quantum reports

IMPACT FACTOR 1.3

CITESCORE 3.0

Article

HHL Algorithm for Tensor-Decomposable Matrices

Cezary Pilaszewicz and Marian Margraf

MDPI

https://doi.org/10.3390/quantum7030037

quantum reports

MDPI

Article

HHL Algorithm for Tensor-Decomposable Matrices

Cezary Pilaszewicz * and Marian Margraf

Department of Mathematics and Computer Science, Freie Universität Berlin, Takustr. 9, 14195 Berlin, Germany
* Correspondence: cezary.pilaszewicz@fu-berlin.de

Abstract

We use the HHL algorithm to retrieve a quantum state holding the algebraic normal form (ANF) of a Boolean function. Unlike the standard HHL applications, we do not describe the cipher as an exponentially big system of equations. Rather, we perform a set of small matrix inversions which correspond to the Boolean Möbius transform. This creates a superposition holding information about the ANF in the form $|\mathcal{A}_f\rangle = \frac{1}{C}\sum_{I=0}^{2^n-1} c_I |I\rangle$, where c_I is the coefficient of the ANF and C is a scaling factor. The procedure has a time complexity of $\tilde{\mathcal{O}}(n)$ for a Boolean function with n-bit input. We also propose two approaches by which some information about the ANF can be extracted from such a state. Next, we use a similar approach, the Dual Boolean Möbius transform, to compute the preimage under the algebraic transition matrix. We show that such a matrix is well-suited for the HHL algorithm when the attacker gets oracle access in the Q2 setting to the Boolean function.

Keywords: cryptanalysis; quantum computing; algebraic normal form

1. Introduction

The most successful quantum algorithms rely on the quantum computer's ability to efficiently switch between different representations of a function. Then, a measurement in the corresponding basis delivers information about some structural property of the function. The most prominent example, a measurement in the Fourier basis, can extract information about the periodicity of the function and allows breaking schemes like the RSA. Similarly, the algorithm proposed in [1] uses the Fourier transform over the $(\mathbb{Z}/2\mathbb{Z})^n$ to find the parity basis representation [2]. In this paper, we want to follow a similar approach, however, in a different setting. We begin with an easy-to-prepare yet exponentially big quantum state. Next, we perform a quantum basis change, which in a classical setting is costly, and retrieve meaningful information about the function.

The HHL algorithm, proposed in 2009 by [3], is a quantum procedure that computes the preimage of a given input. Unlike the classical approaches, it does not linearly depend on the size of the matrix. Rather, its runtime is defined by the sparseness and condition number of the matrix, and only logarithmically by the matrix size. Since most of the modern ciphers can be described as some form of exponentially big equation system, the idea of using HHL to cryptanalyse ciphers came forward [4] and was quickly followed by other publications. However, the common factor was always the abuse of the logarithmic speed-up in the runtime of the HHL algorithm.

In this paper, we will also tackle an exponentially big input $\overrightarrow{b}$. In contrast to the standard approaches of HHL-based cryptanalysis, we will consider matrices of a special form. We prove that, if the matrix M can be decomposed into a tensor product $M = M_1 \otimes M_2 \otimes \ldots \otimes M_n$, then computing the pre-image can be significantly sped up. On classical

check for updates

Academic Editor: Vladimir M. Stojanović

Received: 14 July 2025
Revised: 6 August 2025
Accepted: 13 August 2025
Published: 19 August 2025

Citation: Pilaszewicz, C.; Margraf, M. HHL Algorithm for Tensor-Decomposable Matrices. *Quantum Rep.* **2025**, *7*, 37. https://doi.org/10.3390/quantum7030037

computers, pre-image computation can be done in polynomial time in regard to the vector's size. Only in the case when the input $\overrightarrow{b}$ can likewise be decomposed into the following tensor product:

$$\overrightarrow{b} = \overrightarrow{b_1} \otimes \overrightarrow{b_2} \otimes \ldots \otimes \overrightarrow{b_n},$$

are we able to obtain a subpolynomial runtime. Here, the sizes of $\overrightarrow{b_i}$'s must match the sizes of corresponding matrices M_i for each $i \in 1, \ldots, n$. If such a decomposition of $\overrightarrow{b}$ is unknown, or even impossible, the fastest known classical approximation algorithm is the conjugate gradient method, with a runtime of $\mathcal{O}\left(Ns\sqrt{\kappa}\log_2(\frac{1}{\epsilon})\right)$. In our approach, we do not need a vector decomposition to perform the HHL algorithm, allowing a runtime advantage over classical computers. We summarize this observation in the following Theorem:

