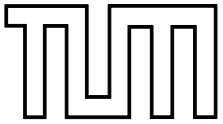
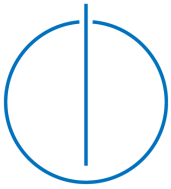


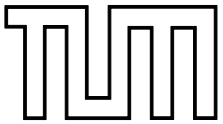
DEPARTMENT OF INFORMATICS
TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Information Systems

Linear Cryptanalysis on Quantum Systems

Lorenzo Pietro Brazzi





DEPARTMENT OF INFORMATICS
TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Information Systems

Lineare Kryptoanalyse auf Quantensystemen

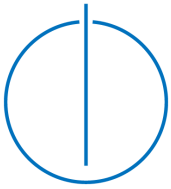
Linear Cryptanalysis on Quantum Systems

Author: Lorenzo Pietro Brazzi

Supervisor: Prof. Dr. Christian Mendl

Advisor: Prof. Dr. Christian Mendl

Submission date: 15.01.2021



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, January 13, 2021

Lorenzo Pietro Brazzi

1. Abstract

It's safe to say that over recent years *Quantum Computing* is attracting a lot of interest. Driving force for that are huge achievements in the development of new quantum algorithms like Shor's algorithm, quantum phase estimation and Grover's algorithm. Due to the massive potential of quantum computation and quantum information processing, it has caught interest of a lot of researchers in a variety of different research fields. Many of those techniques are implemented in cryptography and cryptanalysis, in particular *Linear Cryptanalysis*. This paper is about combining the two principles even more. I offer a walkthrough through the implementation of an important portion of the linear cryptanalysis attack an a quantum computer. This enables a significant speedup in computation and therefore a greater overall complexity. My framework consists of a couple of components. Firstly, I give an introduction to the basic idea about the classical algorithm of linear cryptanalysis. From there I describe which portion of the attack can be accelerated by making use of quantum devices and quantum information processing. I do this by applying the fast Fourier transform (FFT). The applicability of the FFT gives me the possibility of transferring that part of the algorithm to the quantum space. To actually realize this I put forward two different ways of implementing the FFT algorithm on a suitably large enough quantum computer. On the one hand we present the well-known quantum Fourier transform (QFT). On the other hand we describe a dedicated circuit that models the FFT in a so-called quantum fast Fourier transform (QFFT) circuit using quantum arithmetic operations as gates.
Additionally, since the „new" algorithm for linear cryptanalysis is semi-classical and semi-quantum-computational, I introduce a technique which enables the encoding of classical data into quantum states. Finally, we compare the complexity difference between the various approaches.

Contents

2. Introduction

In the 21 century, a field of physics that was previously cautiously touched, is starting to grow and show its full potential: **Quantum Mechanics**. Quantum mechanics is a field of physics that establishes a set of negative rules stating things that can not be done. The theory behind the existence of invisible quantum objects that are able to entangle with each other even being lightyears apart can often appear to sound like an ingenious movie script. Nevertheless, after recognizing its potential, researchers startet digging into the variety of potential application fields, making use of advanced research achievements realized by big names in the field. One of the biggest and most interesting applications of Quantum Mechanics is *Quantum Computing*. Among other things, thanks to technological progress, research and the establishment of the right resources, physicists and computer scientists have been able to put this ungraspable concept into reality. Quantum Computing brings together ideas from classical information theory, computer science, ad quantum physics. In fact, a quantum computer is not a device whose operations are governed by the laws of quantum mechanics, but rather whose operations exploit certain very special transformations of its internal state. The theory of quantum mechanics in itself only allows these peculiar transformations to take place under very carefully controlled conditions.

The question of how properties of this theory could serve existing problems arises. The phenomenon of entanglement and quanta was quickly discovered to have a huge impact on the optimization of algorithms that fundamentally were considered „unbreakable“. The consequence was that it was targeted from computer scientist with the aim to exploit it one of the most troubling and socially asked for fields of computation today: cryptography. The immense impact that quantum computation has shown to have on traditional cryptography, combined with the intensified public discussion over increased privacy and security, *quantum cryptography* might be the first commercial application fo quantum mechanics. On this account, over the last years, many proposals have been published on different angles from which a quantum system could affect cryptography. It's important to look at it from an external prospective, since a potential quantum computer could not only be useful for developing new cryptographic methods, but also for attacking existing ones. Perhaps one of the most prominent discoveries came from Peter Shor in 1999, where he discovered a quantum algorithm, that provided a sufficiently large-scaled quantum computer, would realize integer factorization and discrete logarithms in polynomial time [18]. Since integer factorization is one of the crucial arguments supporting safety in asymmetric cryptography, quantum computation makes for example RSA insecure [10]. Even though a large-enough quantum computer, which would be able to perform these kinds of algorithms, doesn't yet exist, discoveries like this alarmed the experts in the cryptography community to dedicate more and more research into possible solutions and new techniques. Quantum algorithms have been shown to be very effective threats to asymmetrical cryptographic methods. What about symmetric cryptography? Publications like the famous [13] show that also symmetric cyphers can be directly affected by quantum computational devices. Anyway, not to the same extent as asymmetrical cryptography. Although not a spe-

cific mathematical component of symmetric key ciphers can be compromised significantly by quantum computers, that is not sufficient to say that they are immune. Actually, the security within these algorithms is known to rely mainly on the resistance against particular attacks, which implies that only cryptanalysis and security evaluation can confidently approve a primitive [10]. The focus of this paper is on the implementation and optimization, by applying quantum computation, of a specific cryptanalysis technique: *Liner Cryptanalysis*. Introduced by Mitsuru Matsui in 1993, together with differential cryptanalysis, linear cryptanalysis quickly emerged (and still is) as one of the predominant attacks on all block-cyphers. Specifically, it was intended to harm the widely-known DES-algorithm (Data Encryption Standard). In this paper I will go through the conventional linear cryptanalysis algorithm and show which parts of the attack can be implemented on a quantum device. I do this by making use of a well-known classical algorithm, the fast Fourier Transform (FFT). I show that it can be implemented on a quantum device in different ways, one of the being the *Quantum Fourier Transform (QFT)*, the other being a dedicated quantum circuit that I call quantum-FFT (QFFT). Additionally, I present different approaches on how to encode classical data into a quantum state, which we obviously need for our computation. This is required because our „new“linear cryptanalysis algorithm is partly classical and partly quantum-computational and we need to transition from one realm to the other efficiently.

The aim of the following two chapters is to give a brief introduction to the building blocks the will help us reach our endeavor. This first part is going to be about the algorithm that we want to work with: linear cryptanalysis. After that, I will describe how the attack can be specifically improved by quantum application. From there, I show how the two spaces can be connected by presenting different approaches on data encoding, as well as a complexity analysis of our improvements.

3. Linear Cryptanalysis

As mentioned before, linear cryptanalysis is one of the most effective attacks against block ciphers. Block ciphers are performed in specific *modes of operation*, to make use of messages of variable length to provide specific security properties. In classical cryptography, the security of the block cipher itself is used to imply the security of the operation that it is used for. It has been shown that the robustness of these modes of operation for block ciphers varies when it comes to their implementation on quantum devices [10]. Although the combination with modes of operation is important to solidify the whole picture, in this section of the paper I want to focus on the cryptanalysis technique that makes block cyphers vulnerable in a quantum setting.

Developed by Matsui in 1993, linear cryptanalysis studies statistical linear relations between **bits** of the plaintexts, the corresponding ciphertexts and the keys they are encrypted under. These relations are used to predict values of bits of the actual key in a Substitution-Permutation-Netweork (SPN), assuming that many plaintexts and their corresponding ciphertexts are known. Its goal is to approximate a round function of the non-linear part of an encryption setting with a linear (or affine) function. This approximation is achieved by making use of so-called linear masks of encryption process, which in this setting is realized by *S-boxes*. Through linear cryptanalysis we can make use of the parity of each round's input and output masks, which allow us to evaluate the correlation between the approximation and the full cipher using the so-called *Piling-Up lemma*. In this section of the paper I will work thorough a simplified version of what was described in the previous paragraph, since the fundamental understanding of linear cryptanalysis will reveal the effects of a quantum computational device in this context.

As mentioned before, linear cryptanalysis is a known plaintext attack on a SPN, as displayed in Figure 1. A SPN hast a couple of components that recur for all n rounds of the cipher:

- Derived from a main key, each round, a **round key** is applied to the bit-sequence of the previous round with a mod-2 bit-wise operation, typically exclusive-OR $\oplus$ (XOR).
- **S-Boxes.** S-boxes (substitution boxes) are the part of the cipher that add non-linearity. This is achieved by the S-box's property of nonlinear mapping [8]: that is, that the output bits cannot be represented as a linear operation on the input bits. Effectively, an S-box can be represented by a lookup-table of 2^m m-bit values, where every m-bit block that goes into an S-box can be represented by 2^m m-bit values. For our explanation of linear cryptanalysis we will assume that all the S-boxes are equal, while in general, as in DES, S-boxes in a round are different. In Figure 1, the S-boxes are represented by $S_1, S_2, ..., S_\ell$.
- **P-Boxes.** P-boxes (permutation boxes) realize the portion of the round that transposes the bits, that is, changes their positions. Effectively, this means that the output of an P-box in round $t-1$ is fed into the input of *another* P-boxe's input in round t. Both S- and P-boxes satisfy Shannon's **confusion** and **diffusion** properties [17].

