\sqrt{a})\big).$$

With that knowledge we can repeat Q a fixed amount of times when we know a, if that is not the case, we can apply something similar to exponential search resulting in the algorithm shown in algorithm 4.

```
Algorithm 4: Amplitude Amplification
  input : A unitary operator $A \in \mathbb{C}^{2^n \times 2^n}$, and a Boolean function $\rho : \mathbb{F}_2^n \to \mathbb{F}_2$. The success
          probability $a = \langle 0|\, A^\dagger P_1^\dagger P_1 A\,|0\rangle \in (0,1)$ of $A$ is not known.
  $\ell \leftarrow 0, z \leftarrow 0$ and choose $c \in (1,2)$;
  while $\rho(z) \neq 1$ do
      $\ell \leftarrow \ell + 1; M \leftarrow \lceil c^\ell \rceil$;
      $|\psi\rangle \leftarrow A\,|0\rangle$;
      choose integer $j \in [1, M]$ at random ;
      $|\psi\rangle \leftarrow Q^j\,|\psi\rangle$;
      Measure $|\psi\rangle$ with respect to the standard basis to get $z$;
  end
  output: $z$
```

This algorithm is inherently probabilistic, but to halt and get the expected result it needs an expected runtime of $\mathcal{O}\left(\sqrt{\frac{1}{a}}\right)$ (in the oracle-model) [14].

5.2.1 Eliminating the Trivial Solution

As Malviya et al. stated in their paper, the probability of measuring the trivial approximation with a chance of 6.25% was already quite high. With the hypothesis of this chance increasing with the amount of qubits, as other approximations get worse while this stays perfect, this would not be a good sign for the success rate of the algorithm.
Luckily this hypothesis is false as shown in section 4.3.1. It decreases exponentially with the amount of output bits of f, i.e. for $f : \mathbb{F}_2^{n_m+n_k} \mapsto \mathbb{F}_2^{n_c}$ the chance of measuring the trivial approximation after running the Malviya algorithm is $1/2^{n_c}$.

As this is nearly negligible, we will not go into much detail here, but we could further decrease this chance by using something similar to an inverse Grover that decreases the amplitude of a selected state instead of increasing it. Fortunately with the knowledge of amplitude amplification, an inverse Grover is easy to construct as amplitude reduction and amplification is congruent to each other by simply swapping the good and bad states.
As the good states get pushed to certainty, the bad states get squished to amplitude zero, which is exactly what we want.

As the success probability of the single bad state[4] (the trivial mask) and therefor $a = 1 - 1/2^{n_c}$ is known, we can employ a generalized version of amplitude amplification to increase the amplitude of all good states (except the trivial mask) and therefor decrease the amplitude of the bad trivial mask-state to zero. According to Brassard et al. the algorithm starts similar to algorithm 4 with

$$j := \left\lfloor \frac{\pi}{4 \arcsin(\sqrt{a})} - \frac{1}{2} \right\rfloor$$

Applications of Q, which in this case (since $n \geq \frac{\log}{\frac{4}{3}} \log 2$) is always zero, and then continues with a generalized step, that also needs our subroutine unitaries to be generalized:

Definition 5.2.2. *The operators $Q^{(\phi,\varphi)}, R_{2n}^{(\phi)}, V_\rho^{(\varphi)} \in \mathbb{C}^{2^n \times 2^n}$ are defined for any $x \in \{0, 1, ..., 2^n - 1\}$ and unitary operator $A \in \mathbb{C}^{2^n \times 2^n}$ over*

$$Q^{(\phi,\varphi)} := -A^{-1} R_{2n}^{(\phi)} A V_\rho^{(\varphi)}, \ R_{2n}^{(\phi)} |x\rangle := \begin{cases} e^{i\phi} |x\rangle & x = 0 \\ |x\rangle & x \neq 0 \end{cases} \text{ and } V_\rho^{(\varphi)} |x\rangle := \begin{cases} e^{i\phi} |x\rangle & \rho(x) = 1 \\ |x\rangle & \rho(x) = 0 \end{cases}. \quad (5.2)$$

With this partial applications of Q are possible. Choosing φ and ϕ s.t. [14, eq. 11]

$$e^{i\varphi}(1 - e^{i\phi})a = a - (a - 1)e^{i\phi}$$

holds, allows us to eliminate the trivial approximation with a single $Q^{(\phi,\varphi)}$-oracle call.
This is $\in \mathcal{O}(1)$ and guarantees that the trivial approximation is not measured, but it does not guarantee that the algorithm is successful, as other bad non-trivial approximations still have non-zero amplitude.

5.3 Amplitude-Amplification of "good" Approximations

Eliminating only one, known bad state as described in the previous section is a rather simple procedure. But for a successful algorithm this would be insufficient, as there are still other bad states that can be measured.

As eliminating all "bad" states is equivalent to amplifying all "good" states, we will from now on focus on the latter. The amplitude amplification algorithm requires a partition function ρ that partitions the canonical basis vectors of $\mathbb{C}^{2^n}$ into good and bad states. As the goodness of a state is defined by the approximation (m_x, m_y) it represents, whose goodness is well-defined via $|c(\bar{t}_{m_x,m_y}(f))|$, we can reuse[5] our threshold activation function δ_τ as partition[6] function ρ. This allows us to employ amplitude amplification, boosting the amplitudes of states that represent approximations with a high enough correlation, while eliminating the bad states, i.e. approximations with a correlation lower than τ.

Creating an oracle that does something similar to $|x\rangle \mapsto (-1)^{\delta_\tau |c(\bar{t}_{m_x,m_y}(f,x))|} |x\rangle$ to be used in amplitude amplification efficiently is not trivial, and will be the focus of the following section.

5.4 Creating the Oracle

As mentioned in section 5.2 the amplification operator can be split into the labelling and diffusion operators. While the diffusion operator is the same for every oracle, the labelling operator is different for every oracle and needs to be constructed individually depending on which elements are "good".

[4]There might be multiple bad states, but only one with non-zero probability.
[5]Obviously the codomain has to be changed from $\mathbb{R}$ to $\mathbb{F}_2$, but as the image of δ_τ only has cardinality 2 that is straight forward.
[6]It is possible to not use partition functions for amplitude amplification, but directly use the goodness of an approximation and boost those more, that have higher goodness, as described in [48], but that is out of the scope for this thesis and might be part of future work.
[6]Most graphics in this section where generated with my own code, provided at https://github.com/HannesGitH/MA-public/blob/main/creating_the_oracle_stuff.ipynb.

We have already established, that good states are those in $\mathcal{S}$, i.e. those with $\delta_\tau\left(\left|c(\bar{t}_{\alpha+\gamma,\beta}(f'))\right|\right) = 1$, but equivalently we can define that η is a good approximation of a given function f if it holds true for a certain amount of inputs.

As visible in fig. 5.2 a possible labelling operator then consists of two parts:

1. A counting oracle $U_\#$ * that counts for how many inputs the approximation holds true.
   This is similar to calculating the correlation.
2. A marking oracle U_ε that decides whether this count is greater than a threshold ε or not and marks the corresponding elements accordingly by inverting an ancilla qubit, which in turn triggers a z-flip labelling of elements that are "good" while leaving "bad" ones unchanged.
   This is similar to using δ_τ on the previously calculated correlation.

The marking oracle is easily constructed by using the quantum variant of the classical digital comparator as shown in fig. 5.1.

Constructing the counting operator on the other hand is more complex and the main focus of the following sections, providing different approaches to construct it. For ease of readability they will first be introduced in a simplified non-vectorial setting, before being extended to the full cryptographic setting in section 5.5.

5.4.1 Using basic quantum counting

The most straight forward way to construct the labelling operator is to use quantum counting. The $U_\#$-gate (*) in fig. 5.2 would be replaced by a quantum counting algorithm that counts on how many plaintext, key, ciphertext tupels a given approximation η would hold true.

5.4.1.1 Basic Quantum Counting in general

This subsection introduces the quantum counting algorithm as described in [14], which will then transformed to be used as $U_\#$ *.

Recall, $N := 2^n$, $n \in \mathbb{N}_{\geq 1}$. For any Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ with $c := |f^{-1}(1)|$ being the count of elements on which f evaluates true, the *quantum counting* algorithm can estimate c when given the corresponding unitary oracle $U_f : \mathbb{C}^{N\cdot 2} \to \mathbb{C}^{N\cdot 2}, |x\rangle|y\rangle \mapsto |x\rangle|f(x) \oplus y\rangle$ as an input. The precision of the estimate can be tuned to a multiplicative error metaparameter ε which is antiproportional to the complexity of the quantum counting algorithm.

As f induces a partition on its domain $\mathbb{F}_2^n$ with cardinality 2, namely:

$$f^{-1}(0) := \{x | f(x) = 0\} \text{ and } f^{-1}(1) := \{x | f(x) = 1\},$$

the corresponding projectors (for "good" and "bad" elements) can be defined as:

$$P_{\text{bad}} := \sum_{x \in f^{-1}(0)} |x\rangle\langle x| \text{ and } P_{\text{good}} := \sum_{x \in f^{-1}(1)} |x\rangle\langle x|.$$

These span the 2-dimensional subspace of "bad" and "good" states:

$$|\psi_{\text{bad}}\rangle := P_{\text{bad}}|\psi\rangle \text{ and } |\psi_{\text{good}}\rangle := P_{\text{good}}|\psi\rangle; |\psi_{\text{bad}}\rangle \cup |\psi_{\text{good}}\rangle = |\psi\rangle. \tag{5.3}$$

With usage of an ancilla qubit these states are easily created with following simple projectors (the "good" states are those with ancilla being 1 and the "bad" ones those with 0):

$$P^+_{\text{bad}} := \sum_i a_i |i\rangle|0\rangle \text{ and } P^+_{\text{good}} := \sum_i a_i |i\rangle|1\rangle,$$

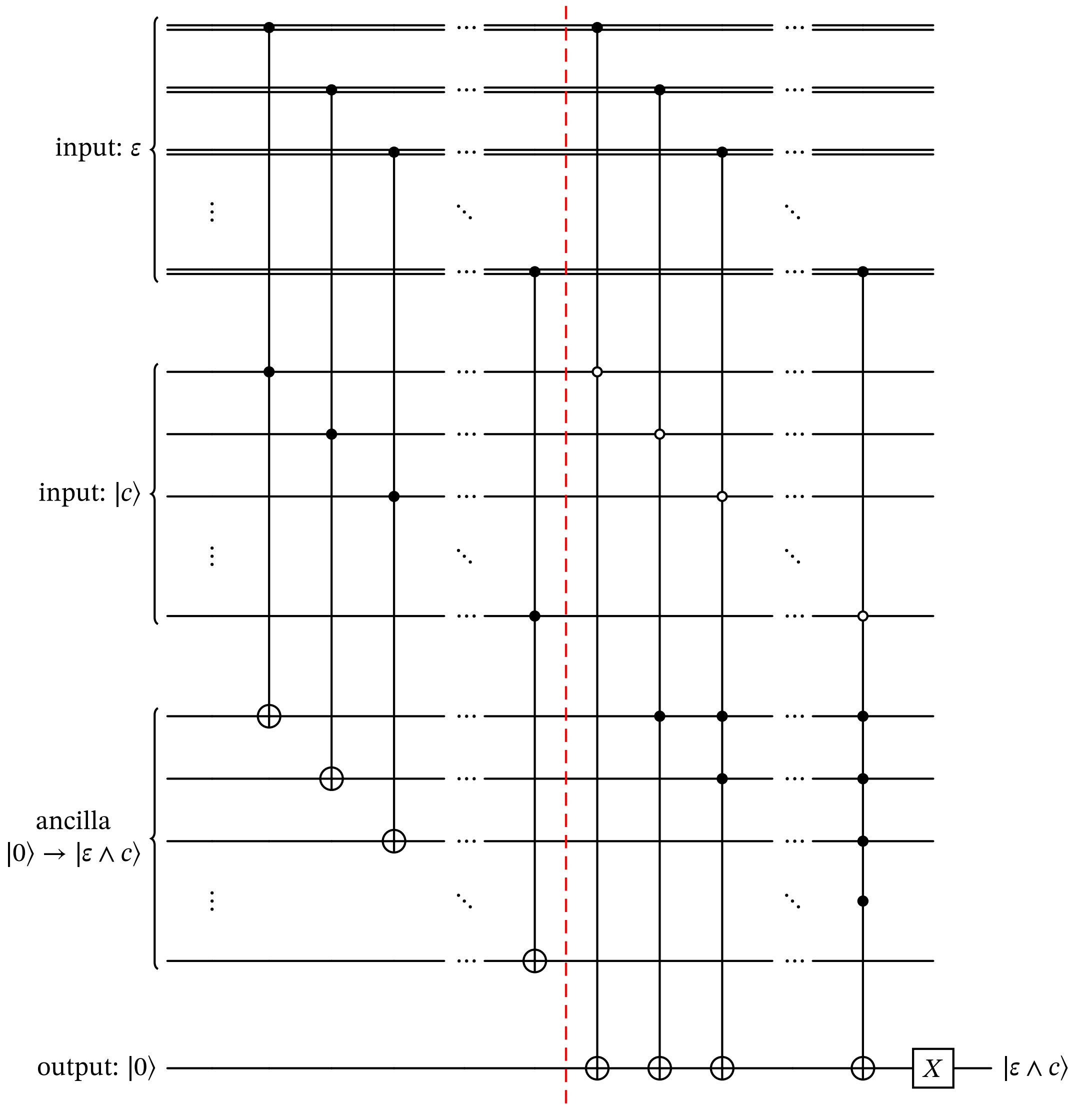

input: ε
input: $|c\rangle$
ancilla
$|0\rangle \to |\varepsilon \wedge c\rangle$
output: $|0\rangle$
X
$|\varepsilon \wedge c\rangle$

Figure 5.1: Quantum "$\geq \varepsilon$"-comparator, implementation of U_ε from fig. 5.2.
The first part (left of the slice) only fills an ancilla register with the $\varepsilon \wedge c$ bitstring. That is needed for the second part to only flip the output after the first unequal bit and not flip it back when another less significant (and therefor irrelevant) bit mismatches.
This simply shows how to realize the standard oracle definition $|\varepsilon\rangle\,|c\rangle\,|b\rangle \mapsto |\varepsilon\rangle\,|c\rangle\,|b \oplus f(\varepsilon, c)\rangle$ with $f(\varepsilon, c) = \varepsilon \wedge c$ and b being the output qubit.

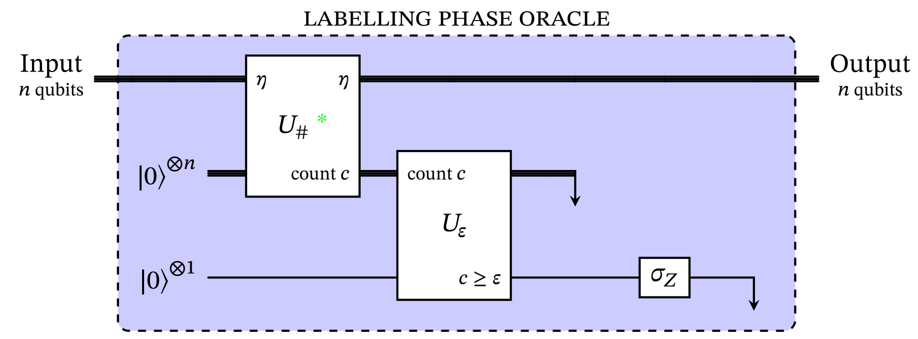


Figure 5.2: Possible structure for deciding which elements are “good” depending on whether the output c of the counting oracle $U_{\#}$ is greater than a threshold ε or not and labelling them with an amplitude inversion accordingly. This circuit is only a schematic visualization as the trashing of ancilla qubits is more complicated in reality. A more realistic implementation is depicted in fig. 5.3.

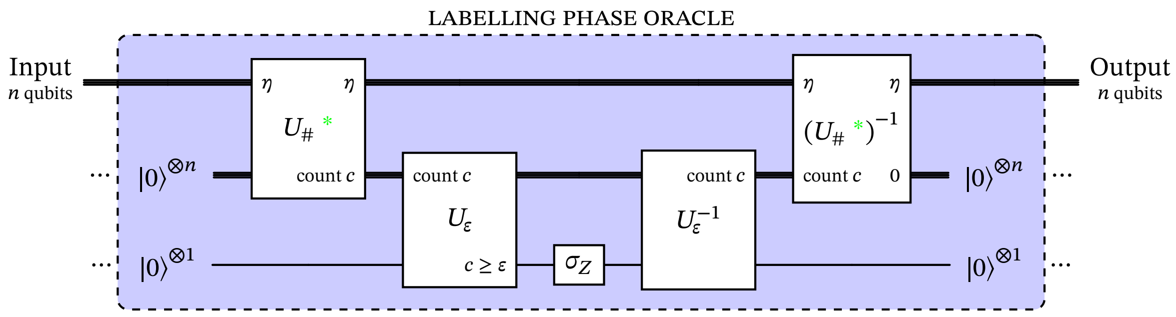
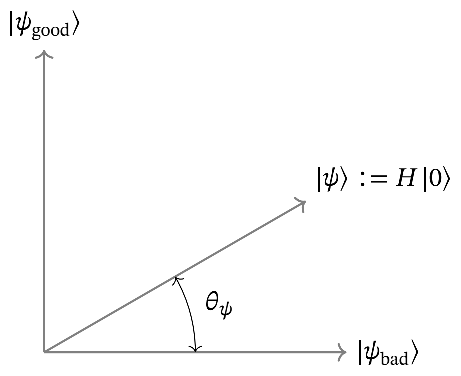


Figure 5.3: As trashing qubits, i.e. disregarding them has the same effect as measuring them, which in turn would destroy the superposition, the ancilla qubits need to be *uncomputed* to $|0\rangle$ after usage [42, 187]. This depicts how fig. 5.2 would be implemented in practice. Similar procedures have to be employed for all other circuits using the trash symbol.