Theorem 1. *Let $M = \bigotimes_{i=1}^{t} M_i$ be a horizontal decomposition of a Hermitian matrix M. Further, for all $i = 1, \ldots, t$, let $M_i \in \mathbb{C}^{N_i \times N_i}$ be a Hermitian matrix with κ_i its condition number and s_i its sparseness. The time complexity of the vertical HHL from Theorem 2 for the matrix M is:*

$$\tilde{\mathcal{O}}\left(t \cdot (\max_i \kappa_i \cdot s_i \cdot poly(\log N_i))\right)$$

Based on the above Theorem, our approach is next applied to a cryptographically significant matrix corresponding to the Boolean Möbius transform. The transform allows transition between the algebraic normal form (ANF) of a Boolean function and its truth table (TT). We decided to decompose this matrix as, in its composed form, it has high sparsity (one full column). Applying the standard HHL algorithm in such a scenario would be counterproductive, as the runtime would be comparable to the gradient approach ($\tilde{\mathcal{O}}(N)$, where N is the number of rows in the matrix. Instead, we decompose the Möbius transform into a tensor product of n small sub-matrices (2×2 before Booleanization) and achieve a runtime of $\tilde{\mathcal{O}}(n)$ with $N = 2^n$.

By capturing the truth table in the superposition and computing its preimage under the Boolean Möbius transform, we manage to encapsulate the algebraic normal form of a Boolean function f under attack in the quantum register. The result is as follows:

$$|\mathcal{A}_f\rangle = \frac{1}{\sqrt{\mathrm{hw}(\mathcal{A}_f)}} \sum_{I=0}^{2^n-1} c_I |I\rangle$$

where c_I is the binary coefficient of the monomial x^I. $|\mathcal{A}_f\rangle$ can be used to both retrieve the ANF of the function, as well as to estimate its Hamming weight. We also use the technique introduced in [5] to extract information about a Boolean function using the *algebraic transition matrices (ATMs)*.

A standard requirement in quantum computation is the unitarity of evolutions applied to the qubits. We employ the HHL algorithm to allow inversion of non-unitary matrices. The HHL algorithm is a well-established framework suitable for this operation. However, in the Boolean Möbius transform setting, the procedure could be adjusted. As the Möbius transform is self-inverse, we do not need to compute the preimage. Instead, computing the image under the transform delivers the same result.

We introduce the notation in Section 1.2. Section 2 focuses on the HHL algorithm. We estimate the runtime and present how to Booleanize a matrix and how the HHL algorithm is used in cryptanalysis. In Section 3, we discuss our new approach to applying HHL to a tensor product. The final section concludes with the proof of Theorem 1. Section 4 shows how to apply the tensor approach to a specific function—the Boolean Möbius transform.

We define the Boolean Möbius transform and its tensor decomposition. We explain why it is better to apply the algorithm to decomposed matrices and how to generate the input quantum state. Finally, we describe the result of our algorithm. In Section 5 we elaborate on the cryptographic use-cases of the Boolean Möbius transform. We show how the transform can be used to analyse the algebraic normal form of a Boolean function.