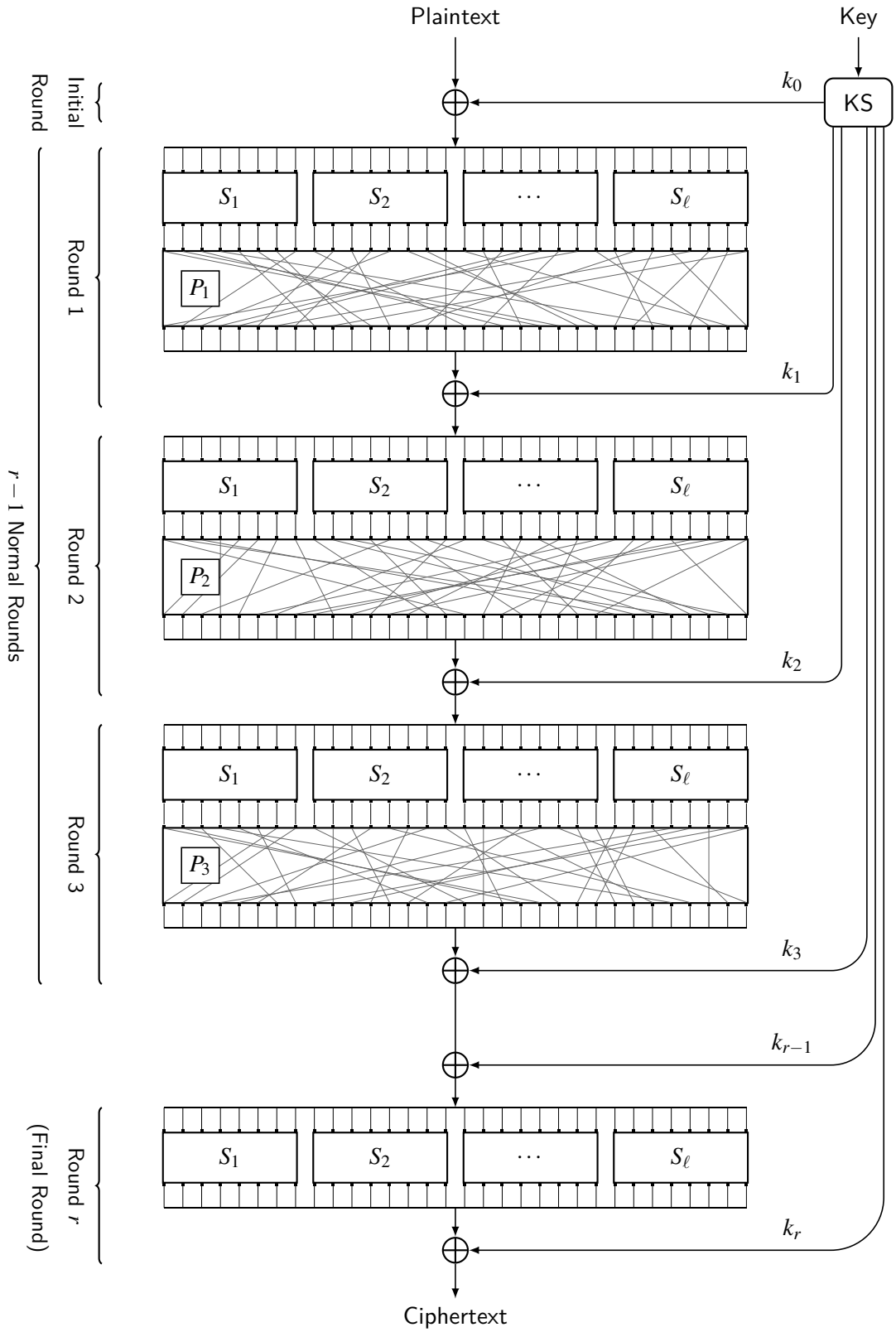


Figure 1. Classic Substitution-Permutation Network using [19].

In a known plaintext attack the attacker knows a set of plaintexts with the corresponding ciphertexts. Although this set in most cases is random, since we focus on the common process that they all go through, it's enough to perform cryptanalysis. The basic idea then is to approximate the portion of the cipher which is non-linear with a linear expression that represents an XOR „sum"of u input bits and v output bits:

$$X_0 \oplus X_1 \oplus \cdots \oplus X_{u-1} \oplus Y_0 \oplus Y_1 \oplus \cdots \oplus Y_{v-1} = 0 \qquad (1)$$

Where X_i represents the i-th input bit and Y_j represents the j-th output bit.
It is important to note that in DES, and in most other SPN, all operations except for the S-boxes are linear, which implies that it's sufficient to derive linear relations for the S-boxes to perform linear cryptanalysis [2]. In a cryptographically ideal scenario, due to the randomizing properties of the S-boxes, selecting random values for u and v and injecting them into the above term, would lead to a probability p of equation (1) actually holding true of 50%. This only means that an S-box that receives an input with property X gives an output with property Y exactly half of the time, which just means that there is no linear relation between property X and property Y. Observe that $p = 1$ as a result for term (1) would imply that that specific setting is a perfect representation of the cipher, which would therefore be maximally weak. Furthermore, since affine functions in a mod-2 addition system can be effectively seen as the complement of a linear function, the same can be said about $p = 0$. Therefore, the aim in linear cryptanalysis is to find an expression similar to the above, which has a high *or* low probability of being valid. In fact, linear cryptanalysis exploits exactly this deviation from 50%. That is, the further away the probability outcome $|p - 50\%|$ is from $1/2$, the more effective linear cryptanalysis can be deployed.
The next step in linear cryptanalysis is to derive the linear relations for each S-box by choosing a subset of input and output bits, calculating their parity for every possible input of the S-box, and counting the number of whose subset's parity is zero. What this means exactly, and how it is done, will be explained in the next section.

3.1 Exploiting S-boxes

Now that we have a rough understanding about how SPNs work, I want to look in more detail into the mechanics creating linear approximations for the cipher. Therefore, I will look at a more reduced version of a SPN with three rounds and four S-boxes per round, as it's displayed in Figure 2. Here we consider four 4-bit blocks, which lead to an entirety of 16 possible output sequences per S-box. Note that the permutation layer is a lot more clear in Figure 2 than in Figure 1. Now you can see exactly how the permutation is performed, and moreover, the difference between S-boxes and P-boxes becomes clearer. The S-box acts like a *blackbox*, performing its non-linear mapping, while the P-box's permutation is pretty straight-forward.

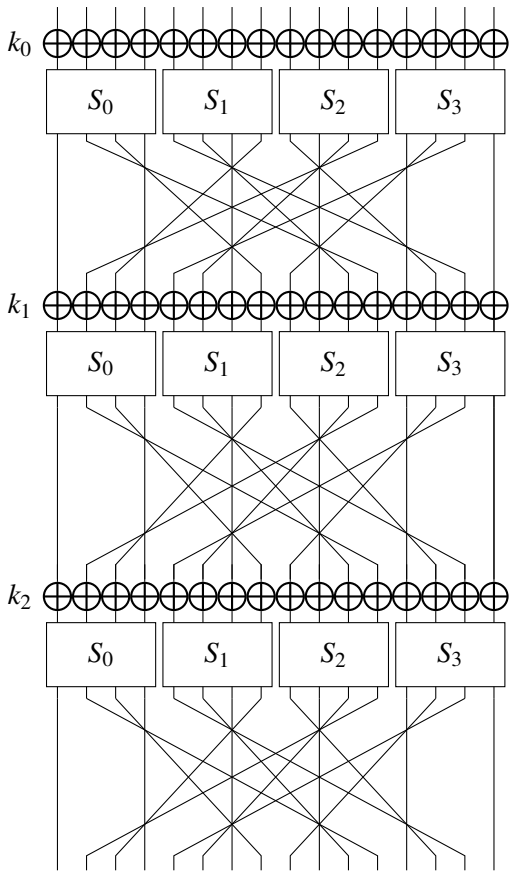


Figure 2. SPN reduced to 3 rounds using [19].

Now, given the knowledge that the key to performing linear cryptanalysis is the establishment of a linear approximation of the cipher's S-boxes and that all S-boxes in our setting are alike, it is shown to be possible to examine the S-boxes of successive rounds and combine the derived linear approximations. The first step to perform now is simply to go through all linear approximations constructed as in equation (1). This is done by defining which X_i and which Y_j to consider. The parameter that gives us this definition is a so-called **mask**. The usefulness of a mask can be shown easily through the following example. Since we deal with a linear approximation of a 4-bit S-box, we can represent the term of this linear approximation by

$$\alpha_0 \cdot X_0 \oplus \alpha_1 \cdot X_1 \oplus \alpha_2 \cdot X_2 \oplus \alpha_3 \cdot X_3 \qquad (2)$$

with $\alpha_i \in \{0,1\}$.