$|\psi_{\text{good}}\rangle$
$|\psi\rangle := H|0\rangle$
θ_ψ
$|\psi_{\text{bad}}\rangle$

Figure 5.4: visualization of $|\psi\rangle$ as a vector in the 2-dimensional subspace of good and bad states and its relation to θ_ψ

In case of an equally distributed superposition his can be simplified to

$$P^+_{\text{bad}} := |+\rangle|0\rangle \text{ and } P^+_{\text{good}} := |+\rangle|1\rangle$$

to then result in the following identity[7]:

$$|\psi_{\text{bad}}\rangle \otimes |0\rangle = P^+_{\text{bad}} \cdot U_f \cdot (H \otimes I)|0\rangle \text{ and } |\psi_{\text{good}}\rangle \otimes |1\rangle = P^+_{\text{good}} \cdot U_f \cdot (H \otimes I)|0\rangle$$

Since

$$\langle\psi_{\text{good}}|\psi_{\text{good}}\rangle = \frac{c}{N} \tag{5.4}$$

it suffices to estimate the amplitude of $|\psi_{\text{good}}\rangle$ to get an estimate for c.

As $|\psi_{\text{good}}\rangle$ and $|\psi_{\text{bad}}\rangle$ are orthogonal, the angle θ_ψ as shown in fig. 5.4 can be described as:

$$\begin{aligned}
\theta_\psi &= \arcsin\left(\left|\langle\psi_{\text{good}}|\sqrt{N} * H|0\rangle\right|\right) && (5.5)\\
&= \arcsin\left(\left|\langle\psi_{\text{good}}|\psi_{\text{good}}\rangle\right|\right) && (5.6)\\
&= \arcsin\left(\sqrt{\langle\psi_{\text{good}}|\psi_{\text{good}}\rangle}\right) && (5.7)\\
&= \arccos\left(\sqrt{\langle\psi_{\text{bad}}|\psi_{\text{bad}}\rangle}\right) && (5.8)
\end{aligned}$$

Conveniently this angle is related to the change through the corresponding (efficiently implementable) Grover operator

$$G := D \cdot V_f$$

by a factor of two[42, p. 253]. Compare fig. 5.5. In this case $V_f|x\rangle := (-1)^{f(x)}|x\rangle$ is the z-flip-oracle corresponding to f and D is the Grover diffusion operator.

The change of angle of an eigenstate through a quantum oracle (in this case G) can be estimated by the quantum phase estimation (QPE) algorithm[8], which is described in appendix B.2.

The problem is though, that $|\psi\rangle$ is not (necessarily) an eigenstate of G. But as described in [42, pp. 224-225] the QPE algorithm can be applied to a superposition (or linear combination) of eigenstates $\{\psi_i\}$ of

[7]This is the variant of equal superposition (as caused by the H gate), but it can simply be transformed to any other superposition by applying a corresponding unitary gate instead.

[8]Using QPE for quantum counting is not a new idea as mentioned by the following quotes: “Quantum counting is an application of the phase estimation procedure of Section 5.2 to estimate the eigenvalues of the Grover iteration G”[42, p. 262]; “The goal of amplitude estimation is, in its simplest form, to estimate the unknown parameter θ contained in the state $|\phi\rangle = \sin\theta\,|good\rangle + \cos\theta\,|bad\rangle$”[49].

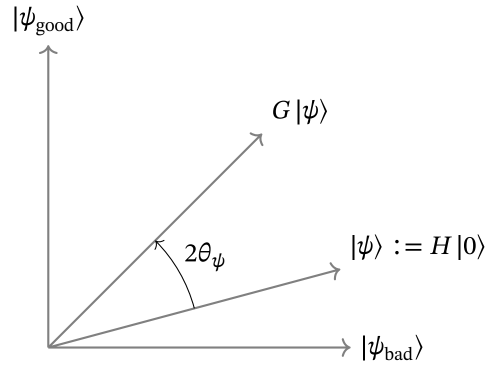

$|\psi_{\text{good}}\rangle$
$G|\psi\rangle$
$2\theta_\psi$
$|\psi\rangle := H|0\rangle$
$|\psi_{\text{bad}}\rangle$

Figure 5.5: visualization of $|\psi\rangle$ as a vector in the 2-dimensional subspace of good and bad states and its relation to θ_ψ as interpreted as the phase estimation of the corresponding Grover operator

G and ends in a state with equal superposition between their corresponding approximated eigenphases $\{e^{2\pi i\theta_i}\}$:

$$QPE\left(\sum_i \alpha_i |\psi_i\rangle\right) = \sum_i \alpha_i |\psi_i\rangle |\tilde{\theta}_i\rangle$$

The two eigenstates of G are $|\psi_{\text{good}}\rangle$ and $|\psi_{\text{bad}}\rangle$ and since they are orthagonal to each other their phases (of the corresponding eigenvalues) are negated to each other: $\theta_{\psi_{\text{good}}} = -\theta_{\psi_{\text{bad}}}$[14, p. 15]. So measuring either of these will result in knowing θ_ψ.

It has to be remarked though, that the canonical QPE outputs the eigenphase which in this case is $\frac{2\theta_\psi}{2\pi}$ due to the Grover operator rotating by 2θ and the QPE outputting the eigenphase instead of the angle (factor of 2π).

After that the quantum counting algorithm can output $c = \sin(\theta_\psi)^2 * N$ as an inversion of eq. (5.7).

5.4.1.2 Success probability

Corollary 5.4.0.1. *The success probability of the quantum counting algorithm is at least*

$$\Pr[\text{“}success\text{”}] := \Pr\left[|\tilde{c} - c| \leq \varepsilon\right] \geq 1 - \frac{1}{2^{t+1}\arcsin\left(\sqrt{\frac{\varepsilon}{\pi N}}\right) - 4}$$

Proof. [9] As seen towards the end of section 5.4.1.1 the algorithm outputs $c = \sin(\theta_\psi) * N$ and uses QPE to get $\tilde{\theta}_\psi$. QPE (compare appendix B.2) in turn has success probability

$$\Pr[\text{“success”}] := \Pr\left[|\frac{\tilde{\theta_\psi}}{\pi} - \frac{\theta_\psi}{\pi}| \leq \frac{1}{2^\tau}\right] \geq 1 - \frac{1}{2^{t-\tau+1} - 4}$$

Converting this to getting within a general error ε results in

$$\Pr[\text{“success”}] := \Pr\left[|\tilde{\theta_\psi} - \theta_\psi| \leq \varepsilon\right] \geq 1 - \frac{1}{2^{t+1}\frac{\varepsilon}{\pi} - 4}$$

Directly applying that to our case of $c = \sin(\theta_\psi) * N$ results in

$$\Pr[\text{“success”}] := \Pr\left[|\arcsin\left(\sqrt{\frac{\tilde{c}}{N}}\right) - \arcsin\left(\sqrt{\frac{c}{N}}\right)| \leq \varepsilon\right] \geq 1 - \frac{1}{2^{t+1}\frac{\varepsilon}{\pi} - 4}$$

[9]This bound (as shown in fig. 5.6) is much tighter than the previously known bound provided by Brassard et al. (corollary 5.4.0.2), this seems highly unlikely, so chances are this proof might not be correct.

Since arcsin and the square root is monotonically increasing between 0 and 1 and $0 \leq \frac{c}{N} \leq 1$ we can apply it to the inequality and get

$$\Pr[\text{“success”}] := \Pr\left[|\tilde{c} - c| \leq \varepsilon\right] \geq 1 - \frac{1}{2^{t+1}\arcsin\left(\sqrt{\frac{\varepsilon}{\pi N}}\right) - 4}$$

□

This is already a useful result that allows to balance success probability and precision by choosing the number of counting qubits t and the error ε accordingly. Additionally, the runtime is directly influenced by t[10], so the runtime can be tuned as well.

As this bound is suspiciously more tight than the one previously provided by Brassard et al., it is worth comparing them und using their estimate as well.

Corollary 5.4.0.2. *The success probability of the quantum counting algorithm is also at least*

$$\Pr[\text{“success”}] := \Pr\left[|\tilde{c} - c| \leq 2\pi k\frac{\sqrt{c(N-c)}}{2^t} + \pi^2 k^2 \frac{N}{2^{2t}}\right] \geq 1 - \frac{1}{2k-2}$$

Proof. [14, p. 16] where $M > j$ is a strict upper bound on the maximum exponentiation of our oracle: U^j [14, pp. 15-16] as used in fig. B.2. Therefor $M = 2^t$. □

As shown in fig. 5.6 this bound is worse than my bound. It has the additional disadvantage, that it depends on the actual value of c which is unknown to the attacker and is parameterized by k which is neither directly the success probability nor the precision.

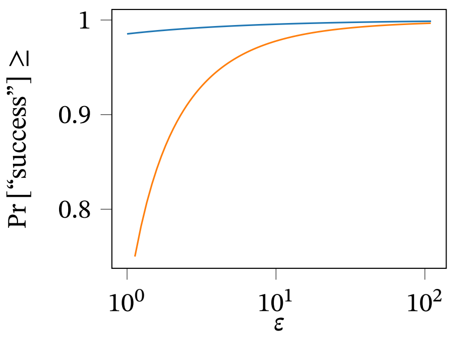


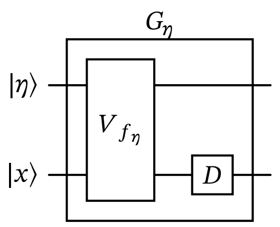
Figure 5.6: Comparison of Brassard et al.'s probability bound (orange) and mine (blue): Absolute error vs. success probability with 2 additional counting bits and $N = 256, c = 15$.

5.4.1.3 Quantum Counting Linear Approximation Goodness

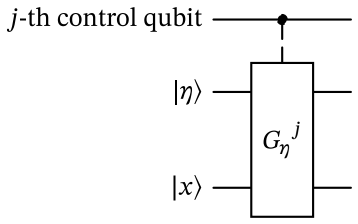
In our case we do not want to check any specific function, rather a family, parameterized by η: $f_\eta(x) := \langle \eta | x \rangle$ and get its count $c_\eta := |f_\eta^{-1}(1)|$.

So instead of the normal j-controlled $(DV_f)^j$ used in quantum counting we use a bigger $((I \otimes D)V_{f_\eta})^j$ with $V_{f_\eta} |\eta\rangle |x\rangle \mapsto (-1)^{\langle \eta | x \rangle} |\eta\rangle |x\rangle$, as shown in fig. 5.7.

[10] As mentioned in section 5.4.1.4 it depends on the attack model but in most cases it has to be considered $\mathcal{O}(2^t)$.



(a) Circuit for $G_\eta := (I \otimes D)V_{f_\eta}$



(b) Circuit for j-controlled G_η

Figure 5.7: Circuit for $((I \otimes D)V_{f_\eta})^j$

And therefor the Δ used in the QPE process will also do the transform $|j\rangle\,|\eta\rangle\,|x\rangle \mapsto |j\rangle\,(G_\eta^j\,|\eta\rangle\,|x\rangle)$.

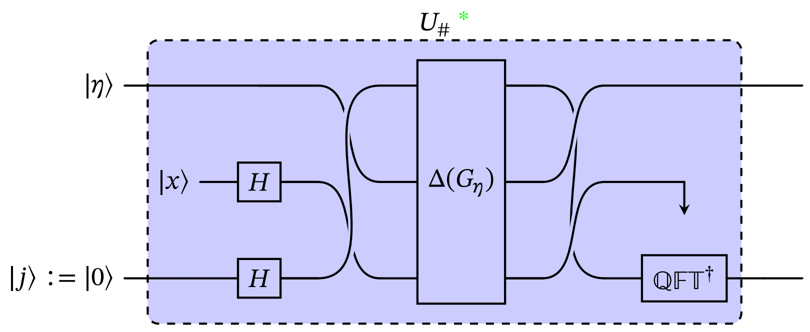


Figure 5.8: Circuit for $U_\#{}^*$ using quantum counting

An analysis will be omitted here, but the effect of applying this new $U_\#{}^*$ will be $|\eta\rangle\,|0\rangle \mapsto |\eta\rangle\,|c_\eta\rangle$ with the same complexity as in the general case, except the need of an additional register of length n.

5.4.1.4 Oracle access and runtime

As also mentioned in lemma B.2.2 the runtime of this basic quantum counting algorithm greatly depends on the chosen attack model. Three models come to mind, which are listed in table 5.1.

Name	Description	Attacker can access	Runtime in oracle-model
A0	The simplest and arguably most straight-forward form is when the attacker can only access U itself. This results in the weakest security-model as the attacker is quite restricted.	U	$\mathcal{O}(2^t)$
A2	The most loose form is when the attacker can directly access the conditionally exponentiated oracle $\Lambda(U)$ as defined in definition B.2.1.	$|j\rangle\,|y\rangle \mapsto |j\rangle\,(U^j\,|y\rangle)$	$\mathcal{O}(1)$
A1	In another variant that sits in between the previous, more obvious ones, the attackee has to provide all U^{2^i}. This would allow to skip the lower k exponentiations $\{U^{2^0}, \cdots, U^{2^{k-1}}\}$ with a loss of precision in exchange for a direct linear speedup resulting in a runtime of $\mathcal{O}(t-k)$.	$i \mapsto U^{2^i}$	$\mathcal{O}(t)$

Table 5.1: Multiple attack models on exponentiating U and their corresponding runtimes.

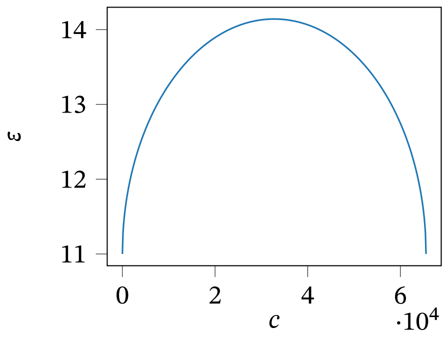
If, rather arbitrarily, the counting register[11] is chosen to be half the size of the function register[12], the

[11] Compare estimation register in fig. B.1.
[12] Also compare fig. B.1.

error results to be [14, p. 21] :

$$|\tilde{c} - c| = \varepsilon_{t:=\frac{n}{2}} < 2\pi\sqrt{\frac{c(N-c)}{2^n}} + 11.$$

A short analysis of this choice is depicted in fig. 5.9.



(a) Chart of upper error bound in relationship to the actual count c for $N = 65536$.

Attack Class	Runtime with $\varepsilon_{t:=\frac{n}{2}}$
A0	$\mathcal{O}\left(2^{\frac{t}{2}}\right) = \mathcal{O}\left(\sqrt{N}\right)$
A1	$\mathcal{O}\left(\frac{n}{2}\right)$
A2	$\mathcal{O}(1)$

(b) Runtime table for a counting register half the size of the function register.

Figure 5.9: Analysis of error and runtime when choosing $t := \frac{n}{2}$.

5.4.2 Optimized quantum counting

The next idea, proposed by Brassard et al. in [14], is, that in many cases the exact value of c is not needed, but only an approximation. It is based on the observation that $\tilde{c}$ is likely zero iff t is chosen s.t. $2^t \in o\left(\sqrt{\frac{N}{c}}\right) \Leftrightarrow t \in o(n - \log(c))$ or in other words the approximated count is expected to be zero iff the counting register is less than half the size of the function register minus the logarithm of the actual count.

So the algorithm outputs zero if the actual count is overestimated and the counting register is not precise enough to detect it.

This can be used to optimize the previous algorithm by choosing t s.t. it is just big enough to detect the actual count. As c is unknown though, t is chosen to be 0 and linearly increased, s.t. the approximatible count increases exponentially, similar to exponential search. This new algorithm is described at the top of [14, p. 22] and will be called the *basic approximate count algorithm* here as well.

Unfortunately this algorithm uses measurements in between and therefor can not be simply represented as a unitary operator. Therefor it and its derivations can not be used as an oracle as they are currently defined in a straight forward manner. But as it can still be used as-is to e.g. verify the "goodness" of a specific approximation (as done in appendix A.1) it will still be covered here for completeness reasons.

Theorem 5.4.1. *The basic approximate count algorithm outputs an approximation $\tilde{c}$ to c with error ε with probability at least $\frac{2}{3}$ using a runtime in*

$$\Theta\left(\frac{\sqrt{cN}}{\varepsilon}\right)$$

in the A0 model.

Proof. Follows from Theorem 15 in [14, p. 22] by substituting their ε with our $\frac{\varepsilon}{c}$ (and their t with our c). □

Remark 5.4.2. *This algorithm is optimal for any fixed ε [14, p. 23].*

They also provided a generally optimal algorithm (not only for fixed ε) for approximate quantum counting which will not be detailed here as its runtime of

$$\Theta\left(\sqrt{\frac{N}{\lfloor\varepsilon\rfloor+1}}+\sqrt{\frac{c(N-c)}{\lfloor\varepsilon\rfloor+1}}\right)$$

in the A0 model is only marginally better, as visible in fig. 5.10.