1.1. State-of-the-Art of HHL in Cryptanalysis

The cryptographic appeal of the HHL algorithm is clear. Most modern ciphers are characterized by complex equation systems needed to describe them. The first attempt was introduced in [4], and used the so-called Macaulay matrix—an exponentially big and sparse matrix which can be solved for the Boolean solution of a polynomial system. The next proposal targeted the NTRU system [6] and used the HHL algorithm to solve a polynomial system with noise [7]. Finally, in [8], the Grain-128 and Grain-128a ciphers were cryptanalysed using techniques based on [4,7]. Ref. [9] suggested improvements upon the above papers, and proved a lower-bound on the condition number of the Macaulay matrix approach. In [10], two systems of equations for AES-128, one over $GF(2)$ and another over $GF(2^8)$ based on the BES construction introduced in [11], were analysed under the HHL algorithm.

1.2. Notation

We will shortly introduce the common notation used throughout this paper. A function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$ is called a Boolean function. It is known that each Boolean function f has a polynomial representation $f(x) \in \mathbb{F}_2[x_1, \ldots, x_n]$ [2]. For an index set $I \subseteq \{1, 2, \ldots, n\}$, we define $x^I := \prod_{i \in I} x_i$. We also define $c_I \in \mathbb{F}_2$ as the coefficient of the term x^I, and will sometimes switch to the notation where the index is the decimal number defined by the binary interpretation of I. For a Boolean function $f(x) = \sum_{I \in [n]} c_I x^I$, the Algebraic Normal Form (ANF) of f is $\mathcal{A}_f := (c_0, \ldots, c_{2^n-1})$. The representation of f in the ANF is unique and defines the function f. Moreover, we can compute the coefficients c_I's as follows:

$$c_I = \sum_{\text{supp}(x) \subseteq I} f(x), \tag{1}$$

where $\text{supp}(x) := \{i : x_i \neq 0\}$ [2]. In other words, we can compute the coefficient c_I by adding (over $\mathbb{F}_2$) the images of all inputs dominated by I (cf. to precursor sets defined in [5]).

The Hamming weight of a Boolean vector $x \in \mathbb{F}_2^n$ is denoted as $\text{hw}(x)$, and it is the number of non-zero entries. The degree of a coefficient can be computed using the Hamming weight:

$$deg(c_I) = \text{hw}(I)$$

The truth table of a Boolean function f is a binary vector $\mathcal{T}_f \in \mathbb{F}_2^{2^n}$ defined as $\mathcal{T}_f = \big(f(0), f(1), \ldots, f(2^n - 1)\big)$.

A vectorial Boolean function is defined as $F : \{0,1\}^n \to \{0,1\}^m$ for $n, m \in \mathbb{N}$. Further, for all $0 \leq i \leq m-1$, $F_i : \{0,1\}^n \to \{0,1\}$ is a Boolean function and the projection of F to the ith component. A projection of a string s to its i'th component will be denoted by $s|_i$.

By $H_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ we denote the Hadamard gate, and by $H_n = \bigotimes_{i=1}^n H_1$ its n-fold tensor product. The E is an identity matrix. In the cases where the size of the identity matrix is not explicitly stated, the index j indicates the size of the matrix $E_j \in \mathbb{C}^{2^j \times 2^j}$.

2. HHL Algorithm

The HHL algorithm, named after the authors Harrow, Hassidim, and Lloyd, is one of the new tools in the quantum toolbox used for cryptanalysis. It was first proposed in [3] as an efficient algorithm to solve exponentially big systems of linear equations, outperforming any classical algorithm. Its cryptographic significance was quickly discovered, and [4] proposed the first attack scenario. Since then, a series of cryptographic publications have considered the HHL setting.

We introduce two problems closely related to the HHL algorithm:

Problem 1. *Given a Hermitian matrix* $M \in \mathbb{C}^{2^n \times 2^n}$ *and an input state* $\overrightarrow{b} \in \mathbb{C}^{2^n}$ *with* $\overrightarrow{b} = (b_0, \ldots, b_{2^n-1})$ *find the state* $\overrightarrow{x}$ *such that:*

$$M \cdot \overrightarrow{x} = \overrightarrow{b}$$

We can also reframe this into the quantum setting as follows:

Problem 2. *Given a Hermitian matrix* $M \in \mathbb{C}^{2^n \times 2^n}$ *and an input state*

$$|b\rangle = \sum_{i=0}^{2^n-1} b_i |i\rangle$$

with $\overrightarrow{b} = (b_0, \ldots, b_{2^n-1}) \in \mathbb{C}^{2^n}$ *find the state* $|x\rangle = \sum_{i=0}^{2^n-1} x_i |i\rangle$ *such that:*

$$M \cdot \overrightarrow{x} = \overrightarrow{b}$$

with $\overrightarrow{x} = (x_0, \ldots, x_{2^n-1}) \in \mathbb{C}^{2^n}$.