Now the mask comes into play. The mask is a hexadecimal value, that represented in the binary format $a_0a_1a_2a_3$ (where a_0 is the most significant bit) can be inserted into equation (2) to tell me which bits are important to us. For instance, if I apply arbitrarily chosen masks 5, 9, B and F we get the following terms:

$$5 \xrightarrow{\text{binary}} 0101 \implies X_1 \oplus X_3 = 0$$

$$9 \xrightarrow{\text{binary}} 1001 \implies X_0 \oplus X_3 = 0$$

$$B \xrightarrow{\text{binary}} 1011 \implies X_0 \oplus X_2 \oplus X_3 = 0$$

$$F \xrightarrow{\text{binary}} 0101 \implies X_0 \oplus X_1 \oplus X_2 \oplus X_3 = 0$$

Notice the application of the mask only on the X values of equation (1). This is the reason why in linear cryptanalysis there are both an **input mask** and an **output mask**. The output mask is responsible for the definition of the Y values. Hence, in my system a linear equation using input mask $= 3$ and output mask $= 9$ would look like this:

$$X_2 \oplus X_3 = Y_0 \oplus Y_3$$

Now let's get into a little more detail about the entirety of the attack. Let's assume that by applying all possible input values of X to the cipher, I get output values for Y as described in Table 3. This helps me get a first feeling of what a linear approximation of an S-box means. Provided this information, I can start examining different realizations of linear approximations and calculate their probability-deviance from $1/2$. Let's take the following equations for instance:

X_0	X_1	X_2	X_3	Y_0	Y_1	Y_2	Y_3
0	0	0	0	1	1	1	0
0	0	0	1	0	1	0	0
0	0	1	0	1	1	0	1
0	0	1	1	0	0	0	1
0	1	0	0	0	0	1	0
0	1	0	1	1	1	1	1
0	1	1	0	1	0	1	1
0	1	1	1	1	0	0	0
1	0	0	0	0	0	1	1
1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0
1	0	1	1	1	1	0	0
1	1	0	0	0	1	0	1
1	1	0	1	1	0	0	1
1	1	1	0	0	0	0	0
1	1	1	1	0	1	1	1

Table 1. Example of linear approximation of S-box.

$$X_1 \oplus X_2 = Y_0 \oplus Y_2 \oplus Y_3 \qquad (3)$$

$$X_2 \oplus X_3 = Y_0 \oplus Y_3 \qquad (4)$$

By inserting the X_i values from Table 3 into the give equations (3) and (4), you can see that the linear approximations hold for 12 *out of 16* and 2 *out of 16* respectively. Therefore the probability deviations of these approximations are $|12/16 - 1/2 = 1/4|$ and $|2/16 - 1/2 = -3/8|$. As you can see, the second approximation holds for only 2 of the 16 inputs. At first intuition this may look like a bad approximation, but recall that in linear cryptanalysis we are mainly interested in the absolute value of the probability deviating from $1/2$. Actually, approximation (4) is one of the before-mentioned affine approximations.
So, how do I proceed? To get more information our of our S-box, I want to know the actual probability of a linear equation holding true. To reach this information, I obviously need to set up linear equations first, so that they can be tested. When this is done, there is a neat way of displaying my results. Therefore, the next step is to create a so-called **linear approximation table**. This is constructed as follows: I „mask" every possible of the 16 inputs with every possible input mask. The same is to be done for all possible outputs (here applying every possible output mask). By comparing every single one of those implementations, we get a table that for every combination of possible input and output mask tells you the amount of times a linear approximation formed by these two masks held true when tested against all 16

		Output mask															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
Input Mask	**0**	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	-2	-2	0	0	-2	6	2	2	0	0	2	2	0	0
	2	0	0	-2	-2	0	0	-2	-2	0	0	2	2	0	0	-6	2
	3	0	0	0	0	0	0	0	0	2	-6	-2	-2	2	2	-2	-2
	4	0	2	0	-2	-2	-4	-2	0	0	-2	0	2	2	-4	2	0
	5	0	-2	-2	0	-2	0	4	2	-2	0	-4	2	0	-2	-2	0
	6	0	2	-2	4	2	0	0	2	0	-2	2	4	-2	0	0	-2
	7	0	-2	0	2	2	-4	2	0	-2	0	2	0	4	2	0	2
	8	0	0	0	0	0	0	0	0	-2	2	2	-2	2	-2	-2	-6
	9	0	0	-2	-2	0	0	-2	-2	-4	0	-2	2	0	4	2	-2
	A	0	4	-2	2	-4	0	2	-2	2	2	0	0	2	2	0	0
	B	0	4	0	-4	4	0	4	0	0	0	0	0	0	0	0	0
	C	0	-2	4	-2	-2	0	2	0	2	0	2	4	0	2	0	-2
	D	0	2	2	0	-2	4	0	2	-4	-2	2	0	2	0	0	2
	E	0	2	2	0	-2	-4	0	2	-2	0	0	-2	-4	2	-2	0
	F	0	-2	-4	-2	-2	0	2	0	0	-2	4	-2	-2	0	2	0

Table 2. Linear approximation table.

possible inputs. A tabular presentation of such a table has been built for our exemplary cipher and is displayed in Table 2. To every entry in the table I have already subtracted 8, so that a division by 16 would give me directly the deviation from $1/2$. By subtracting this deviation from $1/2$ I get the probability of that linear equation actually holding true. Notice here, that an absolute non-linearity of the S-boxes would lead to all entries being 8. Since that is not the case, this is what we can exploit for our linear cryptanalysis attack.

3.2 Combining the S-boxes

Now that we know how to approximate S-boxes into linear expressions, the next and last step of this attack is to combine them into a single expression that characterizes the whole cipher.

3.2.1 Piling-Up-Lemma

The *Piling-Up Lemma* is a concept introduced by Matsui [7] that defines how to merge independent linear expressions to create a single compact linear expression. As you can already imagine, this is a very helpful tool in linear cryptanalysis, since the goal is to combine the linear approximations of all the single S-boxes into a definitive linear approximation for the whole system. The important premise here is that the probability of n random independent binary variables X as in expression (1) is

$$Pr(X_0 \oplus X_1 \oplus \cdots \oplus X_{n-1}) = 1/2 + 2^n \prod_{i=0}^{n-1} \varepsilon_i \qquad (5)$$

where ε represents the before-mentioned deviation from $1/2$, specifically called *bias*, whose definition is implied from (5) as follows:

$$\varepsilon_{X_0 \oplus X_1 \oplus X_2 \oplus \cdots \oplus X_{n-1}} = 2^n \prod_{i=0}^{n-1} \varepsilon_i$$

Now let's consider four three binary variables X_i (with $i \in 0,1,2$) as linear approximations of S-boxes as well as the following properties: $Pr(X_0 \oplus X_1) = 1/2 + \varepsilon_{X_0 \oplus X_1}$ and $Pr(X_1 \oplus X_2) = 1/2 + \varepsilon_{X_1 \oplus X_2}$. By the assumption that all concerned random binary variables are independent, the *Piling-Up lemma* can be applied and to output $X_0 \oplus X_2$ as follows:

$$Pr(X_0 \oplus \cdots \oplus X_2) = Pr([X_0 \oplus \cdots \oplus X_1] \oplus [X_1 \oplus \cdots \oplus X_2])$$

$$\Downarrow$$

$$Pr(X_0 \oplus \cdots \oplus X_2) = 1/2 + 2\varepsilon_{X_0 \oplus X_1} \varepsilon_{X_1 \oplus X_2}$$

This leaves us with a cumulative bias of $\varepsilon_{X_0 \oplus X_2} = 2\varepsilon_{X_0 \oplus X_1} \varepsilon_{X_1 \oplus X_2}$, which tells us that $X_0 \oplus X_2 = 0$ is analogous to a cipher approximation where the intermediate bit X_1 is eliminated [8].

3.2.2 Creating one linear expression

In this section, I will merge all the knowledge gained about linear cryptanalysis so far and use it to approximate a couple of rounds of an illustrative 3-round-cipher with 4 S-boxes in each round (as depicted in Figure 2). To do that, some notation has to be defined. In every round, the input and output of the S-boxes are defined by $I_{x,y}$ and $O_{x,y}$ respectively, where y defines the bit of the block (numbered from 0 to 15) in round x. Likewise I define $K_{x,y}$ as the key that is applied each round (before coming into the S-box). Having said that, here are specified a couple of linear approximations for the S-boxes and its probability in each round (assuming that all S-boxes in one round perform the same action):

$$Round\ 0 :\longrightarrow X_0 \oplus X_1 \oplus X_2 = Y_3 \quad probability : 14/16$$

$$Round\ 1 :\longrightarrow X_3 = Y_0 \oplus X_2 \quad probability : 2/16$$

$$Round\ 2 :\longrightarrow X_2 = Y_1 \oplus Y_3 \quad probability : 4/16$$

The attack starts with the trivial observation that I get the input I_0 for the first round by XORing the plaintext P with the Key K_0. Since I assumed that all the S-boxes in one round are equal, in this example it is not relevant which plaintext block is observed in our approximation. Therefore, I will start by looking at the first block of input I. By applying our linear expression for the first round to the input bits of the first block, the result shows

$$I_{0,0} \oplus I_{0,1} \oplus I_{0,2} = (P_0 \oplus K_{0,0}) \oplus (P_1 \oplus K_{0,1}) \oplus (P_2 \oplus K_{0,2}) = O_{0,3}$$

with probability $7/8$. Keep in mind that the biases can easily be derived from the single probabilities. Now I can carry on with this pattern into the next rounds. By looking at the

second round, it clarifies that $I_{1,3} = O_{1,0} \oplus O_{1,2}$ with probability $1/8$, or rather

$$O_{0,3} \oplus K1,3 = O_{1,0} \oplus O_{1,2}$$

which, by application of the Piling-Up Lemma and combination with the first rounds, has a probability of $1/2 + 2(13/16 - 1/2)(1/8 - 1/2) = \mathbf{7/32}$.
To add a little more complexity to the last round, let's assume that after the second round of S-boxes, the permutation has lead to the first bit of the first block being mapped to the third bit of the second block ($I_{2,6}$ since it's the 10th bit) for the third round. Also, the third bit of the first block being mapped to the third bit of the fourth block ($I_{2,14}$ since it's the 10th bit) for the third round.
Now the following is required

$$I_{2,6} = O_{2,5} \oplus O_{2,7} \qquad \text{AND} \qquad I_{2,14} = O_{2,13} \oplus O_{2,15}$$