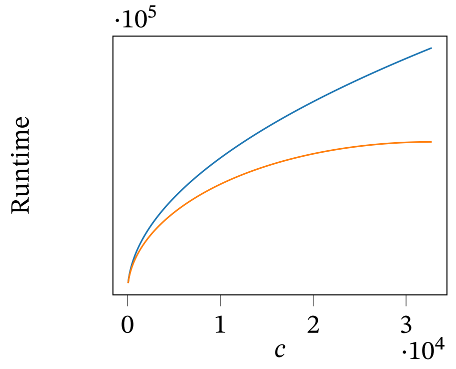


Figure 5.10: Chart of runtime of Brassard et al.'s approximate counting algorithm in relation to the actual count c for $N = 65536$ and $\varepsilon = 25$. The here discussed variant in blue and the optimized variant in orange.

As a special case of this approximate counting it might be interesting to consider the case of $\varepsilon = 0$ [13] i.e. exact counting. Their runtimes are depicted in fig. 5.11.

Weirdly the exact counting algorithm provided by Brassard et al. performs asymptically worse in relation to c than both their optimized and unoptimized approximate counting algorithms with a nearly zero error. In relation to N they all perform similarly.

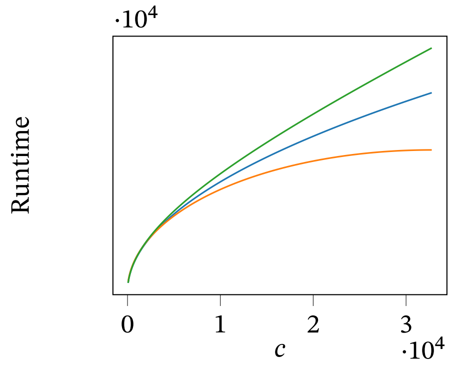
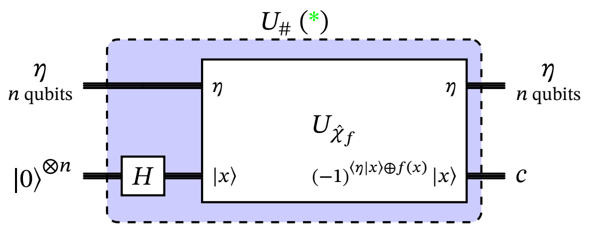


Figure 5.11: Chart of runtime of Brassard et al.'s approximate counting algorithm in relation to the actual count c for $N = 65536$ and $\varepsilon = 0.35$. The here discussed variant in blue and the optimized variant in orange. This time compared to the runtime of the exact counting algorithm [14, pp. 23-24] in green.

[13]The unoptimized variant has $\varepsilon > 0$ as a precondition, but as c is an integer we can set $\varepsilon := \frac{1}{3} \leq \frac{1}{2}$ to still get the correct result. Similar with the optimized algorithm which has a precondition of $\frac{c}{3N} < \varepsilon$, also allowing $\varepsilon := \frac{1}{3} \leq \frac{1}{2}$ since $c \leq N$ to get the correct integer.



(a) Possible Circuit for $U_{\#}$

$$U_{\hat{\chi}_f} : |\eta\rangle |x\rangle \mapsto |\eta\rangle (-1)^{\langle \eta | x\rangle \oplus f(x)} |x\rangle$$

(b) Definition of $U_{\hat{\chi}_f}$

Figure 5.12: $U_{\#}$ with a direct Walsh transform

5.4.3 Standalone oracle

As an alternate approach to the rather inefficient[14] quantum counting, we could replace the counting oracle $U_{\#}$ (*) with a custom-made standalone oracle specific for this use-case.

5.4.3.1 Direct Walsh transform

An important observation is, that according to corollary 3.2.3.1 the "goodness" of a function f is encoded in the Walsh transform of f. With the following lemma we can also directly reuse the threshold oracle (U_{ε} in fig. 5.2).

Lemma 5.4.3. *Given $f \in F(\mathbb{F}_2^n, \mathbb{F}_2)$ and c as used in eq. (5.4) being the amount of times the linear approximation η holds true, it holds that*

$$\frac{\hat{\chi}_f(\eta) + 2^n}{2} = (Corr(\bar{t}_{\eta,1}(f)) + 1) \cdot \frac{2^n}{2} = c.$$

That means we can either transform the output of our Walsh transform to the amount of good states or choose ε to be $\varepsilon_c \cdot \frac{2}{2^n} - 1$ with ε_c being the threshold used in section 5.4.1[15].

One might be tempted to think that Walsh transformation is what a quantum computer is especially good at and construct our oracle as done in fig. 5.12, but as the following analysis shows we cannot simply use the quantum Walsh transform as a standalone oracle.

Analysis

Before applying the $U_{\hat{\chi}_f}$-gate we have the following state:

$$\begin{aligned} |\Phi_0\rangle &= |\eta\rangle |+\rangle^{\otimes n} \\ &= \sum_{x \in \mathbb{F}_2^n} |\eta\rangle |x\rangle \end{aligned} \tag{5.9}$$

Which gets mapped by $U_{\hat{\chi}_f}$ to:

$$\begin{aligned} |\Phi_1\rangle &= \sum_{x \in \mathbb{F}_2^n} |\eta\rangle (-1)^{\langle \eta | x\rangle \oplus f(x)} |x\rangle \\ &= |\eta\rangle \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \eta | x\rangle \oplus f(x)} |x\rangle \end{aligned} \tag{5.10}$$

Now the "goodness" is encoded in the amplitude again (which does not help, since that cannot directly be used). Getting it out would again require quantum counting, so there is no benefit.

[14] I.e. still exponential runtime in regard to the function or counting register size in the A0 model.

[15] While the latter seams easier, we would have to modify the threshold oracle as its current implementation only works for integers. Likely its construction would then be similar to the first approach.

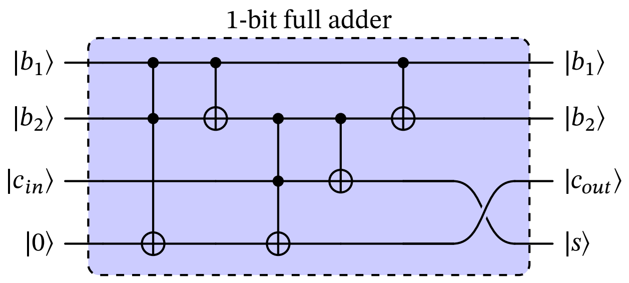


Figure 5.13: 1-bit full adder (1bfa) as a quantum circuit. b_1 and b_2 represent the input bits, while c_{in} is another input bit used as the carry over from the previous full adder (compare fig. 5.15), $s := b_1 \oplus b_2 \oplus c_{in}$ is their sum and $c_{out} := a + b + c_{in} \geq 2 \bmod 2$ is the output carry bit that is true exactly when more than one input is true. The correctness of this circuit can be simply evaluated by checking all possible inputs and comparing to the truth table of the classic full bit adder.

But as described in section 4.1 the Walsh transform can be computed classically. That also means its circuit can be mapped to a quantum circuit and executed on a quantum computer as descibed in appendix C and [38, chapter 8].

There are two majorly different approaches classically used, the naive approach and the *fast wlash-hadamard transform* (fwht). Both are discribed in the following sections.

5.4.3.2 Naive approach

The naive variant would work via a direct implementation of the definition of the walsh transform as described in definition 3.1.6. This would include a summation of 2^n terms, per possible input η. Calculating all possible $\hat{\chi}(\eta)$ would therefor take $\mathcal{O}\left(2^{2n}\right)$ additions.

As the direct convertion of that algorithm to quantum hardware would not need to calculate all possible 2^n values in sequence, but in superposition its complexity would only be in $\mathcal{O}\left(2^n\right)$[16].

A description of how to convert that calculation to quantum hardware will follow in the next parts.

Converting to Quantum Hardware

Arguably the main part of the naive approach is a summation of 2^n terms therefor we would need to implement a *quantum adder*.

Building this adder takes multiple steps. Starting with the quantum version of the 1-bit full adder, whose circuit is depicted in fig. 5.13. Notice how in contrast to the classical version which has three inputs and two outputs, the quantum version has four inputs and four outputs. That is because in quantum computing information can neither be created nor destroyed, that means the input has to be reconstructible from the output. To be precise, the transformation has to be unitary. But we can see that if we would e.g. override b with s we could not reconstruct the input if the output is $s = 0$ and $c_{out} = 1$, as we would not know whether b or c_{in} was true, analogously if we pass b and override a with s.

n of these 1-bit full adders can then be chained together to form a ripple carry binary adder as depicted in fig. 5.15 which can add two n-bit bit-strings representing two natural numbers[17]. As bit-strings can either be interpreted as having the most significant bit first or last (or in case of quantum circuits at the top or the bottom respectively) there are two variants of this addition circuit, the big-endian and the little-endian variant, depicted in fig. 5.15 and fig. 5.14 respectively. This thesis primarily uses the big-endian variant (compare e.g. algorithm 10), but the little-endian variant is also depicted for completeness. This circuit, when used as a building block, will simply be called *adder*.

[16]The output would as well be in a corresponding superposition and therefor not easily usable/ measurable, but that is fine for our use-case.

[17]Or integers, if represented using the 2's complement.

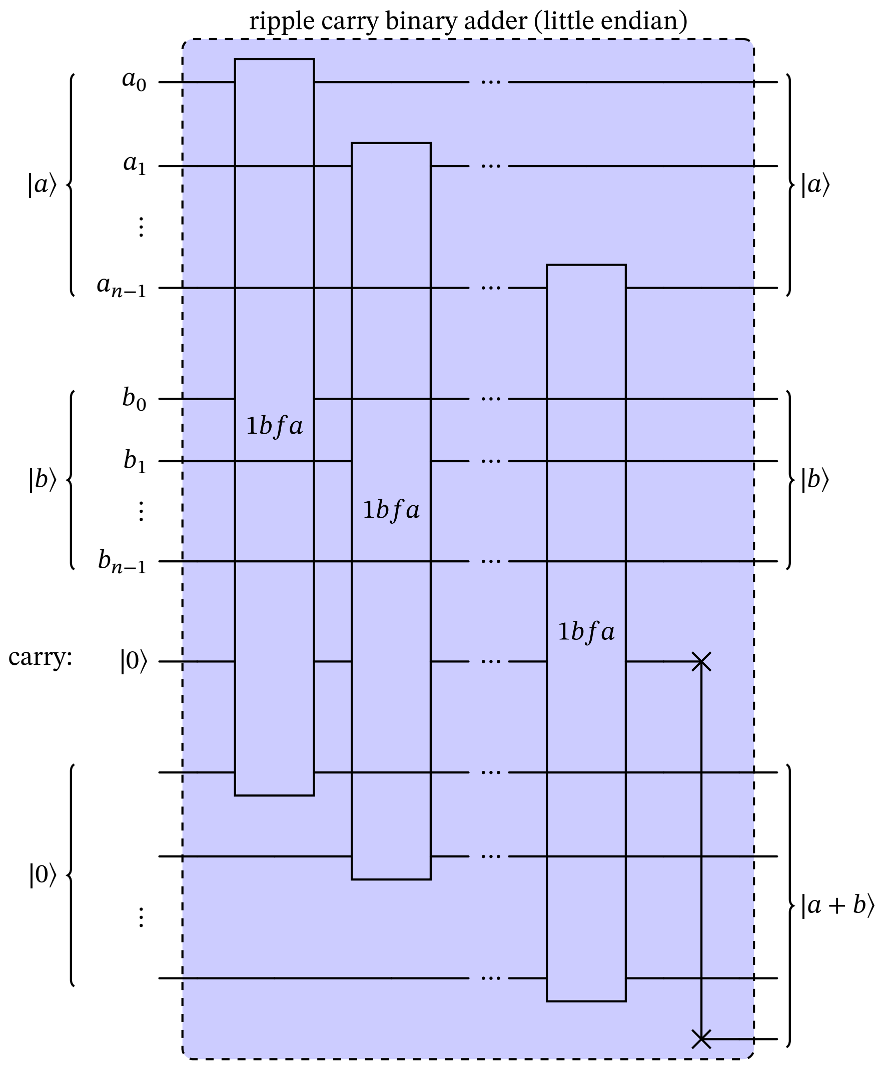


Figure 5.14: Binary addition circuit for two n-bit numbers, in this case using the little endian representation which e.g. means $a = \sum_{i=0}^{n-1} 2^i a_i$ or that the most significant bit is below the less significant ones.

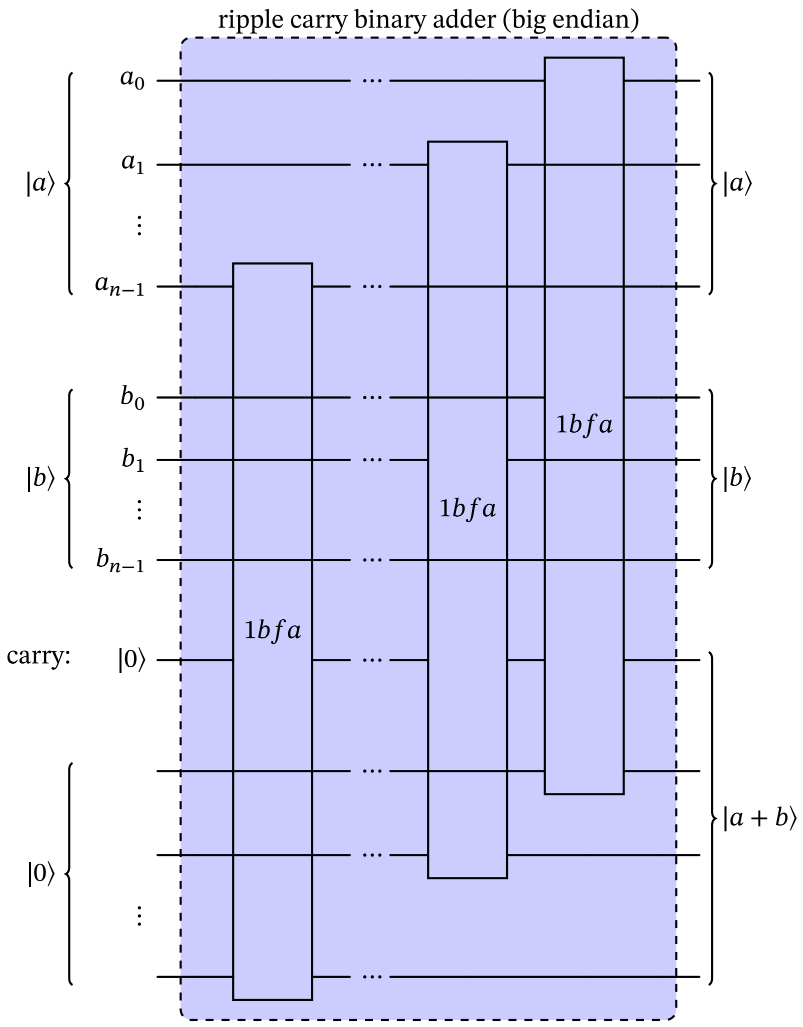


Figure 5.15: Binary addition circuit for two n-bit numbers, in this case using the big endian representation which e.g. means $a = \sum_{i=0}^{n-1} 2^i a_{n-i}$ or that the most significant bit is above the rest.

```
sum_loop:
    add al, [esi] ; Add the value in the array to the accumulator
    inc esi       ; Point to the next element in the array
    loop sum_loop ; Repeat the loop until ecx (array_size) becomes 0
```

Listing 5.1: Example assembly code for a loop that adds all elements of an array (pointed to by `esi`) in `al`. This would not run on its own, working code is appended in listing D.1.

In classical computing the adder can simply be reused by loading the next number and looping until all numbers are added like shown in listing 5.1.

This is not possible as the graph representing a quantum circuit has to be acyclic [42]. But instead of using the same logic gate recursively, the same quantum gate can be applied over and over again resulting in the circuit shown in fig. 5.16 that sums m n-bit bit-strings.

One thing that is not quite obvious from the circuit is that the ancilla register has to be reset to zero after every addition. This can be done by measuring the register and re-initializing it to $|0\rangle$ if the state is a product-state, this is not a quantum operation, as information is lost in the process, but it is a valid operation on a quantum computer as it is composed of measuring (ignoring the result) and initialization. If this would not be possible the circuit would have a register for every step of the summation, increasing the number of qubits needed and therefor the width of the circuit but neither the depth nor runtime of the circuit. Contrary to that, in its current form of the circuit, all inputs are initialized at the start, but they could be initialized one after another, reusing the same register, measuring it and initializing it to the next bitstring, decreasing the number of qubits needed but not the depth or runtime of the circuit. This could be achieved by using a non-deterministic oracle, with an internal counter, that returns the next bitstring on every call[18].

[18]This would actually be m different oracles, each supplying its own value, in our model.