The HHL algorithm solves Problem 2. To compute the pre-image of $\overrightarrow{b}$, we need to represent the vector as a quantum register $|b\rangle = \sum_{i=0}^{N-1} b_i |i\rangle$. Then, M is transformed into a unitary operator e^{iMn}, and the operator is applied to $|b\rangle$ using the Hamiltonian simulation technique [12]. This is equivalent to decomposing $|b\rangle$ to an eigenbasis of M and determining the corresponding eigenvalues. The register after this transformation is in the following state:

$$\sum_{j=0}^{N-1} \beta_j |u_j\rangle |\lambda_j\rangle$$

where $|b\rangle = \sum_{j=0}^{N-1} \beta_j |u_j\rangle$, $\{u_j\}_{j=0,\ldots,N-1}$ is the eigenbasis and λ_j's are the corresponding eigenvalues. Using an additional ancilla register, we perform a rotation conditioned on the value of λ_j to obtain the following state:

$$\sum_{j=0}^{N-1} \beta_j |u_j\rangle |\lambda_j\rangle \overbrace{\left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle\right)}^{\text{ancilla register}}$$

We finish with a measurement of the ancilla register. If $|1\rangle$ is observed, the eigenvalues $|\lambda_j\rangle$ of the matrix M are inverted, and the register ends in the following state:

$$C \cdot \sum_{j=0}^{N-1} \beta_j \lambda_j^{-1} |u_j\rangle = M^{-1} |b\rangle = |x\rangle \tag{2}$$

2.1. HHL Runtime

In [3], the authors show how to run the HHL algorithm for a matrix $M \in \mathbb{C}^{N\times N}$ in time $\tilde{\mathcal{O}}\left(\kappa^2 \cdot s^2 \cdot \text{poly}(\log N)\right)$, where s is the sparseness of the matrix M, κ is its condition number, and N is the size. The $\tilde{\mathcal{O}}(\cdot)$ notation suppresses all slowly-growing parameters (logarithmic with respect to s, κ or $\log N$). The algorithm requires that M is either efficiently row-computable, or an access to the oracle which returns the indices of non-zero matrix entries for each row. The phase estimation (PE) step is responsible for a significant portion of the runtime of the HHL. For an s-sparse matrix, the PE takes $\tilde{\mathcal{O}}(\log Ns^2)$ computations. We observe that, after the rotation conditioned on λ_j, we have only a certain probability of observing the ancilla register in a state $|1\rangle$. However, since the constant C from Equation (2) is of magnitude $\mathcal{O}(1/\kappa)$, with $\mathcal{O}(\kappa)$ applications of the amplitude amplification algorithm [13] we can lower the probability of the undesired measurement to a negligible value. Ref. [14] improved the HHL algorithm by lowering the condition number dependence from quadratic to almost linear. Ref. [15] proposed a further improvement to achieve the runtime as follows:

$$\tilde{\mathcal{O}}(\kappa \cdot s \cdot \text{poly}(\log N))$$

This will be the complexity that we will use in this paper. Nevertheless, since the values s, κ of the matrices we propose are constant, the choice of the implementation should not have much influence on the feasibility of the algorithm we develop.