Finally, for the third and last round, by combining all terms leading up to the above equations and applying the Piling-Up Lemma, I get

$$P_0 \oplus P_1 \oplus P_2 \oplus K_{0,0} \oplus K_{0,1} \oplus K_{0,2} \oplus K_{1,3} \oplus K_{2,7} \oplus K_{2,14} \oplus O_{2,5} \oplus O_{2,7} \oplus O_{2,13} \oplus O_{2,15} = 0$$

with probability $1/2 + 2^2(1/8 - 1/2)(1/4 - 1/2) = \mathbf{7/8}$. Note here that for any arbitrary cipher, the mod-2 binary sum $\sum_{\oplus} K$ of all the key bits can only either result in 0 or 1, which means that this linear expression holds with probability $7/8$ OR $1/8$.
Now let's think back at what was defined as the key to this attack: the deviation from $1/2$, or rather the *bias*. Notice that from the probability of $7/8$ (or $1/8$) a bias of $3/8$ can de derived. This is not the highest bias which means that in a real setting shouldn't be it shouldn't be for the last step of the attack. Here, it servers to show the steps for aligning approximations together so that a linear expression could be defined for the whole cipher, although in practice this goes way beyond 3 rounds and also adds up in complexity.

3.3 Last-round attack

What wasn't mentioned so far is that Matsui, in his original paper on linear cryptanalysis actually proposed two ways of performing the attack [7]. Up to this point of the cryptanalysis, both *Algorithm 1* and *Algorithm 2* are alike. Both require $N \approx \mathcal{O}(1/\varepsilon^2)$ know plaintext-ciphertext pairs and the corresponding linear approximation (as described in the previous sections) to perform the attack. In the following we will display the key differences of the two and discuss which one is better suited for a significant speed-up using quantum devices. Algorithm 2 is a so-called *last-round attack*.

3.3.1 Matsui's Algorithm 1

The first attack proposed by Matsui, also denoted as key-recovery attack, will give us a bit of the key according to the sign of ε of the linear approximation. This means that, using a variety of different linear approximations, this attack can be repeatedly performed in order

to recover as many key bits as possible. After that, a process of reversing the cipher by running ciphertexts through the discovered key bits and through the corresponding S-boxes, is executed. This is done for all plaintext-ciphertext pairs while keeping track of a counter, which for each guessed key (assembled by the discovered key bits) represents the times the linear expression holds true for the bits into the last round's S-boxes and the known plaintext bits. The guessed key which has the count which differs the greatest from half the number of plaintext-ciphertext samples is assumed to represent the correct values of the key bits [8]. By assumption that I have n linear approximations (which can be defined by following the previous steps) with minimum bias ε and k recovered key bits, the data (D_C) and time (T_C) complexities are:

$$D_C = 1/\varepsilon^2, \qquad T_C = n/\varepsilon^2 + 2^{k-n}$$

3.3.2 Matsuis's Algorithm 2

Alternatively, linear cryptanalysis can be performed in a so-called *last-round attack*. In this attack, a $r-1$-round linear approximation is used together with a partial decryption of the last round. Latter is realized by guessing the key bits of the last round, which leads to a much more efficient attack in practice [4]. Depending on the literature, this attack is composed by 2 or 3 phases. However, it can be described by 2 main phases:

1. **Distillation phase:**

   - Initialize a set of 2^k counters. (where k is number of bits in the keyguess)
   - For each generated ciphertext, extract the k-bit value corresponding to the active S-boxes and evaluate the parity of the plaintext subset defined by the approximation. Increment or decrement the counter corresponding to the extracted k-bit value according to the parity.

   An important addition to this step is the parallel construction of a table that for each k-bit ciphertext indexes the difference between the frequency of its apparition leading to a null input parity and the frequency leading to a non-null input parity. This will help us evaluate the bias of the approximation for each key during the analysis phase.

2. **Analysis phase:**

   - For each k-bit ciphertext and k-bit subkey, partially decrypt the k-bit ciphertext under the k-bit subkey and evaluate the parity of the output subset (as defined by the linear approximation). Keep this value in a table of size $2^k \cdot 2^k$.
   - For each k-bit subkey, evaluate its bias by checking, for each k-bit ciphertext, the parity of the approximation and the value of the corresponding counter. Then output the subkey which has maximal bias.

   Here, the actual bias for each subkey candidate is evaluated. In order to avoid multiple evaluations of the same operation, a table is constructed which indexes, for each k-bit ciphertext and each subkey candidate, the parity of the output subset obtained after the partial decryption of the ciphertext XORed with the subkey. For a given subkey candidate, its bias can then be evaluated by summing, for each k-bit ciphertext, the corresponding counter, taking the parity of the approximation for the given ciphertext and subkey into account (this parity is given by the sign of the counter and the correct index in the precomputed table) [4].

Notice that in the last step of the analysis phase, the table is accessed the precomputed table is accessed 2^k times for each subkey candidate. This leads us to the following complexity for *algorithm 2*:

$$\mathcal{O}(2^k \cdot 2^k)$$

Because it uses $r-1$ rounds instead of r, algorithm 2 has a massive edge over the first one if data complexity is concerned [4]. The flip side to it is that the increase in efficiency has its counterpart in a higher computational complexity, due to the possible involvement of a large number of key bits in the key guesses. That is why in the evaluation of linear cryptanalysis attacks against modern block ciphers, the attacker usually does a tradeoff between the bias of the approximation and the size of the keyguess. This is why in the next chapter I will propose a framework that exploits the cryptanalytic steps we have done until now and speeds up the attack significantly. This will be realized by showing that a *Fast Fourier Transform* in itself already leads to a significant speed-up in the classical algorithm, and further, that we could implement a big part of that procedure on a quantum computer. This will be shown to gain a huge leap in computational complexity in the attack.

4. Improving the attack

This chapter will be all about optimization. We have seen the basic concept of the linear cryptanalysis attack. Now I want to focus on the improvement of the complexity. In this paper, I will focus in particular on Matsui's *Algorithm 2* for two reasons:

- The data complexity is already shown to be lower than for algorithm 1.
- The upside in complexity improvements provided by the techniques we will propose lie on the side of computational complexity, which is what we can exploit.

Putting these two components together, I think that we can achieve a notable speedup in overall time complexity of this attack.
There are different ways proposed to accelerate linear cryptanalysis, which for instance involve *maximum-likelihood* approaches [9]. As mentioned before, the main advantage for me is the applicability of a FFT, since I can efficiently implement it on a quantum computer. Therefore, those two methods combined is what will be presented in this chapter. Firstly, I explain how the classical algorithm can be improved by the **Fast Fourier Transform**. This improvement will already give me a significant complexity speedup, namely from the before-mentioned $\mathcal{O}(2^k \cdot 2^k)$ to $\mathcal{O}(k \cdot 2^k)$ (where k is the number of bits in the keyguess). After that, we will give a detailed explanation on how to realize the proposed improvements using FFT on a hypothetically large enough quantum computer, which will provide a notable speedup thanks to computation on superposition states.
Also, by realization that with the procedure proposed in this paper, I are dealing with a mix of classical and quantum components, which means that I have to find a way to translate classical information into a quantum wave function. Therefore, I describe a framework that generalizes the preparation of the quantum state needed for the algorithm. More precisely, I will present a quantum circuit that gives us the state $|\psi\rangle$ form a base state $|00...0\rangle$, where $|\psi\rangle$ is a complex vector of 2_n entries (for n $= \#$ qubits), which will be the input state for the FFT (or rather QFFT) algorithm.