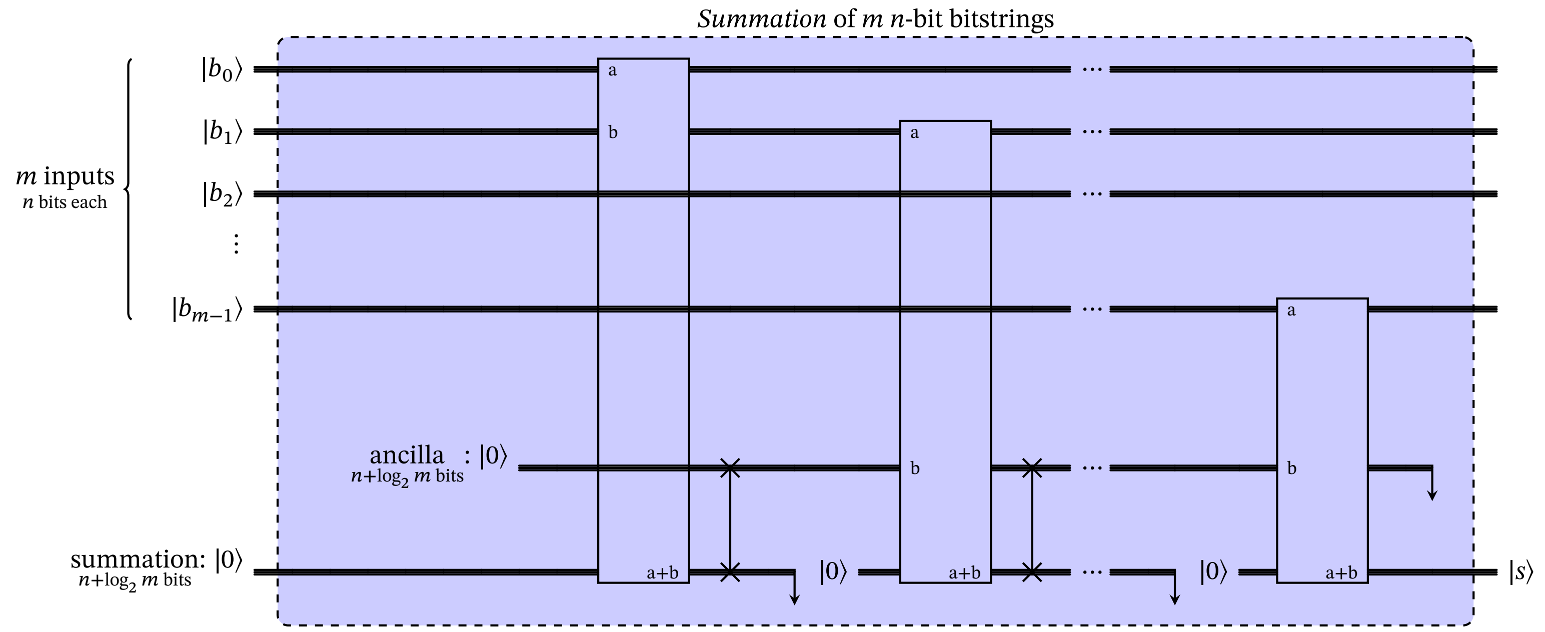


Figure 5.16: Summation of m bitstringths of length n each. I.e. $s := \sum_{i=0}^{m-1} b_i$. This takes m adders as defined in fig. 5.15.Additionally, we need an ancilla register, that gets reset to zero after every addition.

Recall, in accordance with lemma 5.4.3 the sum we are trying to compute is related to the correlation and therefor the Walsh transform of the function f for a specific η:

$$\eta \mapsto \sum_{x\in\mathbb{F}_2^n} (-1)^{\langle\eta|x\rangle\oplus f(x)}.$$

Therefor we can define the loading oracles as

$$'O^f_{\eta,x} : |0\rangle \mapsto |b_x\rangle := |(-1)^{\langle\eta|x\rangle\oplus f(x)}\rangle .$$

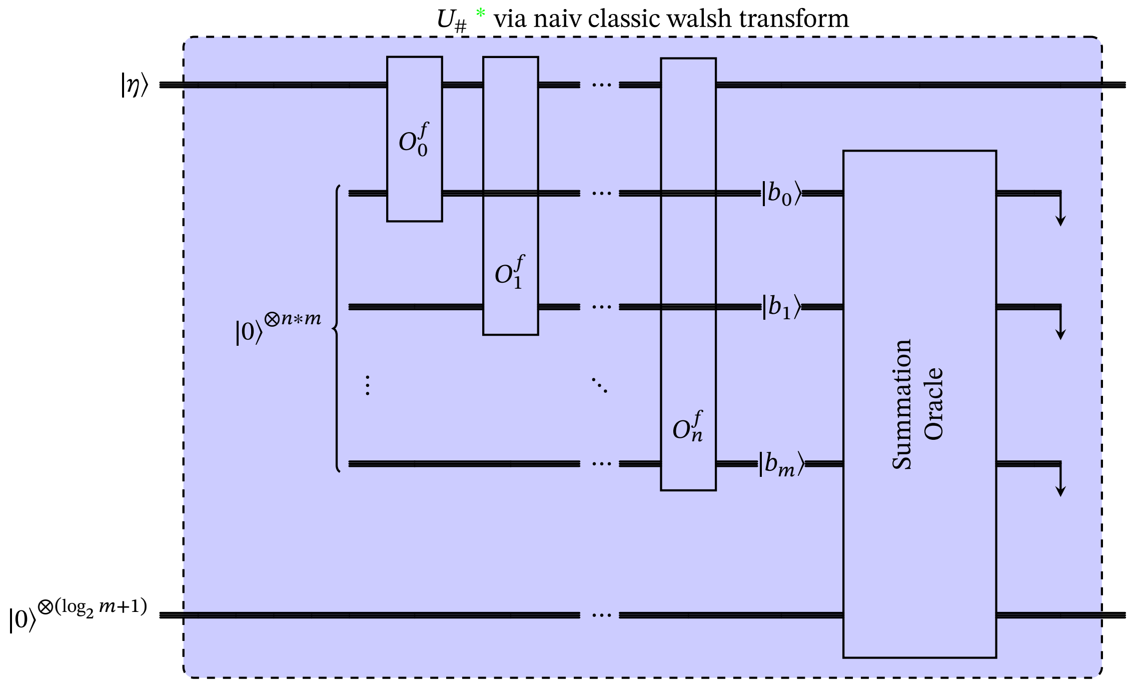
or as

$$O^f_x : |\eta\rangle|0\rangle \mapsto |\eta\rangle|b_x\rangle := |\eta\rangle|(-1)^{\langle\eta|x\rangle\oplus f(x)}\rangle , \tag{5.11}$$

since we later want to change which η we are using via a quantum register. The other values are not needed and can be chosen arbitrarily, as these oracles are only used for loading the values into the summation register, which is initialized to $|0\rangle$. Furthermore, the oracle can be implemented efficiently as long as f is efficiently computable.

One important thing to note is that the result of these oracles is always either $|1\rangle$ or $|-1\rangle$. This brings one advantage and one disadvantage. The advantage is that we can shrink the ancilla and summation register to $\log_2 m + 1$ qubits each. The disadvantage is that we need to represent negative numbers, s.t. the summation circuit still works. But (as mentioned in footnote 17) this can be done by using the 2's complement representation of negative numbers[27, p. 17]. We just have to keep that in mind for the rest of the circuits as the output obviously will be in 2's complement representation as well.

$U_\#$ * can therefor be implemented as shown in fig. 5.17.



$U_\#$ * via naiv classic walsh transform
$|\eta\rangle$
O^f_0
O^f_1
O^f_n
$|0\rangle^{\otimes n*m}$
$|b_0\rangle$
$|b_1\rangle$
$|b_m\rangle$
Summation Oracle
$|0\rangle^{\otimes(\log_2 m+1)}$

Figure 5.17: Implementation of $U_\#$ * using the naive classical Walsh transform. With $m = 2^n$ as we will sum over all possible $x \in \mathbb{F}_2^n$.

As clearly visible when computing the whole sum m has to be chosen to be $m = 2^n$ which results in $\mathcal{O}(2^n)$ oracle calls and gates.

5.4.3.3 Fast Walsh-Hadamard transform

One well-known fact from classical computing (very important in signal processing) is that, as is the case with the Fourier transform, there exists a naive and a fast version of the transform.

While the naive method works as described above and calculates every value for any η independent of each other, the fast version reuses many calculation results and computes the whole spectrum (for all 2^n possible η) at once using only $\mathcal{O}(n2^n)$ steps (instead of $2^n * 2^n$ for the naive spectrum). How that is achieved can be seen in listing 4.1.

Unfortunately this approach cannot be used in our construction as the input into the oracle is a single η (which might be in superposition) and it therefor still takes the whole 2^n steps to compute the result for a single η, or multiple in superposition. One might think of precomputing the whole Walsh spectrum into quantum RAM (qram) as described in [25].

This however has the major disadvantage, that filling the qram now takes $\mathcal{O}(n2^n)$ steps, caused by the fwht. $U_\#$ * then simply queries this qram at location η[19].

Unfortunately reading and using (measuring) any datapoint in the qram destroys it unrestorably as caused by theorem 3.4.3. Therefor the data in the qram cannot be reused and has to be reinitialized for every query to $U_\#$ *, effictivly making it part of its runtime. Although [43] proposed a countermeasure to some extend, called *quantum forking* (QF), it turns out this is merely an additional control qudit (of dimension d) deciding what unitary (of dimension n) to apply to the qram, resulting in the unitary

$$\begin{pmatrix} U_1 & 0 & \cdots & 0 \\ 0 & U_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & U_d \end{pmatrix},$$

which does not have an obvious advantage in our case.

5.4.3.4 Exploiting the Hoeffding bound to build a phase oracle

While the FWHT is not directly applicable, we can still use faster classical methods to compute a good approximation of the Walsh transform, proposed by Christoph Graebnitz. This method will exploit the following theorem by Hoeffding[30].

Theorem 5.4.4 (Hoeffding). *Let $X_1, \dots, X_s \in [0,1]$ iid.*[20] *random variables with average $\aleph := \frac{1}{s}\sum_{i=1}^{s} X_i$. It then holds for an error $\varepsilon > 0$ that*

$$\Pr[|\aleph - \mathrm{E}[\aleph]| < \varepsilon] > 1 - 2e^{-2\varepsilon^2 s}. \tag{5.12}$$

Proof. this follows directly by the counter-probability of theorem 1 in [30]. □

For further use we will need a slightly modified variant of this theorem that can be derived from a more general form of theorem 5.4.4, namely theorem 2 in [30]:

Lemma 5.4.5. *Let $X_1, \dots, X_s$ be independent random variables with $a_i \leq X_i \leq b_i$ and average $\aleph := \frac{1}{s}\sum_{i=1}^{s} X_i$. It then holds for an error $\varepsilon > 0$ that [30, Theorem 2]:*

$$\Pr[\aleph - \mathrm{E}[\aleph] < \varepsilon] > 1 - e^{-2\lambda^2 s^2 / \sum_{i=1}^{s}(b_i - a_i)^2}.$$

As we are only interested in the case where $a_i = -1$ and $b_i = 1$ for all i we can simplify this to

$$\Pr[\aleph - \mathrm{E}[\aleph] < \varepsilon] > 1 - e^{\frac{-\varepsilon^2 s}{2}}, \tag{5.13}$$

leading to the following theorem:

[19] η might be in superposition, as is a main purpose of using qram [43].

[20] Independent identically distributed

Theorem 5.4.6 (Hoeffding). *Let $X_1, \dots, X_s \in [-1, 1]$ iid.*[21] *random variables with average* $\aleph := \frac{1}{s}\sum_{i=1}^{s} X_i$. *It then holds for an error $\varepsilon > 0$ that*

$$\Pr\left[|\aleph - \mathrm{E}[\aleph]| < \varepsilon\right] > 1 - 2e^{\frac{-\varepsilon^2 s}{2}}. \tag{5.14}$$

Proof. Starting with the left-hand side of eq. (5.14) we can split it up into two cases:

$$\Pr\left[|\aleph - \mathrm{E}[\aleph]| \geq \varepsilon\right] = \Pr\left[\aleph - \mathrm{E}[\aleph] \geq \varepsilon\right] + \Pr\left[-\aleph - (-\mathrm{E}[\aleph]) \geq \varepsilon\right],$$

which, as all X_i and therefor $\mathrm{E}[\aleph]$ are finite, making it linear, is equivalent to

$$\Pr\left[\aleph - \mathrm{E}[\aleph] \geq \varepsilon\right] + \Pr\left[-\aleph - \mathrm{E}[-\aleph] \geq \varepsilon\right].$$

When we now create a new set of iid. random variables X_i^- with $X_i^- := -X_i$, resulting in $\aleph^- = -\aleph$, we can rewrite this as

$$\Pr\left[\aleph - \mathrm{E}[\aleph] \geq \varepsilon\right] + \Pr\left[\aleph^- - \mathrm{E}[\aleph^-] \geq \varepsilon\right].$$

As both summands now conform to the prerequisites of eq. (5.13) we can apply it to both of them, resulting in

$$\Pr\left[\aleph - \mathrm{E}[\aleph] \geq \varepsilon\right] + \Pr\left[\aleph^- - \mathrm{E}[\aleph^-] \geq \varepsilon\right] \leq 2e^{\frac{-\varepsilon^2 s}{2}},$$

which is equivalent to

$$\Pr\left[|\aleph - \mathrm{E}[\aleph]| \geq \varepsilon\right] \leq 2e^{\frac{-\varepsilon^2 s}{2}},$$

being the complement of eq. (5.14). □

As it is now possible to choose some iid. random variables ranging from −1 to 1 and get a good approximation of the average, we can use this to build a phase oracle for $U_{\#}$ *. As a recap, the oracle gets a $|\eta\rangle$ as input and outputs something related to the count as, in this case $|\hat{\chi}_f(\eta)\rangle$ in accordance with lemma 5.4.3, or in other words a measure of how "good" of a linear approximation η is for f.

Given a fixed η we can now use the Hoeffding bound to approximate $\hat{\chi}_f(\eta)$ by using the following algorithm:

Algorithm 5: Walsh transform approximation

```
input : function $f : \mathbb{F}_2^n \to \mathbb{F}_2, \eta \in \mathbb{F}_2^n, s \in \mathbb{N}$
sample[a] $x = (x_1, \dots, x_s) \sim (\mathbb{F}_2^n)^s$ uniformly[b];
for $i \in \{1, \dots, s\}$ do
    $X_i \leftarrow (-1)^{\langle \eta | x_i \rangle \oplus f(x_i)}$;
end
$\aleph \leftarrow \frac{1}{s}\sum_{i=1}^{s} X_i$;
output: $\aleph \cdot 2^n$ with $\aleph \approx \frac{\hat{\chi}_f(\eta)}{2^n}$
```

[a] This sampling can be done beforehand in a classical manner.

[b] x is an outcome of a random experiment, the sampling does not need to be uniform for theorem 5.4.6 to apply. But for lemma 5.4.7 to hold true all x_i must be iid. from each other.

There are however three major caveats we need to address when running algorithm 5.

Firstly, we have to prove that the X_i are indeed iid. random variables $\in [-1, 1]$, as this is a prerequisite for theorem 5.4.6.

Lemma 5.4.7. *For a fixed η and f the X_i in algorithm 5 are iid. random variables* $\in [-1, 1]$.

Proof. This proof consists of three parts:

[21] independent identically distributed

1. The values X_i can take are $\in [-1, 1]$.
   As f is a Boolean function, X_i can only take the values -1 and 1.

2. The X_i are independent random variables.
   Here some restrictions to the probability distribution of x have to be put in place. As every X_i only depends on x_i and not on any other x_j for $j \neq i$, the X_i could be independent random variables iff (and only if!) the x_i are independent. This is e.g. the case if x is chosen uniformly. More formally it has to hold $\forall i, j \in \{1, \dots, s\}$ with $i \neq j$ and $\forall b_i, b_j \in \{-1, 1\}$:
   $$\Pr\left[X_i = b_i \wedge X_j = b_j\right] = \Pr\left[X_i = b_i\right] \Pr\left[X_j = b_j\right].$$
   Which, assuming uniform distribution (*), is shown as follows:
   $$\begin{aligned}\Pr\left[X_i = b_i \wedge X_j = b_j\right] &= \sum_{\substack{y_i, y_j \in \mathbb{F}_2^n \\ X_i(y_i) = b_i \\ X_j(y_j) = b_j}} \Pr\left[x_i = y_i \wedge x_j = y_j\right] && (5.15)\\ &\overset{*}{=} \frac{2^{(s-2)n} \cdot \#X_i^{=b_i} \cdot \#X_j^{=b_j}}{2^{sn}} = \frac{\#X_i^{=b_i} \cdot \#X_j^{=b_j}}{2^{2n}} && (5.16)\\ &= \frac{\#X_i^{=b_i}}{2^n} \cdot \frac{\#X_j^{=b_j}}{2^n} && (5.17)\\ &= \Pr\left[X_i = b_i\right] \Pr\left[X_j = b_j\right], && (5.18)\end{aligned}$$
   where $\#X_i^{=b_i}$ denotes the number of y_i s.t. $X_i(y_i) = b_i$, similarly for $\#X_j^{=b_j}$.

3. The X_i are identically distributed.
   Here a similar argument as in the previous part can be made. While this is not necessarily the case, we just choose a distribution for x s.t. it is. E.g. the uniform distribution. Now, as all x_i are identically distributed, all X_i are identically distributed as well.

□

Secondly, we have to show that $\mathrm{E}[\aleph] = \frac{\hat{\chi}_f(\eta)}{2^n}$ holds. This is done by the following lemma:

Lemma 5.4.8. *For a fixed $\eta \in \mathbb{F}_2^n$ and $f \in F(\mathbb{F}_2^n, \mathbb{F}_2)$ it holds that*
$$\mathrm{E}[\aleph] = \frac{\hat{\chi}_f(\eta)}{2^n}$$
for $\aleph := \frac{1}{s} \sum_{i=1}^{s} (-1)^{\langle \eta | x_i \rangle \oplus f(x_i)}$ and $x = (x_1, \dots, x_s) \sim (\mathbb{F}_2^n)^s$ uniformly.