2.2. HHL over Finite Fields

The HHL algorithm in its initial form is defined for matrices and vectors over $\mathbb{C}$. Since most of the cryptographic constructions use finite field arithmetic, the question of whether HHL can be adapted to this setting came forward. In this part, we want to present a set of rules which describe how to transform an equation system M over a finite field $\mathbb{F}_2$ to an equation system over $\mathbb{C}$ as required by the HHL algorithm. Most of the reductions were proposed in [7]. The result will be an equation system over $\mathbb{C}$ which shares the solution with the original system. We will call the resulting system the *Booleanization of M*, and the $\mathbb{C}$-matrix corresponding to the system a *Booleanized matrix*. The total sparseness of a matrix is the number of non-zero entries in all of its rows and columns.

Reduction from Boolean Equations to Equations over $\mathbb{C}$

For a given set of Boolean functions $\mathcal{F} = \{f_1, \ldots, f_m\} \subseteq \mathbb{F}_2[\mathbb{X}]$ with n variables, we want to find a system $\mathcal{F}_\mathbb{C} = \{f'_1, \ldots, f'_{m'}\} \subseteq \mathbb{C}[\mathbb{X}, \mathbb{Z}]$, such that when $(\hat{\mathbb{X}}, \hat{\mathbb{Z}})$ is a solution to $\mathcal{F}_\mathbb{C} = 0$, then $\hat{\mathbb{X}}$ is a solution of $\mathcal{F} = 0$. To do this, we will define a new set of variables $\mathbb{Z} = \{z_1, \ldots, z_m\}$ and $m + n$ new quadratic equations as follows:

$$f_i(x_1, \ldots, x_n) - z_i = 0 \qquad \forall i = 1, \ldots, m \tag{3}$$

$$z_i/2 \in \mathbb{Z} \qquad \forall i = 1, \ldots, m \tag{4}$$

$$x_j - x_j^2 = 0 \qquad \forall j = 1, \ldots, n \tag{5}$$

The first two sets of equations guarantee that the value of $f_i(\hat{\mathbb{X}}) = 0 \mod 2$. This represents the Boolean addition logic in the original system $\mathcal{F}$. The last equation guarantees that the only possible values of $x_1, \ldots, x_n$ are from $\mathbb{F}_2$. An important observation here is also that $\forall i = 1, \ldots, m\ 0 \leq z_i \leq \#f_i$.

Next, we need to present z_i's in a form usable by the quantum computer. Each z_i will be represented in its binary form with $\log(\#f_i)$ bits:

$$z_i = \sum_{b=0}^{\log \#f_i} 2^b z_{i_b}$$

However, since Equation (4) holds, we know $\forall 1 \leq i \leq m : z_{i_0} = 0$. This means we actually only consider the following representation:

$$z_i = \sum_{b=1}^{\log \#f_i} 2^b z_{i_b}$$

We need to incorporate this into the equation system mentioned above in the following way:

$$\begin{aligned} f_i(x_1, \ldots, x_n) - \sum_{b=1}^{\log \#f_i} 2^b z_{i_b} = 0 \quad & \forall i = 1, \ldots, m \\ z_{i_b} - z_{i_b}^2 = 0 \quad & \forall i = 1, \ldots, m, \forall b = 1, \ldots, \log \#f_i \\ x_j - x_j^2 = 0 \quad & \forall j = 1, \ldots, n \end{aligned}$$

Similarly, as in (5), the equation $z_{i_b} - z_{i_b}^2 = 0$ forces the values of $z_{i_b} \in \mathbb{F}_2$. The additional equations mentioned in this section increase the number of needed variables from n to $n + m \cdot \log \#f_i$ and the number of equations from m to $m \cdot (1 + \log \#f_i) + n$. Thus, in the $\tilde{\mathcal{O}}$ notation, the runtime of the HHL remains the same for $n \sim m$.