4.1 Improvements with the Fast Fourier Transform

The Fast Fourier Transform (FFT), an idea already introduced (but not published) by Carl Friedrich Gauß in 1805, is an algorithm that decreases by several orders of magnitude the number of algorithmic operations required to compute a discrete Fourier transform (DFT). The DFT takes a vector of N complex numbers and transforms it into a different problem, to which the solution is known [13]. This is a very important feature, that in mathematics is used in a variety of problems. The DFT is defined as follows (where $y_0, \ldots, y_{N-1}$ are the entries of the input vector and $y_0, \ldots, y_{N-1}$ those of the output vector):

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i \frac{jk}{N}} \qquad (6)$$

The associated FFT is a widely celebrated algorithmic innovation of the 20th century [6] allows for the computation of a DFT with an entry-vector of size N in $\mathcal{O}(N \log N)$[3]. It derives its efficiency by replacing the computation of one large DFT with that of several smaller DFTs. In the realm of algorithmic terms, the procedure of this algorithm can be characterized by the *divide and conquer* approach. This means that a DFT of length $x = n \cdot m$, solved by the traditional DFT algorithm, requires $n^2 \cdot m^2$ operations, while a FFT applied on the same setting reduces those operations to $n \cdot m \cdot (n + m + 1)$ [14]. In practice, this has lead to the solution of many algorithm otherwise thought to be very complex. As shown in [4], the application of the FFT in the context of linear cryptanalysis and *algorithm 2*, the overall complexity of the attack can be reduced from $\mathcal{O}(2^k \cdot 2^k)$ to $\mathcal{O}(k \cdot 2^k)$ (with $k = \#$ bits in the keyguess). That was shown by picturing the table constructed during the analysis phase as a matrix C with $2^k \cdot 2^k$ entries and by exploiting the circulant nature of this matrix, the FFT can be applied to compute the approximation biases for the subkey candidates. Here, C is a matrix that maps every entry to ± 1 according to the parity of a specific k-bit subset. That is, $C(i, j)$ is the parity of a specific linear approximation after a partial en/decryption of a k-bit text i with k-bit subkey j. They discovered the following correlation: a matrix-vector product $\varepsilon = Cx$ can be written as $\varepsilon = F^* diag(\lambda) Fx$ [4] , where F is a k-dimensional Discrete Fourier Transform matrix, λ is the vector of eigenvalues of C and x is the vector of counters constructed during our distillation phase. This relationship can be exploited as such, that by applying the FFT algorithm to solve the different matrix-vector products in the formula, already on a classical device this reduces the complexity to $k \cdot 2^k$. From this we can deduce that the FFT has to be applied multiple times within a single calculation. Indeed, that is the case. Therefore, later on I will also show that our interaction with the quantum components is everything but redundant and resource-consuming, since due to the nature of the chosen quantum encoding it will be possible to make bits reusable. It is also shown that in the most common linear cryptanalysis attack scenarios, a maximum of $2n + 1$ FFTs will be needed (with n being the number of approximations). The details of the classical FFT algorithm can be found in numerous other resources, and explaining it step by step would exceed the realm of this paper. Here I want to focus on the main advantages to be gained by creating a FFT quantum circuit (from here denoted as QFFT).

Therefore, in the following parts of the paper I will focus on the quantum informational side of the problem statement. Firstly, I will discuss how a hypothetically large quantum computer would realize a quantum circuit for the QFFT. After that, I ask the question of transitioning between classical and quantum data. Therefore, a couple of ideas on how to come about this transition will be proposed, which will enable me to take classical data and prepare it quantum-mechanically for the QFFT circuit.

4.2 Improvements on a quantum computer

In this section I use the previously introduced idea of a classical FFT as a means for the purpose of accelerating the entirety of the attack by substituting a classical component with a quantum-mechanical one. This enables me to utilize heuristics of quantum information processing for the specific task that we want to tackle. The task meant herewith is the implementation of

a quantum version of the FFT (QFFT). This can be seen as a „quantumization“of a portion of the algorithm.

4.2.1 Quantum Fourier Transform

It is important to understand here that, although their names may lead to false correlation, the hereby gradually introduced QFFT circuit doesn't specifically relate to the Quantum Fourier Transform (QFT) circuit. The QFT, as depicted in Figure 3,

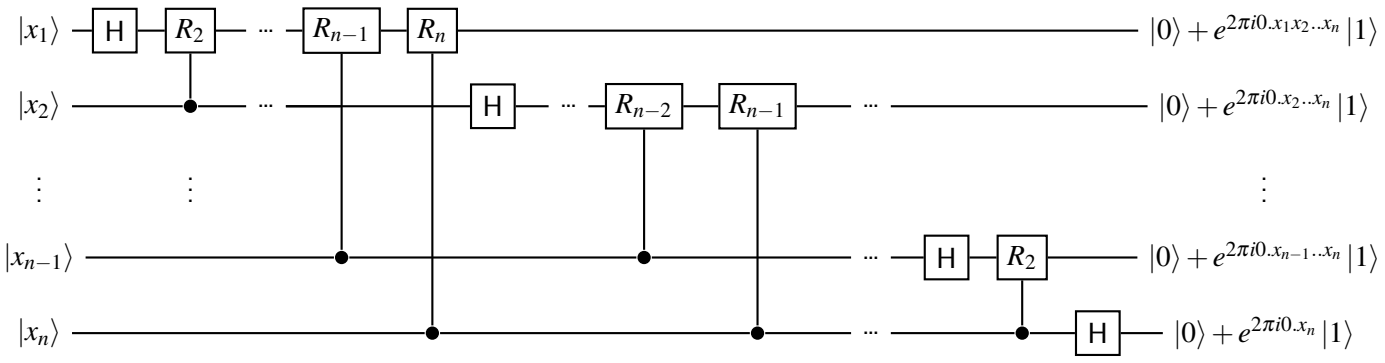


Figure 3. QFT circuit using [19][11].

is the quantum mechanical equivalent to the earlier introduced DFT (6), which performs the following unitary linear operation on the orthonormal basis states $|0\rangle, \ldots, |N-1\rangle$.

$$|j\rangle \longrightarrow \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i \frac{jk}{N}} |k\rangle \qquad (7)$$

which on an arbitrary state can be expressed by the following transformation of the respective amplitudes x_j (and transformed amplitudes y_k)

$$\sum_{j=0}^{N-1} x_j |j\rangle \longrightarrow \sum_{k=0}^{N-1} y_k |k\rangle \qquad (8)$$

As we can see in Figure 3, the only two gates required for the QFT are the H gate and the R_k gate.

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad R_k \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$$

Essentially the QFT consists of a cycle of gate applications, where the Hadamard gate is initially applied to the first qubit, whereafter the $R_2, \ldots, R_N$ are applied sequentially. This procedure is applied to every one of the subsequent qubits.

The QFT is the equivalent of the classical DFT, where, due to the nature of quantum in-

formation processing, the computation can be reduced to $\mathcal{O}((\log N)^2)$. This already shows the massive potential that quantum computation naturally brings to the affair. In fact, the QFT algorithm is widely used in a variety of quantum algorithms that embody quantum advantage. Shor's factorization algorithm and the quantum phase estimation being only two of them. Different work [3][12] has shown that actually a couple of the main properties of the FFT can be found in the QFT. The procedure of matrix decomposition, within the scope of the FFT also referred to as *QR decomposition*, is undisputedly one of the main features of the FFT. This is shown to be used within the QFT as well. Furthermore, it was established that the application of a so-called *approximate QFT* could reach an even more significant speedup to $\mathcal{O}(m \log N)$ by repeatedly applying it $\mathcal{O}(\frac{\log^3 N}{m^3})$ times, where $1 \leq m \leq \log N$ defines the degree of freedom of the approximation. In fact, this approximate version not only turns out to be simpler and faster, but also leads to more precise results in presence of decoherence, which is a plus since due to the decoherence in our quantum systems. We will not go further into this proposition in this paper.
All these characteristics of the QFT discussed until now, make it a from-running candidate for the realization of our goal. The implementation of a QFT within the scope of linear cryptanalysis attack would realize a significant speedup of computation over a classical implementation of a FFT (at already at best-case, without considering the utilized hardware components).

4.2.2 Quantum Fast Fourier Transform

Since we have now seen that there is an equivalent to the DFT algorithm, the following question arises: if there is a speedup in the realm of classical computation owing to the FFT being more efficient than the DFT, is there such an optimization in the quantum world? The answer is yes. The answer lies in the before-mentioned **QFFT (Quantum fast Fourier transform)**. This is a quantum circuit that has been recently developed and proposed by [1] is supposed to be a quantum implementation of the FFT modeling the data sequence as a tensor product of vector spaces. The details of this implementation will be discussed in this section. The main difference between the QFT and the QFFT is that while the first consists of linear transformations of the superposition states' amplitudes, the latter consists of multiple small circuits that model elementary arithmetic operations on a quantum computer, such as a quantum adder, subtractor and others, which are implemented effectively as a quantum circuit. As I will explain, this idea in more detail throughout the paper a couple of advantages will emerge:

- No unnecessary generation of any garbage bits thanks to the ability of reusing resources.
- High versatility: the method is always applicable to data sets that could also be processed classically by the FFT algorithm.
- Quantum parallelism, which enables the method to make simultaneous processing on superposition states.

Although this sounds very promising, it is plausible to ask what benefit in complexity this new

algorithm is able to provide. [1] noticed the following leap in performance by using N images of size $L \cdot L$ pixels as the measure for computational cost.

Algorithm	Complexity
FFT	$\mathcal{O}(NL^2 \log_2 L^2)$
QFFT+RAM	$\mathcal{O}(NL) + \mathcal{O}(L^2 \log_2 L^2)$
QFFT+qRAM	$\mathcal{O}(L^2 \log_2 L^2)$

Table 3 QFFT complexity.

In Table 3, a relatively new concept called qRAM is introduced. We will go into more detail about this concept later in the complexity section. But as the name already says, *qRAM (quantum random access memory)* is an effective method to convert classical data into quantum states. Effectively, it's the way we prepare classical data quantum-mechanically to be able to perform quantum computations with it.