Proof. Let $X_i := (-1)^{\langle \eta | x_i \rangle \oplus f(x_i)}$. As the expected value is a linear operation we can split it up into the sum of the expected values of the X_i:
$$\begin{aligned}\mathrm{E}[\aleph] &= \mathrm{E}\left[\frac{1}{s} \sum_{i=1}^{s} X_i\right]\\ &= \frac{1}{s} \sum_{i=1}^{s} \mathrm{E}[X_i].\end{aligned}$$
As the X_i are iid. random variables, it holds that $\mathrm{E}[X_i] = \mathrm{E}[X_j]$ for all $i, j \in \{1, \dots, s\}$. Therefor we can

rewrite the above as

$$\begin{aligned} E[\aleph] &= \frac{1}{s}\sum_{i=1}^{s} E[X_i] \\ &= \frac{1}{s}\sum_{i=1}^{s} E[X_1] \\ &= E[X_1]. \end{aligned}$$

Resubsituting X_1 we get

$$E[\aleph] = E\left[(-1)^{\langle \eta | x_1 \rangle \oplus f(x_1)}\right]$$

with $x_1 \sim \mathbb{F}_2^n$ uniformly, which by definition of the expected value is equivalent to

$$E[\aleph] = \sum_{x_1 \in \mathbb{F}_2^n} \Pr[x_1] \cdot (-1)^{\langle \eta | x_1 \rangle \oplus f(x_1)}.$$

As x_1 is chosen uniformly, $\Pr[x_1] = \frac{1}{2^n}$ for all $x_1 \in \mathbb{F}_2^n$. □

And lastly, we have to choose a fitting sample size s to balance runtime against precision. Obviously the runtime is directly proportional to the sample size s, i.e. it is in $\mathcal{O}(s)$. The relationship between samples (i.e. runtime), precision and success-probability however is not as easy to quantify.

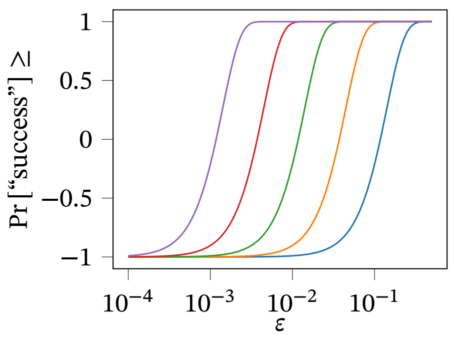


Figure 5.18: absolute error vs. minimal success probability with 100,1000,10000,100000,1000000 samples.

The most interesting takeaway here is that this does not at all depend on the size of the cipher/ input length n. This is because the Hoeffding bound is independent of n, which means that the runtime of the algorithm is independent of n as well, as long as n is not part of the tolerance ε[22]. This seems like a major advantage over the naive approach, but more on that in chapter 6.

A few possible sample sizes and the resulting success probabilities in dependence of the tolerance can be seen in fig. 5.18[23].

As also mentioned in footnote 22, and to allow a fair comparison with previous approaches for $U_\#$ *, we will use the following new definition of success probability

$$\Pr[\text{success}] := \Pr\left[|\aleph - E[\aleph]| < \varepsilon \cdot \frac{2}{2^n}\right] \tag{5.19}$$

as this is the scaling needed to convert from the ε used in algorithm 5 to the ε used in section 5.4.1, shown in lemma 5.4.3.

This results in the following runtime of this version of $U_\#$ * that uses the Hoeffding bound but otherwise the same parameters as the other approaches, particularly the same ε as in section 5.4.1:

[22]That might be needed though as the tolerance probably needs to go down with increasing input size

[23]It has to be noted that our hoeffding-bound is a lower bound (not tight) for the success probability, which results in negative values in the graph in fig. 5.18. This is not a problem, as the success probability is always $\in [0, 1]$ and above the lower bound, the graph is only meant to show the relationship between the three variables.

Theorem 5.4.9. *The runtime of $U_\#$ * using the Hoeffding bound is in*

$$\mathcal{O}\left(\frac{2^n \ln(\frac{1}{1-p})}{\varepsilon^2}\right),$$

where n is the size of the input, p the success probability and ε is the error tolerance.

Proof. Substitung our new error bound as defined in eq. (5.19) into eq. (5.14) we get

$$p = \Pr\left[|\aleph - \mathrm{E}[\aleph]| < \varepsilon \cdot \frac{2}{2^n}\right] > 1 - 2e^{\frac{-(\varepsilon \cdot \frac{2}{2^n})^2 s}{2}} \tag{5.20}$$

$$\Leftrightarrow s < \frac{2^{2n} \ln(\frac{2}{1-p})}{\varepsilon^2} \tag{5.21}$$

Which somewhat counterintuitivly[24] means in order to guarantee at least success probability p we need to choose s to be at least

$$\frac{2^{2n} \ln(\frac{2}{1-p})}{\varepsilon^2} \in \mathcal{O}\left(\frac{2^n \ln(\frac{1}{1-p})}{\varepsilon^2}\right).$$

As the runtime of algorithm 5, which is used as $U_\#$ *, is directly proportional to s, the runtime of $U_\#$ * using the Hoeffding bound is in

$$\mathcal{O}\left(\frac{2^{2n} \ln(\frac{1}{1-p})}{\varepsilon^2}\right)$$

as well. □

When algorithm 5 is used as $U_\#$ *, the inputs f as well as s become a fixed metaparameter while η is the only variable input given as a quantum state $|\eta\rangle$. One question that is not considered in this thesis but might be part of future work, is what exactly happens to the random sampling that is part of algorithm 5. It could either be done classically beforehand and baked into the circuit, or it could be done by something similar to entangling with an additional register that gets initialized to $|+\rangle$ and measured in standard-basis on demand[25].

5.5 Going from $\mathbb{F}_2^n \mapsto \mathbb{F}_2$ to approximating encryption functions

We have now created a few different approaches to create an oracle that marks states that represent a good enough linear approximation to a Boolean function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$. However, as we are interested in the security of cryptographic schemes against linear cryptanalysis, we need to be able to mark states that represent a good enough linear approximation to an encryption function as defined in definition 3.1.3.

Input splitting

The first part of that transformation is to simply redefine the encryption function in accordance to preliminiary 3.1.4 resulting in $f : \mathbb{F}_2^{n_m+n_k} \mapsto \mathbb{F}_2^{n_c}$. The n used in the previous section for the input length of $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is simply split up into n_k and n_m for the key and message respectively, i.e. $n = n_k + n_m$.

[24]It seems like we should have at most (not at least) this amount of samples when looking at eq. (5.21), but thats only to keep the right-hand side of eq. (5.20) small, increasing that would also increase the success probability.

[25]Now using this register as another input to the oracle would result in actual randomness, but the same samples for every approximation/η.

Vectorial Boolean functions

The second part is a bit more complicated. As we are now dealing with a function that has a multiple outputs, i.e. an output vector or bitstring, some calculations have to be generalized to multiple dimensions.

Quantum counting

Recall, that in quantum counting we get an estimate $\tilde{c}$ to $c = |f^{-1}(1)|$ for any $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ with $\varepsilon \leq |\tilde{c} - c|$. Therefor, to get our "goodness" $\mathbb{g} := |corr(\bar{t}_{\alpha \# \gamma, \beta})(g)|$ for a linear approximation charcterized by α, γ, β for any cryptographic function $g : \mathbb{F}_2^{n_m} \times \mathbb{F}_2^{n_k} \mapsto \mathbb{F}_2^{n_c}$ it suffices to set the f inducing $U_\#$ * in section 5.4.1.3 to $f_{\alpha,\gamma,\beta}(m,k) := \bar{t}_{\alpha \# \gamma, \beta}(g, m \# k)$. This therefor simply increases our input length of f, which is the only factor for the runtime, to $n = n_m + n_k$, resulting in a runtime of

$$\Theta\left(\frac{\sqrt{2^{n_m+n_k}}}{\varepsilon}\right).$$

This, again, however only works if the oracle shall not be used in a quantum circuit, as it is not unitary. Instead using basic quantum counting the runtime would be in

$$\mathcal{O}\left(\frac{2^{n_m+n_k}}{\varepsilon}\right).$$

Which does seem bad, but as we can increase ε quite high, this is not as bad as it seems. If, for example we only care if an approximation holds true in more than $\frac{3}{4}$ of cases, we can set $t = 2$ and therefor $\varepsilon = 2^{n_m+n_k-2}$ resulting[26] in a constant runtime.

Standalone oracle

A similar procedure can be applied to the standalone oracle, but as it does not rely on quantum counting we cannot simply increase the input length by using a function f as described above. Instead the standalone variant directly computes the goodness $\mathbb{g}$ by using the walsh transform of f

$$\eta \mapsto \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \eta | x \rangle \oplus f(x)}.$$

Which, if we simply replace f with g, would result in something like

$$\eta \mapsto \sum_{x \in \mathbb{F}_2^{n_m+n_k}} (-1)^{\langle \eta | x \rangle \oplus \bar{t}_{\eta,\beta}(g,x)},$$

which does not make much sense as neither β is part of the consideration nor do we need the additional $\langle \eta | x \rangle$. Instead, what we rather want to calculate is

$$\alpha, \gamma, \beta \mapsto \sum_{x \in \mathbb{F}_2^{n_m+n_k}} (-1)^{\langle \alpha | x_{n_m} \rangle \oplus \langle \gamma | x_{n_k} \rangle \oplus \langle \beta | g(x) \rangle}.$$

As this is not trivially possible we have to calculate every output mask seperatly as described in section 4.1 or more precisely algorithm 2. This however can be done by redifining the loading oracles in eq. (5.11) to

$$O_x^f : |\eta\rangle |0\rangle \mapsto |\eta\rangle |b_x\rangle := |\eta\rangle |(-1)^{\langle \eta | (x \# f(x)) \rangle}\rangle$$

where $\eta = \alpha \# \gamma \# \beta$ and $x \in \mathbb{F}_2^{n_m+n_k}$, or in other words

$$O_x^f : |\alpha\rangle |\gamma\rangle |\beta\rangle |0\rangle \mapsto |\alpha\rangle |\gamma\rangle |\beta\rangle |b_x\rangle := |\alpha\rangle |\gamma\rangle |\beta\rangle |(-1)^{\langle \alpha | x_{n_m} \rangle \oplus \langle \gamma | x_{n_k} \rangle \oplus \langle \beta | f(x) \rangle}\rangle.$$

[26]It is not quite as straight forward as this in practice, as the t first has to approximate an angle that then gets converted via arccos back to a count. The main idea holds though.

The naive variant oracle therefor has a runtime of $\mathcal{O}\left(2^{n_m+n_k+n_c}\right)$ oracle calls.

Similarly, the random variables in the Hoeffding variant, algorithm 5, have to be changed to

$$X_i \leftarrow (-1)^{\langle \eta | (x_i \# f(x_i)) \rangle}.$$

Where $x = (x_1, \dots, x_s) \sim (\mathbb{F}_2^{n_m+n_k})^s$ is sampled uniformly at random. The resulting algorithm, which would get converted to the oracle, would thereby look like algorithm 6.

Algorithm 6: Walsh transform approximation

```
input  : function $f : \mathbb{F}_2^{n_m} \times \mathbb{F}_2^{n_k} \mapsto \mathbb{F}_2^{n_c}, s \in \mathbb{N}, \alpha \in \mathbb{F}_2^{n_m}, \gamma \in \mathbb{F}_2^{n_k}, \beta \in \mathbb{F}_2^{n_c}$
sample $m = (m_1, \dots, m_s) \sim (\mathbb{F}_2^{n_m})^s$ and $k = (k_1, \dots, k_s) \sim (\mathbb{F}_2^{n_k})^s$ uniformly;
for $i \in \{1, \dots, s\}$ do
    $X_i \leftarrow (-1)^{\langle \alpha | m_i \rangle \oplus \langle \gamma | k_i \rangle \oplus \langle \beta | f(m_i, k_i) \rangle}$;
end
$\aleph \leftarrow \frac{1}{s} \sum_{i=1}^{s} X_i$;
output: $\aleph \cdot 2^n$ with $\aleph \approx \frac{corr(\bar{\iota}_{\alpha \# \gamma, \beta}(f))}{2^n}$
```

5.6 Resulting algorithm

In the following ρ denotes the function deciding weather an approximation is "good" while V_ρ is the corresponding phase labelling oracle depicted in fig. 5.2. Now that we have all the necessary tools, namely the amplitude amplification framework and the necessary oracle, we can combine them to create the final algorithm.
There are two possible ways to do so, discussed in the following two subsections.

5.6.1 Without Malviya

In many cases a special variant of the amplitude amplification algorithm is used, where no A is given. Only a partitioning function ρ and the induced oracle. The task is then simply to use amplitude amplification to get one of the good states (those with $\rho = 1$) from an equal superposition. It could be described as the Grover algorithm for multiple good states.

As the ρ is well-defined in our case, and we have multiple ways to create its oracle as done in section 5.4 this is a viable option, the resulting algorithm is shown in algorithm 7.

Algorithm 7: Finding Linear Relations on Quantum Computers without Malviya algorithm

```
input  : The induced oracle $V_\rho |\alpha\rangle |\gamma\rangle |\beta\rangle \mapsto (-1)^{\rho(\alpha,\beta,\gamma)} |\alpha\rangle |\gamma\rangle |\beta\rangle$ for a function $\rho$ that decides
         whether a state is good or bad.
$\ell \leftarrow 0, \alpha \leftarrow 0, \beta \leftarrow 0, \gamma \leftarrow 0$, and choose $c \in (1, 2)$;
while $\rho(\alpha, \beta, \gamma) \neq 1$[27] do
    $\ell \leftarrow \ell + 1; M \leftarrow \lceil c^\ell \rceil$;
    $|\alpha\gamma\beta\rangle \leftarrow |+\rangle$;
    choose integer $j \in [1, M]$ at random ;
    $|\alpha\gamma\beta\rangle \leftarrow Q^j |\alpha\gamma\beta\rangle$ with $Q := DV_\rho$[28];
    Measure $|\alpha\gamma\beta\rangle$ with respect to the standard basis to get $\alpha, \gamma, \beta$;
end
output: $\alpha, \gamma, \beta$
```

Unfortunately the function ρ is not trivially computable in polynomial time. Nonetheless, a possible scheme to evaluate it faster using quantum computing is shown in appendix A.1. As this uses the same steps (as V_ρ) it does not create much of an overhead in terms of asymptotic runtime.

Recall, the runtime of amplitude amplification is $\mathcal{O}\left(\sqrt{\frac{1}{a}}\right)$, where a is the success probability of A. As we use $A := H$ the success probability is the proportion of good states to overall states, i.e. $a = \frac{\mathcal{L}_{\delta_\tau}(f)}{2^{n_m+n_k+n_c}}$. This results in $\mathcal{O}\left(\sqrt{\frac{2^{n_m+n_k+n_c}}{\mathcal{L}_{\delta_\tau}(f)}}\right)$ calls to the oracle V_ρ. Unfortunately this is not the runtime of the algorithm, as the runtime of the oracle is not constant, but depends on the implementation of the oracle.

If this oracle can not be run beforehand to augment the states (compare section 6.1) the runtimes using this approach are shown in table 5.2.

Oracle	Algorithm runtime
Basic quantum counting - A0	$\mathcal{O}\left(\sqrt{\frac{2^{n_m+n_k+n_c+t^2}}{\mathcal{L}_{\delta_\tau}(f)}}\right)$
Approximate quantum counting	not feasible, $\mathcal{O}\left(\frac{2^{n_m+n_k+\frac{n_c}{2}}}{\varepsilon}\sqrt{\frac{c}{\mathcal{L}_{\delta_\tau}(f)}}\right)$
Naive classical Walsh transform	$\mathcal{O}\left((2^{n_m+n_k+n_c})^{3/2}\frac{1}{\sqrt{\mathcal{L}_{\delta_\tau}(f)}}\right)$
Hoeffding exploit	$\mathcal{O}\left(\frac{\ln(\frac{1}{1-p})}{\varepsilon^2}\cdot\sqrt{\frac{2^{5n_m+5n_k+n_c}}{\mathcal{L}_{\delta_\tau}(f)}}\right)$

Table 5.2: runtime of algorithm 7 with different runtimes of the oracle

Another variant would be to use the amplitude amplification algorithm provided by Brassard et al., that does not require ρ, but knowing the success-probability of A in advance. Again, as $A := H$, the success probability is the proportion of good states to overall states. These can be counted using the algorithm described in appendix A.2 beforehand. This would not result in different runtime but only a slightly different algorithm, we will therefor not go into detail here.

5.6.2 With Malviya

The previous section discussed a variant that is quite similar to normal Grover search and does not use anything specific to the problem at hand. But as we can already relate the amplitude of a state to the goodness of the approximation it represents using the Malviya algorithm, we can use this to our advantage.

This results in amplitude amplification still using the same oracle V_ρ as before, but now A is the Malviya algorithm instead of the Hadamard transform. The resulting algorithm is shown in algorithm 8.