Formalization

As far as the QFFT algorithm in itself is concerned, the main idea is the decomposition of the data into $\log_2 N$ „layers", where each layer consists recursively of $N/2$ so-called butterfly diagrams as depicted in Figure 4. Here, $W_k = \exp(-2\pi i/k)$ and $G_k^{(j,p)}$ is determined by the following recursive relation.

$$\begin{bmatrix} |G_k^{(n-m,p)}\rangle \\ |G_{k+N/2^{m+1}}^{(n-m,p)}\rangle \end{bmatrix} = \begin{bmatrix} |G_k^{(n-m-1,p)} + W_{N/2^m}^k G_k^{(n-m-1,p+2^m)}\rangle \\ |G_k^{(n-m-1,p)} - W_{N/2^m}^k G_k^{(n-m-1,p+2^m)}\rangle \end{bmatrix} \tag{9}$$

In total, $(N \log_2 N)/2$ splits as shown in Figure 4 are preformed, which leads to a complexity reduction from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log_2 N)$.

Implementation

Now that we know how the algorithm works, we will look at the actual circuit implementation of it. For that, I have to start by defining the single elements of the computation. [1] show that our QFFT circuit can be implemented with the following components (gates): quantum adder, subtractor and shift operator. In the following I will show how to realize these as quantum gates, to later enable the construction of the full circuit. It is a prerequisite to mention that

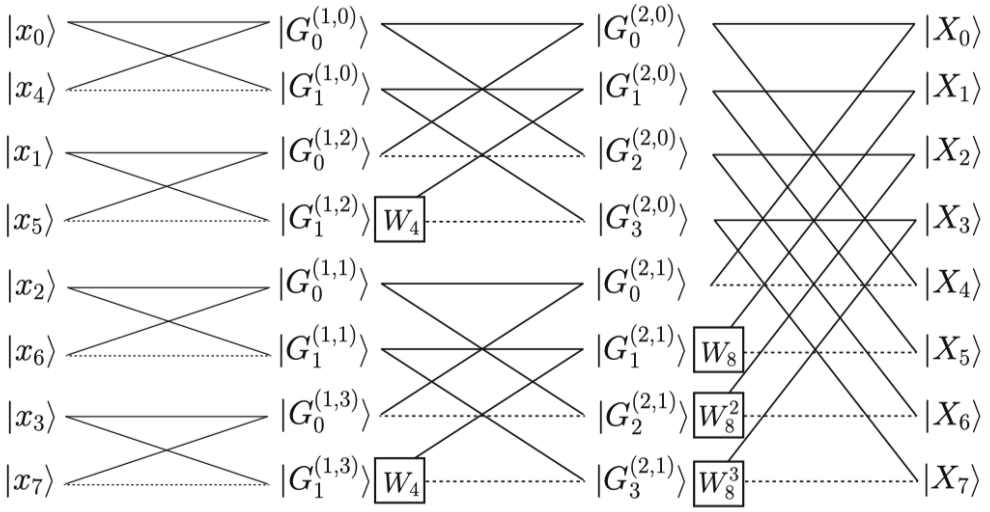


Figure 4. Representation of QFFT for $N = 8$ from [1]

throughout this section a state $|\psi\rangle$ will be represented by its binary representation plus $m-n$ ($m > n$) additional qubits as follows:

$$|\psi_+ \cdots \psi_+ \psi_{n-1} \cdots \psi_0\rangle \qquad for \quad \psi \geq 0 \tag{10}$$

$$|\psi_- \cdots \psi_- \psi_{n-1} \cdots \psi_0\rangle \qquad for \quad \psi < 0 \tag{11}$$

Now, starting from this, I can define the components of the circuit. The three main components consist in an adder, subtractor and an arithmetic shift operation.
A quantum adder is defined as an operation $|a \pm a \pm a_2 \pm a_1 \pm a_0 + b \pm b \pm b_2 \pm b_1 \pm b_0\rangle$ that quantum computationally is carried out as a transformation of the type $|a\rangle \otimes |a+b\rangle$ (accordingly for the subtraction). In the subtractor's case, we have to provide two gates. One for $|-a+b\rangle$ and $|a-b\rangle$ respectively. Each gate is shown in (12), (13), and (14) respectively.

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} |a\rangle \\ |b\rangle \end{bmatrix} = \begin{bmatrix} |a\rangle \\ |a+b\rangle \end{bmatrix} \tag{12}$$

$$\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} |a\rangle \\ |b\rangle \end{bmatrix} = \begin{bmatrix} |a\rangle \\ |-a+b\rangle \end{bmatrix} \tag{13}$$

$$\begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} |a\rangle \\ |b\rangle \end{bmatrix} = \begin{bmatrix} |a\rangle \\ |a-b\rangle \end{bmatrix} \tag{14}$$

Additionally to the adder and subtractor, we have to define an arithmetic shift operation. This operation is essential in the context of an expression $|a\rangle \to |2^p a\rangle$, which can be represented as an operation where digits are shifted like this:

$$|a\rangle = |a_\pm \cdots a_\pm a_{n-1} \cdots a_0\rangle \to |a_\pm \cdots a_\pm a_{n-1} \cdots a_0 0 \cdots 0\rangle = |2^p a\rangle \tag{15}$$

The circuit for this operation can be accomplished by a combination of SWAP and CNOT gates [1].
Now that I have defined basic arithmetic operations in a quantum version, I am able to build arbitrary algorithms. As a consequence, I can describe (by decomposition) the earlier mentioned butterfly circuit, which has input states $|(W_N^k)a\rangle$, as

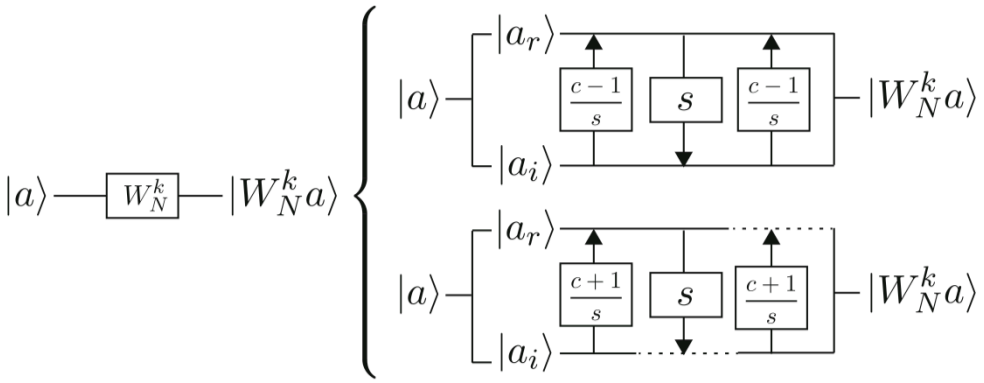


Figure 5. Recursion parameter from [1].

It has been shown that the QFFT circuit consists of $(N\log_2 N)/2$ butterfly operations. Therefore, it has been estimated that that we can build a QFFT circuit with an accuracy of 2^{-A} and at most n_g gates, where

$$n_g = [32n_{in} - 33 + A(45n_{in} - 42)] \cdot \frac{N}{2}\log_2 N \qquad (16)$$

In this chapter I presented two fundamentally different approaches to the implementation of a FFT on a quantum computer. Firstly, I gave an insight into the prominent algorithm in quantum computation, the QFT. This has the advantage of being well known, straightforward to implement and well-established. As an alternative to the QFT I presented the QFFT algorithm [1], which uses quantum implementations of arithmetic operations to create as many butterfly operations as needed to model the FFT on a quantum computer.
Due to the semi-classical nature of our linear cryptanalysis algorithm, in the next chapters I will go over the techniques with which the classical data gathered in the first part can be translated and encoded into quantum states. These quantum states are obviously required for the execution of either the QFT oder the QFFT. After that, I will also talk about the complexity of our approach and its parts.

5. Data preparation and encoding

In this section of the paper I focus mainly on the techniques that enable the transition of data between the classical and the quantum domains. This is important to my use case because I need to have the possibility to encode the data gathered during the analysis phase of linear cryptanalysis into a suitable quantum state. Therefore, the goal is to perform *quantum embedding*, which consists in the representation of classical data as quantum states in a Hilbert vector space [16]. This, in practice, means passing a data-entry x through quantum gates (which will be described later), so that a quantum state $|\psi_x\rangle$.
In principle, for encoding an M-dimensional dataset $D = \{x_1, \cdots, x_m, \cdots, x_M\}$, where each entry consists of an N-dimensional vector x_m (for $m = 1, \cdots, M$) into n qubits, there are different options. I will go over the most important and common ones in the following. Take notice that there exist a variety of other encoding techniques.

5.1 Amplitude Encoding

In this method, the data I want to transform is encoded into the amplitudes of the quantum state itself. Therefore, a quantum system, in the best-case scenario, requires a minimum of $\log_2(NM)$ quantum subsystems to encode $N \cdot M$ amplitudes, since due to its quantum mechanical nature, the system (consisting of n qubits) itself provides 2^n amplitudes. An N-dimensional datapoint x can be encoded into n qubits as follows:

$$|\psi_x\rangle = \sum_{j=0}^{N-1} x_j |j\rangle \qquad (17)$$

where $N = 2^n$ and $|j\rangle$ the j-th computational basis state. One disadvantage of this method is that it requires the dataset to be normalized. This leads us to an encoding of our *normalized*dataset $D_{normalized} = D\alpha_{norm}$ as the following computational basis:

$$|D\rangle = \sum_{j=1}^{2^n} D_{normalized_{(j)}} |j\rangle \qquad (18)$$

5.2 Basis Encoding

In contrast to utilizing the state's amplitudes to embed information, *basis encoding* associates each input with a computational basis state of a qubit system [16]. This already reveals the small advantage that this embedding technique has over the amplitude encoding. Here, the data entries are simply required to be in form of its binary string, which is then mapped to the corresponding states of the qubits. A trivial example for this procedure is the encoding of $x = 12 = 1100$, which is simply represented by the quantum state $|1100\rangle$. It becomes obvious, that this method can get very resource consuming. More precisely, this means that we require $\#qubits \geq N$.

In practice, as far as our dataset is concerned, this means that we can represent D as

$$|D\rangle = \frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} |x_j\rangle \tag{19}$$

with $x_m = (b_1, \cdots, b_N)$, $b_i \in \{0,1\}$.

5.3 Data-encoding quantum circuit

In this section I present a quantum circuit that realizes basis encoding in a very efficient way. *Circuit familiy #3*, as [5] refers to this technique, is able to load N classical bits into $\log_2 N$ qubits through a circuit depth of $\mathcal{O}(\log N)$. The circuit is displayed in Figure 6.

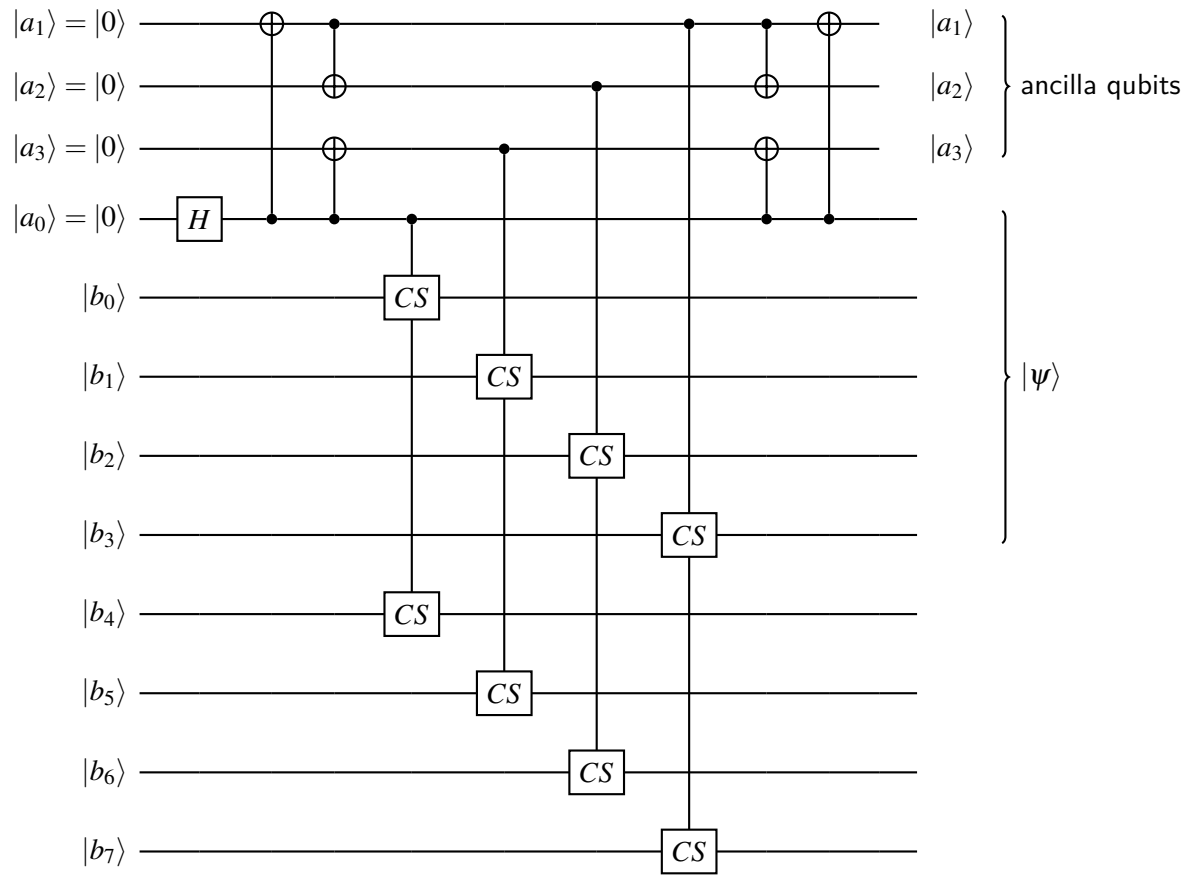


Figure 6. Encoding circuit from [5].

with $|\psi\rangle = |0\rangle \otimes |b_0 b_1 b_2 b_3\rangle + |1\rangle \otimes |b_4 b_5 b_6 b_7\rangle$ and CS-gates = target of the controlled-switch gates. This circuit is composed by a recursion scheme, which consists of n stages (with $N = 2^n$ classical bits), and additional *ancilla qubits*. Ancilla qubits are typically qubits which are only there as a passive helper to the circuit. They are normally discarded after usage. This specific embedding algorithm enables the ancilla qubits to be decoupled from the main qubits after usage. These decoupled quibits are then therefore ready to be assigned to later operation of the circuit. The primary advantage of this optimized circuit is the very efficient deployment

and reuse of resources. Little to no qubits are discarded overall. This leads to a gate depth of $\textbf{Depth} \approx \mathcal{O}(\log N)$.

5.4 FF-qRAM

With the rise of quantum information processing and algorithms that have proven to fulfill quantum-advantage in their specific application, just like in classical information processing, the demand for some type of a *quantum register* has increased. The idea of a qRAM (quantum random access memory) seems to be addressing this problem. qRAM is a promising avenue that stores (either classical of quantum) data, so that it can later be queried and processed [15]. Important to understand here, is that the idea of qRAM is not represented by a specific circuit. Due to the nature and current hardware-development-state of quantum computing, the qubits of the system itself form the qRAM. That is, the gates of the circuit are applied directly to the qRAM itself.
When connecting this to quantum mechanics, it becomes obvious that a qRAM can't possibly satisfy the features of a classical RAM. That is, a quantum state can remain stored in the quantum subsystem, but there is no possibility of *re-accessing* the same information multiple times, since the quantum mechanical features of system disappear after measurement.

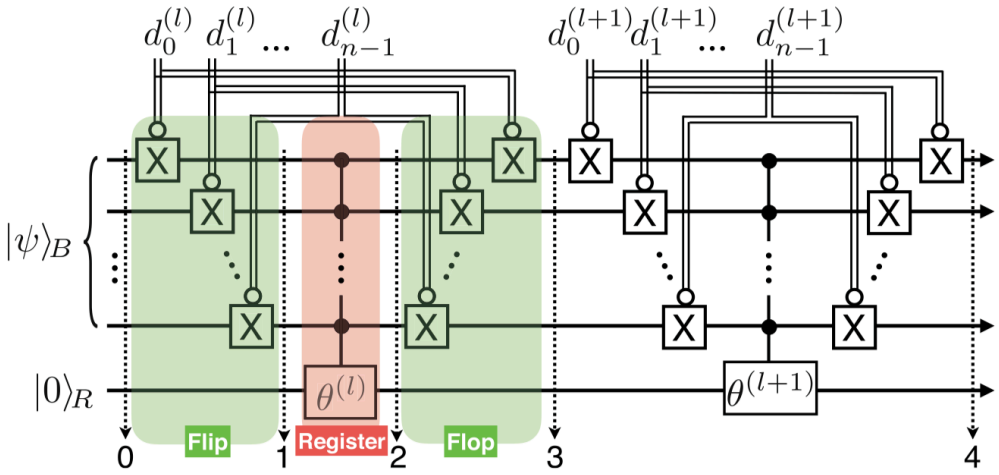


Figure 7. FF-qRAM circuit from [15]

Among others, a method of qRAM called **Flip-flop qRAM** has been developed, which models a quantum database in a quantum system. It utilizes classically-controlled Pauli X gates and n-qubit controlled rotation gates $C^n R_p(\theta)$. The process of registering n-bit classical data in M memory cells into quantum format can be obtained in $\mathcal{O}(\log(Mn))$ flip-register-flop steps [15]. A portion of the implementation is shown in figure 7. It superposes a set of classical data D as

$$qRAM(D)\sum_j \psi_j |j\rangle_B |0\rangle_R \equiv \sum_l \psi_l |\vec{d}\rangle_B |b_l\rangle_R \qquad (20)$$

with B (R) stands for the bus (register) quibt and $\vec{d}$ represents a string of classical bits.