Algorithm 8: Finding Linear Relations on Quantum Computers using Malviya algorithm and Amplitude Amplification

```
input :
    • The induced oracle $V_\rho |\alpha\rangle|\gamma\rangle|\beta\rangle \mapsto (-1)^{\rho(\alpha,\beta,\gamma)}|\alpha\rangle|\gamma\rangle|\beta\rangle$ for a function $\rho$ that decides whether a
      state is good or bad
    • The induced oracle $U_f|m\rangle|k\rangle|o\rangle \mapsto |m\rangle|k\rangle|o \oplus f(m,k)\rangle$ for any $f : \mathbb{F}_2^{n_m} \times \mathbb{F}_2^{n_k} \mapsto \mathbb{F}_2^{n_c}$
$\ell \leftarrow 0, \alpha \leftarrow 0, \beta \leftarrow 0, \gamma \leftarrow 0$, and choose $c \in (1,2)$;
while $\rho(\alpha,\beta,\gamma) \neq 1$ do
    $\ell \leftarrow \ell + 1; M \leftarrow \lceil c^\ell \rceil$;
    $|\alpha\gamma\beta\rangle \leftarrow |+\rangle$;
    choose integer $j \in [1, M]$ at random ;
    $|\alpha\gamma\beta\rangle \leftarrow Q^j|\alpha\gamma\beta\rangle$ with $Q := A^{-1}RAV_\rho$ and $A := H_{2^{n_m+n_k+n_c}}U_f(H_{2^{n_m+n_k}} \otimes I)$;
    Measure $|\alpha\gamma\beta\rangle$ with respect to the standard basis to get $\alpha, \gamma, \beta$;
end
output: $\alpha, \gamma, \beta$
```

Again, the runtime of algorithm 8 is in $\mathcal{O}\left(\sqrt{\frac{1}{a}}\right)$, where a is the success probability of A. This time, with $A := H_{2^{n_m+n_k+n_c}} U_f (H_{2^{n_m+n_k}} \otimes I)$ the success probability is harder to estimate.

Theorem 5.6.1. *Algorithm 8 needs $\mathcal{O}\left(\sqrt{\frac{1}{\mathcal{L}_{\mathbb{P}_\tau}(f)}}\right)$ calls to the oracle V_ρ as well as U_f.*

Proof. The runtime of amplitude amplification is $\mathcal{O}\left(\sqrt{\frac{1}{a}}\right)$ [14]. The success probability a of Malviya algorithm is $\mathcal{L}_{\mathbb{P}_\tau}(f)$ as stated in corollary 4.3.15.1. □

5.6.3 Exponential search

As the runtime of the algorithm is dependent on the success probability of A, which in turn is dependent on the linear approximatibility of f and the chosen threshold τ, we can use this to our advantage. The approximatibility can not be changed, but the threshold can be chosen freely. This can be exploited using an exponential search, where the threshold is increased exponentially until a good approximation is found. Shown in algorithm 9.

Algorithm 9: Exponential search to find Linear Relations using Malviya algorithmand Amplitude Amplification

```
input : Any $f : \mathbb{F}_2^{n_m+n_k} \mapsto \mathbb{F}_2^{n_c}$ and the corresponding induced oracle
        $U_f |(m + k)\rangle |o\rangle \mapsto |(m + k)\rangle |o \oplus f(m + k)\rangle$
$\tau \leftarrow 1$;
choose $c_\tau, c_\rho \in (1, 2)$;
while $\tau < 2^{n_c}$ do
    define $\rho_\tau(\alpha, \beta, \gamma) := \delta_\tau(|c(\bar{t}_{\alpha+\gamma,\beta}(f))|)$;
    create the induced oracle $V_{\rho_\tau}$ for $\rho_\tau$ using an approaches in section 5.4 with $\varepsilon \sim \frac{1}{\tau}$;
    $\ell \leftarrow 0, \alpha \leftarrow 0, \beta \leftarrow 0, \gamma \leftarrow 0$;
    while $\rho_\tau(\alpha, \beta, \gamma) \neq 1$ do
        $\ell \leftarrow \ell + 1; M \leftarrow \lceil c_\rho^\ell \rceil$;
        $|\alpha\gamma\beta\rangle \leftarrow |+\rangle$;
        choose integer $j \in [1, M]$ at random ;
        $|\alpha\gamma\beta\rangle \leftarrow Q^j |\alpha\gamma\beta\rangle$ with $Q := A^{-1}RAV_\rho$ and $A := H_{2^{n_m+n_k+n_c}} U_f (H_{2^{n_m+n_k}} \otimes I)$;
        Measure $|\alpha\gamma\beta\rangle$ with respect to the standard basis to get $\alpha, \gamma, \beta$;
    end
    yield : $\tau$: $\alpha, \gamma, \beta$
    $\tau \leftarrow \tau \cdot c_\tau$;
end
```

Chapter 6

Comparison

As we now have multiple algorithms that can be used to find linear approximations of cryptographic Boolean functions, we want to compare them. The different approaches covered in this thesis are:

0. naive classic brute force (trying every possible linear approximation and calculating its goodness until one sufficiently good is found)
1. the classic approach, calculating the Walsh spectrum using the fwht as described in section 4.1
2. quantum search as described in section 5.6.1
3. the barely modified Malviya algorithm as described in section 4.2
4. the Malviya algorithm with the trivial approximation removed using amplitude reduction as described in section 5.2.1
5. a novel amplitude amplified version of Malviya algorithm as descried in section 5.6.2

Approach	Runtime	p	Remark
0	$\mathcal{O}\left(\frac{2^{n_k+n_m+n_c}}{\mathcal{L}_{\delta_\tau}(f)} \cdot \mathcal{O}_\#\right)$	1	expected runtime
1	$\mathcal{O}\left((n_k+n_m)2^{n_k+n_m+n_c}\right)$	1	guaranteed runtime, only classic resources needed, space $\in \mathcal{O}\left(2^{n_k+n_m+n_c}\right)$, finds any number of the *best* approximations
2	$\mathcal{O}\left(\sqrt{\frac{2^{n_k+n_m+n_c}}{\mathcal{L}_{\delta_\tau}(f)}} \cdot \mathcal{O}_\#\right)$	1	expected runtime
3	$\mathcal{O}(1)$ or $\mathcal{O}(n_k+n_m+n_c)$	$\mathcal{L}_{\mathbb{P}_\tau}$	runtime given as oracle or gate model; better approximations are more likely
4	$\mathcal{O}(1)$ or $\mathcal{O}(n_k+n_m+n_c)$	$\mathcal{L}_{\mathbb{P}_\tau} \cdot \frac{2^{n_c}}{2^{n_c}-1}$	runtime given as oracle or gate model; better approximations are more likely
5	$\mathcal{O}\left(\sqrt{\frac{1}{\mathcal{L}_{\mathbb{P}_\tau}}} \cdot \mathcal{O}_\#\right)$	1	expected runtime, better approximations are more likely

Table 6.1: Comparison of the runtime and success probabilities (p) of the different approaches for finding linear approximations $g_{\alpha,\gamma,\beta}(x) := \langle(\alpha||\gamma)|x\rangle \oplus \langle\beta|f(x)\rangle$ to a function $f : \mathbb{F}_2^{n_m+n_k} \mapsto \mathbb{F}_2^{n_c}$ with $\alpha \in \mathbb{F}_2^{n_m}, \gamma \in \mathbb{F}_2^{n_k}, \beta \in \mathbb{F}_2^{n_c}$ as α, γ, β that holds true in more than τ cases. $\mathcal{O}_\#$ denotes the runtime of an algorithm (previously often denoted $\rho_{\alpha,\gamma,\beta}$) that decides whether an approximation is good or not.

Arguably, 5 is the most promising variant, as it guarantees a good measurement while still having a decent runtime. It also shares the additional benefit of the higher likelyhood of finding better approximations, i.e. the better an approximation is, the quadratically more likely it is to be found by this algorithm. The only downside is (besides the need for a sufficiently large, noise-free quantum computer) the hard to quantify runtime as it directly depends on the inner structure of the investigated function f and the chosen τ. One way to mitigate that and not have the algorithm run for a very long time is to steadily increase τ to get a list of better and better approximations, as described in section 5.6.3. This is probably the best approach in this thesis, combining the advantages of the different approaches.

6.1 Oracle runtime

Item 2 and item 5 both rely on an oracle that marks the good states. While this oracle could be declared to be given as a constant runtime black-box, that would be an unrealistic assumption. Instead we will have a look at the different approaches to creating such an oracle and compare their runtimes, namely:

1. the oracle using the classical walsh calculation as described in section 5.4.3.2
2. the oracle using the Hoeffding bound as described in section 5.4.3.4
3. the oracle using the quantum counting algorithm as described in section 5.4.1

These approaches primarily differ in the way they define an oracle $U_\#$ * that marks the good states. Unfortunately they seem quite large as shown in table 6.2.

Approach	Runtime	Remark
1	$\mathcal{O}\left(2^{n_k+n_m}\right)$	
2	$\mathcal{O}\left(\frac{2^{n_k+n_m}\ln(\frac{1}{1-p})}{\varepsilon^2}\right)$	This oracle does not calculate the actual count, but gives an estimate up to a certain error ε. With a probability of p its estimate is within the error bounds. How this error propagates through the amplitude amplification process is discussed in section 6.2.1.
3	$\mathcal{O}\left(\frac{\sqrt{2^{n_k+n_m}}}{\varepsilon}\right)$	Might not be feasible as the oracle uses measurements and is therefor not unitary.
3	$\mathcal{O}\left(\frac{2^{n_k+n_m}}{\varepsilon}\right)$	Uses basic quantum counting as this is feasible.

Table 6.2: Comparison of the runtime ($\mathcal{O}_\#$) of the different approaches for creating oracles ($U_\#$ *) to decide whether a linear approximation $g_{\alpha,\gamma,\beta}(x) := \langle(\alpha||\gamma)|x\rangle \oplus \langle\beta|f(x)\rangle$ to a function $f : \mathbb{F}_2^{n_m+n_k} \mapsto \mathbb{F}_2^{n_c}$ with $\alpha \in \mathbb{F}_2^{n_m}, \gamma \in \mathbb{F}_2^{n_k}, \beta \in \mathbb{F}_2^{n_c}$ as α, γ, β is good or not.

With an exception to variant 1, we have a parameter ε that defines the error bounds of the oracle. The smaller ε the more accurate the oracle is but the longer it takes to run. More interestingly ε can be chosen very high as we are mostly not interested in the exact count. We only need to know whether the approximation is good or not. This is a binary decision and therefor the oracle can be tuned to be very fast. If, for example we only want to know whether the approximation holds true in $\frac{7}{8}$ of cases, ε can be chosen to be $2^{n_k+n_m-3}$ resulting in a constant runtime. Generally speaking, to check whether an approximation holds true in $\frac{\alpha-1}{\alpha}$ of cases, ε can be chosen to be $2^{n_k+n_m-\log_2(\alpha)}$. resulting in a runtime of $\mathcal{O}(\alpha)$.

6.2 Combined runtime

As mentioned at the top of chapter 6 in order to get the total runtime of the algorithm we have to combine the runtime of the oracle and how often it gets called via the amplitude amplification algorithm. This is simply done by multiplying the two runtimes.

This is unfortunate as the oracle runtime and therefor the resulting combined runtimes are rather large. Combining the fastest procedures (item 5 with oracle 3) for example results in a runtime of

$$\mathcal{O}\left(\frac{2^{n_m+n_k}}{\sqrt{\mathcal{L}_{\mathbb{P}_\tau}\cdot\varepsilon}}\right).$$

Remember that $\mathcal{L}_{\mathbb{P}_\tau} \leq 1$ and therefor lengthens the runtime. It is possible to tune τ to balance better approximations against a faster runtime but how exactly they relate is very hard to quantify[1]. An easier parameter to tune is ε as it directly influences the runtime. The smaller ε the more accurate the algorithm is but the longer it takes to run. This is a trade-off that can be made by the user of the algorithm. Especially it has to be chosen in accordance with the chosen τ which then results in a feasible runtime of

$$\mathcal{O}\left(\frac{\tau}{\sqrt{\mathcal{L}_{\mathbb{P}_\tau}}}\right).$$

A completely different approach could be to try to extract the oracle into a preparation phase that augments the initial state such that the amplitude amplification algorithm only needs a single z gate on an already marked ancilla qubit to flip the marked states. This approach is depicted in fig. 6.1.

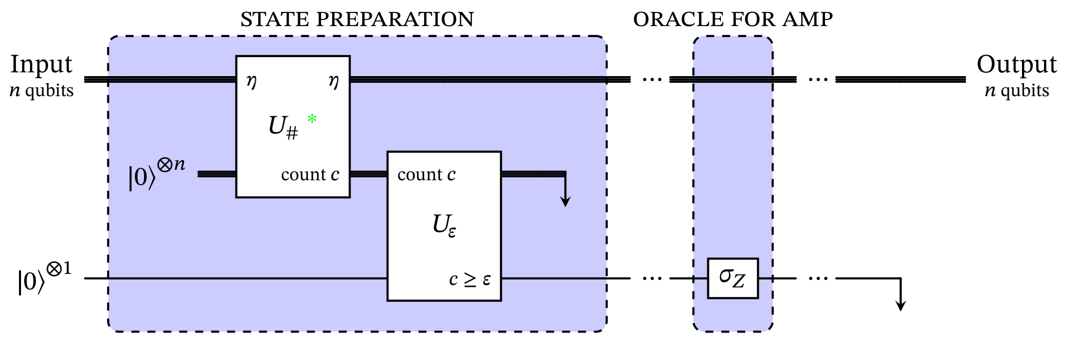


Figure 6.1: Splitting of fig. 5.2 into a preparation phase and the amplitude amplification phase. The first one is only called once to augment the initial states with an additional qubit that flags whether this state is good or not, s.t. in the amplitude amplification phase only a single z gate suffices as the oracle for the Grover operator.

Unfortunatly this approach is not compatible with the amplitude amplification algorithm defined by Brassard et al. Furthermore it is probably impossible to decouple the oracle marking the good states from the actual phase flip used in the amplitude amplification procedure as this has a rather strict uncomputation necessary to not decouple the state from the augmented ancilla qubit when applying the Grover diffusion operator only to the former part.

6.2.1 Additional remarks: probabilistic oracle

As some oracles are probabilistic, i.e. they have a specific error probability of being outside the error bounds, we need to examine how this error propagates through the amplitude amplification process. If we always use the same oracle, i.e. the randomness is "baked-in", my hypothesis is, that some good states are not marked while some bad states are marked. This might influence the runtime as well as the quality of the approximation found, as a slightly wrong set of approximations is amplified. If every run of the oracle uses a new random seed, the error might even out over multiple runs. This is a very interesting question for future research.

[1] As I can only calculate all correlations and therefor this measure for very small toy functions. With more resources a more complex real-world encryption function could be examined.

Chapter 7

Conclusion and Future Work

Conclusion

We have looked at a few different approaches to find linear relations that approximate cryptographic functions. Each such approximation, if sufficiently good, gives about a bit of information about the whole cipher. With enough such approximations the whole cipher is broken and can never be used again.
Finding these approximations is very hard without a quantum computer, but using one the search can be sped up a lot if the approximation is allowed to be somewhat erroneous. This is done by relying on the Malviya algorithm, which is a generalization of the Bernstein-Vazirani algorithm. As this is not very reliable, we used amplitude amplification to boost the success probability. This was not an easy task, as the construction of the oracle is not very straight forward and requires exponential complexity if a certain precision is required. If only a fixed approximation is needed however, which would make sense, the amplitude amplified algorithm might be promising. Although some empirical data would be required to have a good estimation of what values $\mathcal{L}_{\mathbb{P}_\tau}$ would take.

On the other hand we feared that the trivial approximation would become a large problem as it was already measured in 6.25% of cases in the empirical assessment by Malviya et Tiwari, although they only looked at a very small toy-function. However, by modeling how the measurement probabilities relate to the goodness of their corresponding approximations and the linearity of the whole function, we ascertained that the trivial approximation was a non-issue which can be eliminated with a constant computational overhead.

Future Work

There are still some open questions that could be answered in future work. The smallest unanswered question of this thesis is how exactly the amplitude amplified algorithm behaves in the case of a probabilistic oracle as mentioned in section 6.2.1. How would that even be implemented and what effect would it have on the algorithm? Would the error even out over multiple iterations? This would be an interesting question to answer in a future paper.

Another interesting question would be how the algorithm would behave if the oracle would not treat all approximations equally. As mentioned in footnote 6 the oracle could be extended to a real-valued oracle that amplifies better approximations more strongly. This would be a very interesting question to answer as it would be a very practical improvement to the algorithm and a scheme of how that could be done is already presented in [48].

Larger Topics

A few more interesting approaches came to my mind during work on this thesis that are way out of scope for this thesis but might be interesting to look at in future work, so I will briefly mention my ideas here.

Estimating $\mathcal{L}$

This thesis often use some forms of $\mathcal{L}$ as defined in section 4.3.2 to compare and evaluate the success rate of different approach in relation to this property of the examined function. However this is not very practical as it is very hard to estimate the value of $\mathcal{L}$ for a given function. It is improbable that calculating $\mathcal{L}$ for a given function is possible in polynomial time. If possible it would be nice to calculate it for an actually used encryption scheme like DES. This would in its smallest form already require 2^{64+48} possible inputs and is therefor improbable to be feasible. It might be possible though to use a quantum algorithm to calculate $\mathcal{L}$ for a given function. This would be a very interesting question to answer.
If that is not possible it might be possible to derive a proof that shows that a high $\mathcal{L}$ is not possible for a given function if it fulfills certain cryptographic criteria[1]. If this could be proven that would be very interesting and useful as it would allow to evaluate those criteria instead and proof that the algorithm of this thesis is not usable on a given function if it fulfills those criteria. In case that this can not be proven, the previous idea of calculating $\mathcal{L}$ using a quantum algorithm could be used to empirically correlate whether this measure is low for functions that fulfill these criteria.

Examine more complex approximations

Another idea would be to examine more complex approximations than just linear ones. This would require a more complex oracle. One approach could be to use multiple registers and applying U^i to the i-th register each, s.t. they combine to be a somewhat polynomial approximation up to a specific degree. This would allow finding more complex approximations than just linear ones, but I think this might be either useless or not work at all, some research might be able to deny or confirm that. E.g. a procedure similar to the one used in chapter 5 of [29] could be used.
Another approach I have even less of a concrete plan for, could be to extend this algorithm into differential cryptanalysis using an approach that is more closely related to the one in [34].

Finding a new kind of S-Box

As already discussed in section 1.1 the main reason why only a quantum algorithm is feasible for the problem of finding a linear approximation or evaluating a function on its linear approximatibility is the exponential growth of the search space with the size of the function i.e. the length of the input/output bitstring. Conventional methods therefor examine the inner structures of a specific function/cipher. One building block might be the S-Box in a substitution permutation network (SPN). An S-Box can be classically examined as the truth table of the S-Box has to be rather small to still be feasible[2]. The reason for its restricted is its representation as an exponentially (to the S-Box's input bitstring length) growing lookup table. It would not need to be a lookup table though, but it has to be a very non-linear function that is easy to calculate and invertible. It might be possible to use a quantum algorithm to initially find a different S-Box to be than used in SPNs with very large S-Boxes. This new kind of S-Box would not be represented as an exponentially sized truth table, but as a tuple of two efficiently computable functions[3]. One function would be the inversion of the other. This would be further down the horizon though as the quantum algorithm finding such a tuple with sufficiently sized functions would be rather complicated. It must eliminate all functions that have high linear bias and then find a tuple whose functions are each other's inversion using a complicated progress of destructive interference. If that is even possible, it would be a very interesting question to answer. The first part of eliminating linear approximatible functions might be possible using research presented in this thesis.

[1] One such example, solvable in the near future, would be to find the correct fit to replace eq. (4.24) with, as this would allow to directly estimate $\mathcal{L}_{\mathbb{p}_\tau}$ and therefor the runtime of our proposed algorithm for schemes that fulfil plausible deniability. This would directly show the usefulness of the algorithm presented in this thesis for linear cryptanalysis.

[2] Fast evaluation on classical hardware.

[3] E.g. given as classic circuits whose evaluation is in P.

Appendix A

More linear cryptanalysis

In the main thesis we mainly considered the objective of extracting one “good” linear approximation of a function. But there are some more objectives we could use linear cryptanalysis on quantum hardware for. These will be briefly discussed in the following sections.

A.1 Evaluating Goodness of a Linear Approximation

One rather simple generalization would be, instead of searching for a good linear approximation, we could evaluate the goodness of a given linear approximation, schematically defined as follows:

1. Given a linear approximation $\acute{f}_{\acute{k},\acute{m},\acute{c}}(x) := \langle(\acute{k}||\acute{m})|x\rangle \oplus \langle\acute{c}|f(x)\rangle$ to a function $f : \mathbb{F}_2^{n_k+n_m} \mapsto \mathbb{F}_2^{n_c}$ with $\acute{k} \in \mathbb{F}_2^{n_k}, \acute{m} \in \mathbb{F}_2^{n_m}, \acute{c} \in \mathbb{F}_2^{n_c}$ as $\acute{k}, \acute{m}, \acute{c}$ ...
2. ...run any $U_{\#}$ * from section 5.4 with $\eta = \acute{m} \mathbin{+\!\!+} \acute{k} \mathbin{+\!\!+} \acute{c}$ being the linear approximation, not in superposition.
3. Measure the resulting state to get z.
4. $\frac{z}{2^{n_c}}$ is the correlation, i.e. the goodness of the linear approximation.

As this evaluation is the same task $U_{\#}$ * from section 5.4 tries to solve, a faster solution to this problem would also directly reduce the runtime of the main algorithm in this thesis (algorithm 8).

A.2 Approximatibility of a function

In section 4.3.2 we had a look at how linear approximatible a function is and defined some measures for that. While it might be possible[1] to evaluate all of these measures using quantum computing and QPE there is one measure whose evaluation is straight forward with well-known quantum algorithms: $\mathcal{L}_{\delta_\tau}(f)$, defined in definition 4.3.6.

This measure has the advantage that its activation function only has two values, and therefor $c = \#\left\{m_x \in \mathbb{F}_2^{n_x}, m_y \in \mathbb{F}_2^{n_y} \mid \delta_\tau\left(\left|c(\bar{t}_{m_x,m_y}(f))\right|\right) = 1\right\}$ can be counted using quantum counting (explained in section 5.4.1.1) with a query complexity of $\mathcal{O}\left(\sqrt{2^{n_x+n_y}}\right)$[2].

[1]This could be part of future research as well.

[2]$n_x = n_m + n_k$ if considering cryptographic functions.

Appendix B

Basic quantum computing building blocks

B.1 Standard-gates

There is a variety of different gates that are already defined and might be used in this thesis. Some of them are listed below.

Pauli-Gates

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \sigma_Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \sigma_X = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Hadamard-Gate

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Phase-Gate

$$R_\phi = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix}$$

B.2 Standard Quantum Phase Estimation

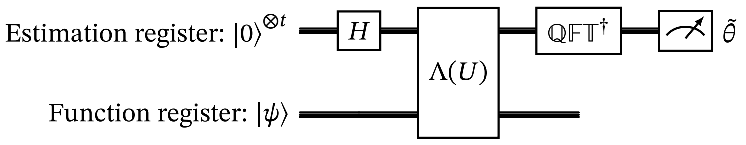
This section gives a short introduction to the quantum phase estimation algorithm (QPE) which estimates the phase of an eigenvector of a unitary operator. It mainly makes use of the following subroutine:

Definition B.2.1. *Given a unitary U, let*

$$\Lambda(U) : |j\rangle |y\rangle \mapsto |j\rangle (U^j |y\rangle)$$

be the unitary that performs controlled exponentiation of U, i.e. applies U to the second register as often as the first register says[1].

[1]For a visualization compare with fig. B.2.



Estimation register: $|0\rangle^{\otimes t}$
Function register: $|\psi\rangle$
H
$\Lambda(U)$
$\mathbb{QFT}^\dagger$
$\tilde{\theta}$

Figure B.1: the QPE algorithm as a circuit (with compact $\Lambda(U)$).

Problem B.2.1 (Quantum Phase Estimation). *Given a unitary operator U and an eigenvector $|u\rangle$ of U with eigenvalue $e^{2\pi i\theta}$, estimate θ (with τ bits of accuracy).*

The QPE-algorithm solves problem B.2.1 using the following procedure:

```
Algorithm 10: Quantum Phase Estimation (QPE)
  input : a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ with associated unitary operator $U_f \in \mathbb{C}^{N\times N}$, $N := 2^n$ as well
          as an eigenstate of $U_f$: $|\psi\rangle$[2]
  $|\Phi\rangle \leftarrow H\,|0\rangle^{\otimes t}$;
  $|\Phi\rangle\,|\psi\rangle \leftarrow \Lambda(U)\,|\Phi\rangle\,|\psi\rangle$;
  $|\Phi\rangle \leftarrow \mathbb{QFT}^\dagger\,|\Phi\rangle$;
  $k \leftarrow$ measure $|\Phi\rangle$ using the standard-basis;
  output: $\tilde{\theta} := \frac{k}{t}$
```

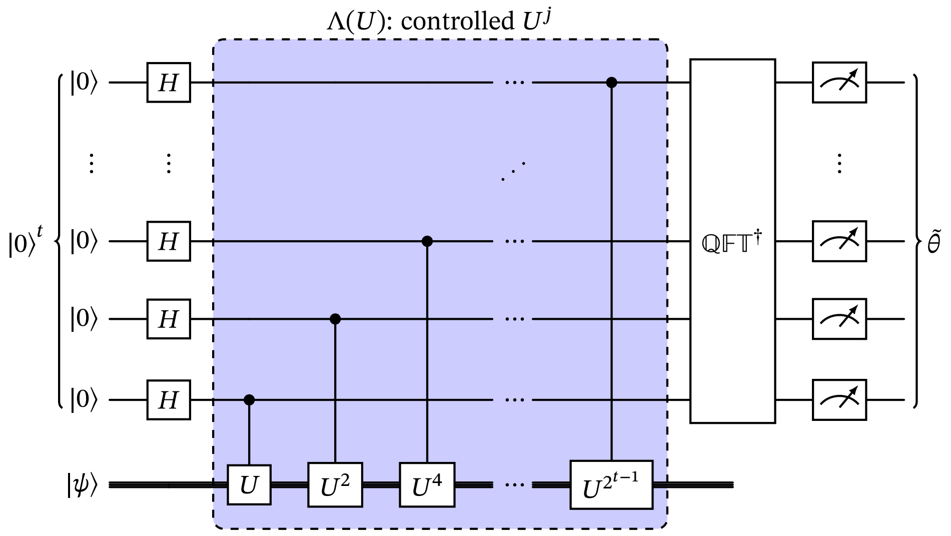


Figure B.2: The QPE algorithm as a circuit (with $\Lambda(U)$ from fig. B.1 being expanded into its components using the gate-model).

Lemma B.2.2. *As shown in fig. B.2, the $\Lambda(U)$-gate can be efficiently implemented in $\mathcal{O}(t)$ iff given oracle-access to all t oracles $U^{2^i}, 0 \leq i < t$. If this is not the case, the $\Lambda(U)$-gate needs $2^t - 1$ calls to the oracle U.*

But as t (i.e. the size of the "counting register") is a metaparameter it can be chosen to balance success probability and runtime using the following equation[42, p. 224 (eq. 5.35)]:

$$\Pr[\text{'success'}] = \Pr\left[|\tilde{\theta} - \theta| \leq \frac{1}{2^\tau}\right] \geq 1 - \frac{1}{2^{x+1} - 4},$$

where $x := t - \tau, x \geq 1$ is the amount of additional bits used for the counting register, which obviously influences the runtime of the algorithm.

B.3 Quantum Fourier Transform

As a subroutine for the QPE algorithm detailed above, the Quantum Fourier Transform (QFT) is used. It converts the amplitudes of the states between the computational basis and the Fourier basis. It is also a main building block of Shors algorithm, but as this is not part of this thesis, only as part of the QPE algorithm, it will not be explained in detail here. For a more detailed explanation see [31, pp. 214-218].

Appendix C

Complexity-Theory *recap*

C.1 Turing machines

C.1.1 Deterministic Turing machines

Definition C.1.1. *A* (Deterministic) Touring Machine *(DTM) is the septtuple* $M = (Q, \Gamma, \Sigma, \delta, q_0, B, F)$*, where:*

- *Q is a finite set of states*
- *Γ is a finite set of tape symbols*
- *$\Sigma \in \Gamma$ is a finite set of input symbols*
- *$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function*
- *$q_0 \in Q$ is the initial state*
- *$B \in \Gamma$ is the blank symbol*
- *$F \subseteq Q$ is the set of final states*

As the formal definition is not that intuitive, one can intuitively think of a Turing machine as a machine that has a tape with an infinite number of cells, each of which can hold a symbol from a finite alphabet Γ. Initially the tape is filled with a finite number of symbols from Σ and the rest of the tape is filled with the blank symbol B. The machine can read and write symbols on the tape, change its state depending on the current state as well as the read symbol (the symbol in the tape cell where the head is currently) and can then move the tape head left or right (using δ, where R denotes moving the head right and L left). The machine has a finite number of states Q, and can be in one of these states at any time, starting with q_0. The machine halts if it is in a final state $f \in F$. f and the tape can then be regarded as the output of the machine.

This already is a very powerful model of computation, as it can (by defintion of computable [50]) compute any computable function. That does not mean it is efficiently computable, though, as it can take up to an infinite amount of time and space to compute a function, but more on that in appendix C.2.

In many cases $F = \{\text{'accept'}, \text{'reject'}\}$ (and the third option, not halting at all).

C.1.2 Non-deterministic Turing machines

A non-deterministic Turing machine (NTM) is probably ($P \overset{?}{\subset} NP$) a more poweful sibling of the deterministic Turing machine since it can have multiple transitions for a given state and symbol and always chooses the “best”.

Formally the only difference lies in the transition function δ which is now a more complicated relation, but the details are not important for this thesis.

Intuitively a NTM can calculate everything a DTM can verify [4].

C.1.3 Probabilistic Turing machines

A probabilistic Turing machine (PTM) is a Turing machine that can make probabilistic decisions. It sits somewhere between a deterministic and a non-deterministic Turing machine, as it also has multiple transitions for a given state and symbol (and therefor multiple execution paths) , but it chooses one of them probabilistically instead of choosing the "best". It suffices to add a second transition function δ' to our definition of the DTM, and now on every step choose randomly between the transitions of δ and δ'.

As we can see the result or success of the machine can now also become probabilistic, which raises new possibilities for complexity classes. Although it is widely conjectured that the NTM is more powerful than PTM or DTM, it is believed that this is not the case for the superiority of PTM over DTM, i.e. randomness might not give a computational advantage [32].

C.1.4 Quantum Turing machines

The Quantum Turing machine (QTM) is a Turing machine that can make use of quantum mechanics. It is superior to the DTM and PTM, as it can simulate any PTM but additionally can use quantum mechanics not computable on any classical system as shown by Bell's theorem [20, p. 11].
When comparing to the classical touring machine, we replace the tape with a quantum register, the states with quantum states (elements of the hilbert space that is the tape alphabet), and the transition function with a unitary operator that is an automorphism of the Hilbert space.
Interestingly the relationship between the QTM and NTM is not yet known [11].

C.2 Complexity Classes

There are already a bunch of established complexity classes, spanned by the different models of computation. A great overview can be found in [2].

P

The smallest complexity class relevant for this thesis is the class of all languages that can be computed by a deterministic Turing machine (DTM) in polynomial time (P). That's because we would regard a language that can be computed or a problem that can be solved in polynomial time as "efficiently" computable.

NP

In contrast to that, the class of all languages that can be computed by a nondeterministic Turing machine in polynomial time (NP), which is a superset of P, is not. Though it is still a big open question whether NP is a *strict* superset of P[1], i.e. whether there is a language that is in NP but not in P, it is widely conjectured that this is the case. A beautiful quote by well known (quantum) complexity researcher Scott Aaronson sums up the situation quite philosophically: "If P=NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in 'creative leaps', no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss;" [8, Scott Aaronson] This is derived by the more intuitive formulation of NP being the class of all languages that can be *verified* by a DTM in polynomial time. The equality of these two formulations lies within the essentially unlimited parallel branching of NTMs,

[1]https://www.claymath.org/millennium-problems

and if a branch of execution exists that computes the language in polynomial time, then this branch can be executed and therefor the language verified in polynomial time on a DTM.

NP-Hard

There is a set of languages in NP, called NP-complete, on which every problem in NP can be reduced to[17]. Proofing that one of these problems is in a specific complexity class C proofs $NP \subseteq C$. One of these NP-complete problems is the "HCP", which is the problem of deciding whether a given graph is hamiltonian, i.e. wheather there exists a cycle in the graph thats visists every note exactly once[3].

If a problem is NP-complete, or not even solvable by a NTM in polynomial time, it is called NP-hard.

BPP

Congruently to NP being created by polynomial NTMs and P being created by polynomial DTMs, the class of all languages that can be computed (with bounded error) by a PTM in polynomial time is called BPP. For simplicity reasons the error often gets bounded to $\frac{1}{3}$ w.l.o.g. since the error can be arbitrarily small when applying a BPP algorithm (with error probability bounded to a fixed $\epsilon < \frac{1}{2}$) multiple times.

In contrast to the conjecture of $P \subsetneq NP$, it is believed that $BPP \overset{?}{=} P$, i.e. a touring machine might not gain anything by using actual randomness, as mentioned in appendix C.1.3.

BQP

The class of all languages that can be computed by a QTM in polynomial time with bounded error probability is called BQP [12]. As with BPP the error probability is often arbitrarily bounded to $\frac{1}{3}$. This class is rather important since it contains the problems that can be solved ‚efficiently' on a quantum computer. And it is believed that $BQP \overset{?}{\supset} P$, i.e. there are probably problems that can be solved ‚efficiently' on a quantum computer but not on a classical computer. It is already proven that $BQP \supseteq BPP$ [12].

PostBQP

An upper bound of BQP is the "class of problems solvable in quantum polynomial time if we take the probability of measuring a basis state with amplitude α to be not $|\alpha|^2$ but $|\alpha|^p$, where p is an even integer greater than 2. ([..] we need to divide all amplitudes by a normalizing factor to make the probabilities sum to 1)"[7]. This class is called BQP with postselection (postBQP) since similar to an NTM it can ‚decide' its execution path / what to measure, i.e. select a state to measure post execution.

This class can formally defined as all problems for which there exists a QTM outputting two qubits (the postselection qubit $|P\rangle$ and the answer qubit $|A\rangle$) such that

- $Pr[A = 1|P = 1] > \frac{2}{3}$ iff answer is yes
- $Pr[A = 0|P = 1] > \frac{2}{3}$ iff answer is no
- $Pr[P = 1] > 0$

If a QTM would have postselection it could e.g. solve database search problems in polynomial time, since it could decide to measure the state that represents the answer to the query.