6. Complexity analysis

We have seen that an adequate quantum computer can significantly speedup a portion of linear cryptanalysis attack. But by how much?
Even though I proposed an improvement technique that covers only a slice of the algorithm, the computational complexity gained from that is worth it. As we have seen in section 4.1, the idea of integrating opens the door to a much more optimized time complexity by adopting it into Matsui's *algorithm 2*. This improvement alone already brings the attack's time complexity (the number of partial decryptions) from $\mathcal{O}(2^k \cdot 2^k)$ to $\mathcal{O}(k \cdot 2^k)$, where k is the number of bits in the keyguess. Moreover, [4] show that further improvements can be achieved classically in the scope of *multiple linear cryptanalysis*. This is manifested by the reduction in complexity from 2^{178} to 2^{116} for a matrix C of size $2^{108} \cdot 2^{108}$ (with 108-bit text and 108-bit subkey).
In order to evaluate the advantage of implementing the classical FFT procedure on a quantum system, I take a close look at the number of gates required for our respective quantum circuits, since these correlate highly with the complexity itself. The QFT has cam be implemented by

- n Hadamard gates
- $\lfloor \frac{3n}{2} \rfloor$ CNOT (controlled-NOT) gates
- $\frac{n(n-1)}{2}$ CR (controlled-R) gates

This leads to a total number of $n + \lfloor \frac{3n}{2} \rfloor + \frac{n(n-1)}{2}$ [3] gates and a time complexity for the QFT of $\mathcal{O}(n^2)$, or equivalently $\mathcal{O}((\log N)^2)$. Furthermore, by applying the addressed method of an *Approximate QFT* [12] we can push the boundaries of time complexity to only $\mathcal{O}(m \log N)$, where the parameter m can be expressed as a $\mathcal{O}(\log \log N)$. The alternative processing circuit to the QFT, the QFFT, creates a dedicated circuit for the FFT on a quantum computer. It quickly becomes clear that the number of gates is much higher, due to the highly composite nature of the circuit. It is estimated [1] that it requires at most

$$n_g = [32n_{in} - 33 + A(45n_{in} - 42)] \cdot \frac{N}{2} \log_2 N \qquad (16)$$

gates. This property leads to an overall time complexity of $\mathcal{O}(N \log N)$, where $N = n^2$ is the inputted dataset.
One of my main goals in the paper was also to try to provide a completion of the protocol by highlighting different ideas on how to encode the data from the first part of our attack into the quantum device. Obviously, the time complexity of this process has to be taken into account, and therefore added to the algorithms complexity, to give an extensive overview. In an optimal scenario, I would certainly like to have some version of a qRAM to work with. Therefore, by utilizing qRAM both methods' complexities change to:

- $\mathcal{O}(N) + \mathcal{O}((\log N)^2)$ for the **QFT**

- $\mathcal{O}(N) + \mathcal{O}(N \log N) = \mathcal{O}(N \log N)$ for the **QFFT**

7. Discussion and Conclusion

This paper discusses the implementation of an important part of the linear cryptanalysis algorithm. It presents a full method on of the process. I start by giving a detailed explanation of the underlying attack. The aim of linear cryptanalysis is to find affine linear approximations for a specific cipher. This is not trivial, due to the non-linearity components of the cipher. Moreover, the cipher itself consists of an encryption process that is split in multiple rounds with different subkeys. Therefore, linear approximations are created using probability biases of certain scenarios. Matsui [7], in his original paper on linear cryptanalysis, describes two different approaches to the attack, *algorithm 1* and *algorithm 2*.
The discovery about the FFT algorithm being applicable to the result of the analysis phase during *algorithm 2* sparks the idea of implementing that specific evaluation portion of the algorithm as a quantum circuit to lower the complexity of large matrix evaluations. The table that is created during the analysis phase in the classical part of the algorithm contains for every entry the bias of each subkey candidate. Due to the circulant properties of the matrix constructed from this table the FFT is applicable.
The suggested speedup by implementation on a quantum computer is what is important here. To achieve this quantum information processing I suggest a couple of methods. These methods are not only the quantum processing circuits themselves, but also the information encoding circuits, since a transition from classical information to quantum states is required. On the one hand, I described the well-known QFT algorithm, which finds its usage in many quantum algorithms. On the other hand, I talked about a dedicated circuit for implementing a FTT on a quantum computer using arithmetic operations (in the form of quantum gates). Both provide a huge leap in time complexity. Combined with qRAM, a complexity of $\mathcal{O}(N \log N)$ can be achieved, which is a big difference from the original algorithm's complexity of $\mathcal{O}(2^n \cdot 2^n)$. Since the improvement suggested in this paper is a relatively modular process, it is applicable to a variety of ciphers, for instance SPN and Feistel schemes.
As the scope of future work, I think it would be very interesting to find a quantum circuit that not only processes a portion of the attack, but rather models the complete linear cryptanalysis algorithm on a quantum device.

Bibliography

[1] Ryo Asaka, Kazumitsu Sakai **and** Ryoko Yahagi. Quantum circuit for the fast Fourier transform **in**: *Quantum Information Processing* 19.8 (2020), **pages** 1–20. ISSN: 15731332. DOI: 10.1007/s11128-020-02776-5. arXiv: 1911.03055. URL: https://doi.org/10.1007/s11128-020-02776-5.

[2] Eli Bihara. On matsui's linear cryptanalysis **in**: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 950 (1995), **pages** 341–355. ISSN: 16113349. DOI: 10.1007/bfb0053449.

[3] Daan Camps, Roel Van Beeumen **and** Chao Yang. Quantum Fourier transform revisited **in**: *Numerical Linear Algebra with Applications* 28.1 (2021), **pages** 1–18. ISSN: 10991506. DOI: 10.1002/nla.2331. arXiv: 2003.03011.

[4] B. Collard, F. X. Standaert **and** Jean Jacques Quisquater. Improving the time complexity of matsui's linear cryptanalysis **in**: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4817 LNCS.May 2014 (2007), **pages** 77–88. ISSN: 16113349. DOI: 10.1007/978-3-540-76788-6_7.

[5] John A. Cortese **and** Timothy M. Braje. Loading classical data into a quantum computer **in**: *arXiv* (2018), **pages** 1–38. ISSN: 23318422. arXiv: 1803.01958.

[6] Jack Dongarra **and** Francis Sullivan. Guest Editors Introduction to the top 10 algorithms **in**: *Computing in Science & Engineering* 2 (2000), **pages** 22–23. DOI: 10.1109/MCISE.2000.814652.

[7] Tor Helleseth. *LNCS 0765 - Advances in Cryptology - EUROCRYPT '93 (Index)*. ISBN: 3540576002. URL: https://link.springer.com/content/pdf/10.1007{\%}2F3-540-48285-7.pdf.

[8] Howard M. Heys. A tutorial on linear and differential cryptanalysis **in**: *Cryptologia* 26.3 (2002), **pages** 189–221. ISSN: 15581586. DOI: 10.1080/0161-110291890885.

[9] David Hutchison **and** John C Mitchell. *2004.CRYPTO.pdf*. ISBN: 3540226680.

[10] Marc Kaplan **andothers**. Quantum Differential and Linear Cryptanalysis **in**: (2015), **pages** 1–25. DOI: 10.13154/tosc.v2016.i1.71-94. arXiv: 1510.05836. URL: http://arxiv.org/abs/1510.05836{\%}0Ahttp://dx.doi.org/10.13154/tosc.v2016.i1.71-94.

[11] Alastair Kay. Tutorial on the Quantikz package **in**: *arXiv* (2018). ISSN: 23318422. DOI: 10.17637/rh.7000520. arXiv: 1809.03842.

[12] F. L. Marquezino, R. Portugal **and** F. D. Sasse. Obtaining the Quantum Fourier Transform from the classical FFT with QR decomposition **in**: *Journal of Computational and Applied Mathematics* 235.1 (2010), **pages** 74–81. ISSN: 03770427. DOI: 10.1016/j.cam.2010.05.012. arXiv: 1005.3730. URL: http://dx.doi.org/10.1016/j.cam.2010.05.012.

[13] Michael A. Nielsen, Isaac Chuang **and** Lov K. Grover. *Quantum Computation and Quantum Information*. **volume** 70. 5. 2002, **pages** 558–559. ISBN: 9781107002173. DOI: 10.1119/1.1463744.

[14] H. J. Nussbaumer. *Fast fourier transform and convolution algorithm-*. **volume** 70. 5. 1982, **page** 527. ISBN: 9783662005538. DOI: 10.1109/PROC.1982.12347.

[15] Daniel K. Park, Francesco Petruccione **and** June Koo Kevin Rhee. Circuit-Based Quantum Random Access Memory for Classical Data **in**: *Scientific Reports* 9.1 (2019), **pages** 1–8. ISSN: 20452322. DOI: 10.1038/s41598-019-40439-3. arXiv: 1901.02362.

[16] PennyLane. *Quantum Embedding*. 2019. URL: https://pennylane.ai/qml/glossary/quantum{_}embedding.html.

[17] C. E. Shannon. Communication theory of secrecy systems. 1945. **in**: *M.D. computing : computers in medical practice* 15.1 (1998), **pages** 57–64. ISSN: 07246811.

[18] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring **in**: (2002), **pages** 124–134. DOI: 10.1109/sfcs.1994.365700.

[19] TikZ. *Latex quantum circuits*. URL: https://www.iacr.org/authors/tikz/.