QMA

Another complexity class spanned by QTMs is Quantum Merlin Arthur (QMA[2]), which is the class of all languages that can be verified by a QTM in polynomial time with bounded error probability. This class could be called the NP of quantum computers as it relates to BQP similar to how NP relates to P [3]. It therefor is believed to be a (strict) superset of BQP, i.e. $QMA \supset BQP$. It is defined as all the decision problems where given a proof (that was generated by computationally unbounded Merlin) that the answer is ‚yes‘ a QTM (Arthur) can verify the proof in polynomial time with bounded error probability, i.e. there exists a proof that can be verified by a QTM in polynomial time with bounded error probability if the answer is ‚yes‘ and if the answer is ‚no‘ the QTM returns false for *any* proof with bounded error probability in polinomial time. If the proof is not given as a quantum state, but as a classical state, it is called quantum classical Merlin Arthur (QCMA) and is a subset of QMA.

PP

Probabilistic Polynomial-Time (PP) is the class of all languages that can be computed by a PTM in polynomial time, but in contrast to BPP the error probability is not bounded, i.e. can can get arbitrarily closer to $\frac{1}{2}$ with bigger input size.

C.3 Relationships between complexity classes and quantum computing

C.3.1 P ⊆ BQP

As P ⊆ BQP a quantum computer can solve any problem in polynomial time that can be solved on a classical computer in polynomial time.

C.3.2 NP and BQP

The HCP (mentioned in appendix C.2) has been tested on quantum computers, although only polynomial speedup compared to classical brute-force solutions could be obtained [4] [35]. But this already shows that quantum computers have a polynomial speedup over classical computers in regard to brute-forcing any NP problem. Indeed, it has been proven that the Grover search algorithm (see section 5.1) can be used to solve any “black-boxed” NP problem in $\mathcal{O}\left(\sqrt{N}\right)$ with N being the cardinality of the input space, whereas the classical brute-force algorithm needs $\mathcal{O}(N)$ steps. On the other hand it has also been shown that these “black-boxed” NP problems cannot be solved in $o(\sqrt{N})$ time on a QTM [11]. This would “not rule out the possibility that NP ⊆ BQP. What these results do establish is that there is no black-box approach to solving NP-complete problems by using some uniquely quantum-mechanical features of QTMs”[11].

But weather a quantum computer can solve any NP problem in polynomial time, i.e. $NP \overset{?}{\subseteq} BQP$, is still an open question. If that were the case, that would not only have disastrous consequences for cryptography but also proof $NP \neq P$, so an answer to this question will likely not be found soon.

As Ph.D. de Beaudrap formulates: “BQP is a *differently* powerful [..] class than NP”[19] since, in comparison to NPMs, QTMs can use destructive interference but lose the ability to ‚choose what to measure‘[5].

As it is proven that the search problem $\notin o(\sqrt{N})$ and therefor $\notin BQP$ it would be proven that $NP \neq P$ if the search problem could be reduced to an NP-complete problem. That results of BQP containing P but

[2]QMA is equal to to quantum interactive proof with one round (QIP(1))

[3]this is more of an opinion than mathematical fact coming from the similarity that both classes verify a given proof in polynomial time. NP on a DTM and QMA on a QTM with the proof being a quantum state. There is also NQP which is the more technical counterpart of NP for QTMs as it contains the problems that produce a non zero amplitude accept state iff the answer is ‚yes‘.

[4]Runtime of AQC is still unknown.

[5]Having both abilities creates the postBQP class described in appendix C.2.

not the search problem and therewith an NP problem. But as easy as it is to reduce any NP-complete problem to the search problem as hard is it vice versa.[6]

C.3.3 PostBQP and PP

It has been proven that the postselection ability of QTMs can be used to solve exactly all problems in the class PP in polynomial time, i.e. postBQP = PP [7]. Therefor PP serves as an upper bound for BQP.

C.4 Quantum Complexity

moved to section 3.4.2

[6]If the whole (exponentially sized) database to search is given as an input to the verifier DTM, the problem would be in P. If the database is virtual and given via an efficiently computable algorithm, the search problem would be in NP, but the proof for Grover being optimal and any solution being $\notin o(\sqrt{N})$ would no longer hold as the problem is no longer black boxed.

Appendix D

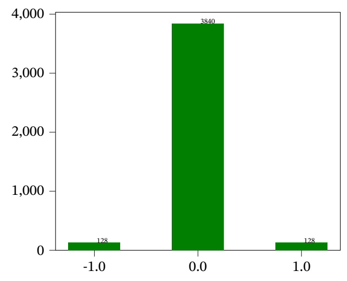
Supplementary Material

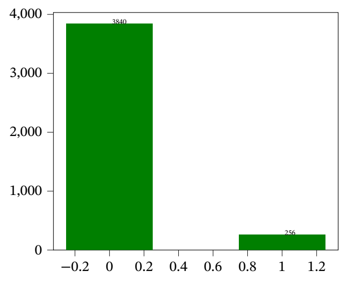
Code

```
section .data
    array db 8, 1, 14, 14, 5, 19,    11 ; Example array with integer values
    array_size equ $ - array ; Calculate the size of the array
    result db 0 ; Reserve space for the result

section .text
    global _start

_start:
    mov ecx, array_size ; Load the size of the array into ecx
    lea esi, [array] ; Load the address of the array into esi
    xor eax, eax ; Set accumulator (sum) to 0

sum_loop:
    add al, [esi] ; Add the value in the array to the accumulator
    inc esi ; Point to the next element in the array
    loop sum_loop ; Repeat the loop until ecx (array_size) becomes 0

output:
    mov [result], al ; Store the sum in result

    ; Output the sum on the console
    mov eax, 4 ; System call number for write
    mov ebx, 1 ; File descriptor for standard output (stdout)
    mov ecx, result ; Address of the result
    mov edx, 1 ; Number of bytes to write (1 byte for the result)
    int 0x80 ; Execute system call

end:
    ; Exit the program
    mov eax, 1
    xor ebx, ebx
    int 0x80
```

Listing D.1: Whole working x86 assembler code for a summation.

Other


(a) Count of how many different masks (m_x, m_y) have a certain correlation $c(\bar{t}_{m_x,m_y}(f))$, for an affine 4-to-8-bit function.


(b) Count of how many different masks (m_x, m_y) have a certain "goodness", i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for an affine 4-to-8-bit function.

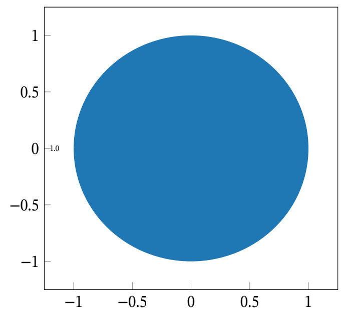
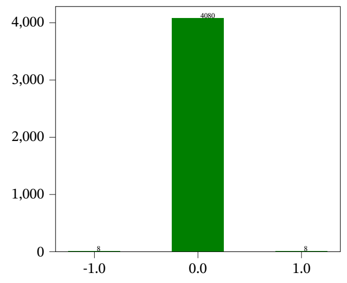
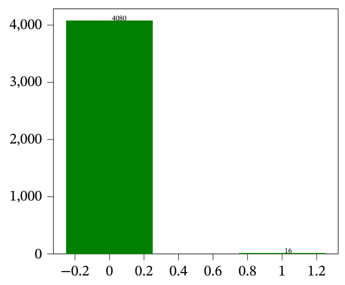

(c) Probability of measuring a mask with a certain goodness, i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for an affine 4-to-8-bit function.

Figure D.1: Analysis of an affine 4-to-8-bit function.


(a) Count of how many different masks (m_x, m_y) have a certain correlation $c(\bar{t}_{m_x,m_y}(f))$, for an affine 8-to-4-bit function.


(b) Count of how many different masks (m_x, m_y) have a certain "goodness", i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for an affine 8-to-4-bit function.

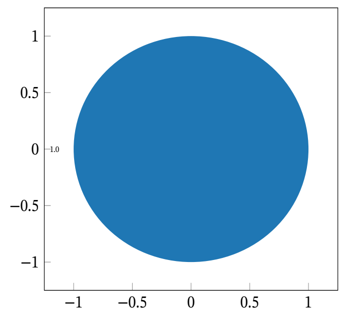

(c) Probability of measuring a mask with a certain goodness, i.e. absolute correlation $|c(\bar{t}_{m_x,m_y}(f))|$, for an affine 8-to-4-bit function.

Figure D.2: Analysis of an affine 8-to-4-bit function.

Glossary

$F(\mathbb{F}_2^n, \mathbb{F}_2^m)$ The set of all possible functions $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$.. 10, 23

basic approximate count algorithm The basic quantum counting algorithm with exponential search as described at the top of [14, p. 22].. 50

eigenstate an eigenvector of a unitary operator. 46

Grover diffusion operator the non-changing part of the grover operator that flips along the mean: $D := -H \cdot R \cdot H$. 40, 46, 70

Bibliography

[1] *Bent Function - an overview | ScienceDirect Topics*. https://www.sciencedirect.com/topics/mathematics/bent-function.

[2] *Complexity Zoo*. https://complexityzoo.net/Complexity_Zoo.

[3] *Computers and Intractability: A Guide to the Theory of NP-Completeness (Michael R. Garey and David S. Johnson) - ProQuest*. https://www.proquest.com/openview/e4a13c8c3d470245a831eb2f3c3d2487/1?cbl=30748.

[4] *Lec13.pdf*. https://people.seas.harvard.edu/~cs125/fall16/lec13.pdf.

[5] *Quantum algorithm for the finding of Boolean function's linear structures*. https://ar5iv.labs.arxiv.org/html/1404.0611.

[6] *Fast Walsh–Hadamard transform*, Wikipedia, (2023).

[7] S. Aaronson, *Quantum Computing, Postselection, and Probabilistic Polynomial-Time*, Dec. 2004.

[8] ———, *Reasons to believe*, Sept. 2006. https://scottaaronson.blog/?p=122.

[9] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf, *Quantum Lower Bounds by Polynomials*, Sept. 1998.

[10] E. Bellini, M. Sala, and I. Simonetti, *Nonlinearity of Boolean Functions: An Algorithmic Approach Based on Multivariate Polynomials*, Symmetry, 14 (2022), p. 213.

[11] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and Weaknesses of Quantum Computing*, SIAM Journal on Computing, 26 (1997), pp. 1510–1523.

[12] E. Bernstein and U. Vazirani, *Quantum complexity theory*, in Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, STOC '93, New York, NY, USA, June 1993, Association for Computing Machinery, pp. 11–20.

[13] F. G. Brandão, W. Chemissany, N. Hunter-Jones, R. Kueng, and J. Preskill, *Models of Quantum Complexity Growth*, PRX Quantum, 2 (2021), p. 030316.

[14] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, *Quantum Amplitude Amplification and Estimation*, vol. 305, 2002, pp. 53–74.

[15] G. Brassard, P. Hoyer, and A. Tapp, *Quantum Counting*, arXiv:quant-ph/9805082, 1443 (1998), pp. 820–831.

[16] L. Budaghyan and A. Pott, *On differential uniformity and nonlinearity of functions*, Discrete Mathematics, 309 (2009), pp. 371–384.

[17] S. A. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71, New York, NY, USA, May 1971, Association for Computing Machinery, pp. 151–158.

[18] N. Datta, *Discriminating between unitary quantum processes*. https://www.youtube.com/watch?v=gHEjszXSjMQ.

[19] N. de Beaudrap, *Why is a quantum computer in some ways more powerful than a nondeterministic Turing machine?*, Mar. 2018.

[20] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proceedings of the Royal Society of London Series A, 400 (1985), pp. 97–117.

[21] C. Dobraunig, M. Eichlseder, and F. Mendel, *Heuristic Tool for Linear Cryptanalysis with Applications to CAESAR Candidates*, in Advances in Cryptology – ASIACRYPT 2015, T. Iwata and J. H. Cheon, eds., Berlin, Heidelberg, 2015, Springer, pp. 490–509.

[22] C. Dohotaru and P. Hoyer, *Exact quantum lower bound for Grover's problem*. https://arxiv.org/abs/0810.3647v1, Oct. 2008.

[23] B. Ferenc Balazs and I. Sandor Imre, *Quantum Computing and Communications: An Engineering Approach*, Wiley-Blackwell, 2013.

[24] D. Floess, E. Andersson, and M. Hillery, *Quantum algorithms for testing and learning Boolean functions*, Mathematical Structures in Computer Science, 23 (2013), pp. 386–398.

[25] V. Giovannetti, S. Lloyd, and L. Maccone, *Quantum random access memory*, Physical Review Letters, 100 (2008), p. 160501.

[26] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, STOC '96, New York, NY, USA, July 1996, Association for Computing Machinery, pp. 212–219.

[27] S. Harris, *Digital Design and Computer Architecture*.

[28] M. Hermelin, J. Y. Cho, and K. Nyberg, *Multidimensional Linear Cryptanalysis*, Journal of Cryptology, 32 (2019), pp. 1–34.

[29] M. Hillery and E. Andersson, *Quantum tests for the linearity and permutation invariance of Boolean functions*, Physical Review A, 84 (2011), p. 062329.

[30] W. Hoeffding, *Probability Inequalities for Sums of Bounded Random Variables*, J. Am. Stat. Assoc., 58 (1963), pp. 13–30.

[31] M. Homeister, *Quantum Computing verstehen: Grundlagen - Anwendungen - Perspektiven*, Computational intelligence, Springer Vieweg, Wiesbaden [Heidelberg], 6., erweiterte und überarbeitete auflage ed., 2022.

[32] R. Impagliazzo and A. Wigderson, *$P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma*, in Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, STOC '97, New York, NY, USA, May 1997, Association for Computing Machinery, pp. 220–229.

[33] P. Kaye, R. Laflamme, and M. Mosca, *An Introduction to Quantum Computing*, OUP Oxford, Nov. 2006.

[34] H.-W. Li and L. Yang, *A quantum algorithm to approximate the linear structures of Boolean functions*, Mathematical Structures in Computer Science, 28 (2018), pp. 1–13.

[35] A. Mahasinghe, R. Hua, M. Dinneen, and R. Goyal, *Solving the Hamiltonian Cycle Problem using a Quantum Computer*, Jan. 2019, pp. 1–9.

[36] S. Maitra, B. Mandal, T. Martinsen, D. Roy, and P. Stănică, *Tools in Analyzing Linear Approximation for Boolean Functions Related to FLIP*, in Progress in Cryptology – INDOCRYPT 2018, D. Chakraborty and T. Iwata, eds., Lecture Notes in Computer Science, Cham, 2018, Springer International Publishing, pp. 282–303.

[37] A. K. Malviya and N. Tiwari, *Linear approximation of a vectorial Boolean function using quantum computing*, Europhysics Letters, 132 (2020), p. 40001.

[38] P. D. M. Margraf, *Vorlesungsskript Kryptoanalyse Symmetrischer Verfahren*, 2020.

[39] M. MATSUI, *Linear Cryptanalysis Method for DES Cipher*, in Advances in Cryptology — EUROCRYPT '93, vol. 765, Springer Berlin Heidelberg, Berlin, Heidelberg, July 1993, pp. 386–397.

[40] M. MATSUI AND A. YAMAGISHI, *A New Method for Known Plaintext Attack of FEAL Cipher*, in Advances in Cryptology — EUROCRYPT' 92, R. A. Rueppel, ed., Lecture Notes in Computer Science, Berlin, Heidelberg, 1993, Springer, pp. 81–91.

[41] S. MESNAGER, *Bent Vectorial Functions*, in Bent Functions: Fundamentals and Results, S. Mesnager, ed., Springer International Publishing, Cham, 2016, pp. 305–327.

[42] M. A. NIELSEN AND I. L. CHUANG, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. https://www.cambridge.org/highereducation/books/quantum-computation-and-quantum-information/01E10196D0A682A6AEFFEA52D53BE9AE, Dec. 2010.

[43] D. K. PARK, F. PETRUCCIONE, AND J.-K. K. RHEE, *Circuit-Based Quantum Random Access Memory for Classical Data*, Scientific Reports, 9 (2019), p. 3949.

[44] P. ROGAWAY, *Nonce-Based Symmetric Encryption*, in Fast Software Encryption, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, B. Roy, and W. Meier, eds., vol. 3017, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 348–358.

[45] O. S. ROTHAUS, *On "bent" functions*, J. Combin. Theory Ser. A, 20 (1976), pp. 300–305.

[46] D. RUSCA, T. VAN HIMBEECK, A. MARTIN, J. B. BRASK, W. SHI, S. PIRONIO, N. BRUNNER, AND H. ZBINDEN, *Self-testing quantum random-number generator based on an energy bound*, Physical Review A, 100 (2019), p. 062338.

[47] P. SHOR, *Algorithms for quantum computation: Discrete logarithms and factoring*, in Proceedings 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 1994, IEEE Comput. Soc. Press, pp. 124–134.

[48] P. SHYAMSUNDAR, *Non-Boolean Quantum Amplitude Amplification and Quantum Mean Estimation*, Feb. 2021.

[49] Y. SUZUKI, S. UNO, R. RAYMOND, T. TANAKA, T. ONODERA, AND N. YAMAMOTO, *Amplitude estimation without phase estimation*, Quantum Information Processing, 19 (2020), p. 75.

[50] A. M. TURING, *Systems of logic based on ordinals*, Proceedings of the London Mathematical Society, Series 2, 45 (1939), pp. 161–228.

[51] L. ZHENG, J. PENG, H. KAN, AND Y. LI, *Constructing vectorial bent functions via second-order derivatives*, May 2019.