Finally, the HHL algorithm requires the input matrix to be Hermitian. The resulting Booleanized matrix corresponding to the system $\mathcal{F}_{\mathbb{C}}$ is not. Luckily, we can use the procedure proposed by the original authors of the HHL algorithm to cover this case [3]. The technique uses the property that for an arbitrary matrix M that is not Hermitian, we can construct a new matrix C_M:

$$C_M := \begin{pmatrix} 0 & M \\ M^\dagger & 0 \end{pmatrix},$$

The newly constructed matrix C_M will be Hermitian and a valid HHL input. Here, the $M^\dagger$ is the conjugate transpose of M. We finish this section with a Lemma characterizing the setting and the runtime of the HHL algorithm:

Lemma 1 ([3,15]). *Let $M \in \mathbb{C}^{2^n \times 2^n}$ be a Hermitian matrix and $|b\rangle \in \mathbb{C}^{2^n}$ be a quantum state. We denote the application of the HHL with the matrix M to a state $|b\rangle$ as:*

$$HHL_M(|b\rangle) = |x\rangle$$

The result $|x\rangle$ is a quantum state which solves Problem 2 and the procedure takes $\tilde{\mathcal{O}}\left(\kappa \cdot s \cdot poly(\log(N))\right)$ time.

3. HHL for Tensors of Matrices

Usually, when considering the use-case of HHL, we want to leverage the exponential speed-up in regard to the size of the matrix. This is the natural direction, as that is the obvious runtime difference between the HHL and the classical approach. The hindrance is, however, that, even though the matrix can be exponentially big, it should not have too

many entries. In fact, a matrix with a single full column eliminates the potential use of the HHL (cf. Section 2.1).

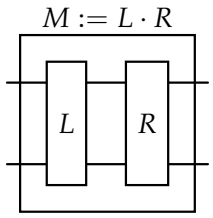
In this section, we present an approach to overcome the above-mentioned obstacles for a special set of matrices. We prove that if for a matrix M we are given a tensor decomposition $M = M_1 \otimes M_2 \otimes \ldots \otimes M_n$, where each M_i is again Hermitian and has a small size, applying the HHL approach to each M_i has a low runtime and delivers the same result. In some cases, our algorithm computes the preimage of a quantum state $|b\rangle$ under the matrix M exponentially faster than the standard HHL approach. A similar technique is a building block of known quantum algorithms, among others, Shor's or Simon's algorithm. For these two algorithms, instead of applying a matrix H_n to an n-long register $|x\rangle$, we apply H_1 to each of n qubits of $|x\rangle$.

We start with a simple Lemma saying that applying a vertical/tensor (cf. Figure 1) decomposition of a unitary matrix $U = \bigotimes_{i=1}^{t} U_i$ to quantum sub-registers results in the same state as applying the matrix U to the whole register:

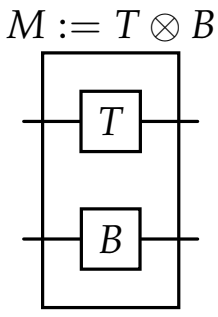
Lemma 2. *Let $U_1, \ldots, U_t$ be unitary matrices with $U_i \in \mathbb{C}^{n_i \times n_i}$. Further, let $|x_1\rangle, \ldots, |x_t\rangle$ be the corresponding quantum states with $|x_i\rangle \in \mathbb{C}^{n_i}$. Then:*

$$\left(\bigotimes_{i=1}^{t} U_i\right) \cdot \left(\bigotimes_{i=1}^{t} |x_i\rangle\right) = \bigotimes_{i=1}^{t} (U_i \cdot |x_i\rangle).$$

This result is unsurprising and is the reason for most quantum algorithms' efficiency. However, we will generalize the argument to show that this equality also holds for non-unitary matrices.



(**a**)



(**b**)

Figure 1. We distinguish between decompositions of a matrix which are well-parallelizable (**b**) and the sequential ones (**a**). To make a clear distinction, we call them vertical and horizontal (regarding how they are stacked in the standard quantum gate-based computation model). (**a**) Horizontal decomposition. (**b**) Vertical decomposition.

Lemma 3. *Let $M_1, \ldots, M_t$ be matrices and $E_1, \ldots, E_t$ be identity matrices with $M_i, E_i \in \mathbb{C}^{n_i \times n_i}$ and $M = \bigotimes_{i=1}^{t} M_i$ and $|x\rangle \in \mathbb{C}^{\prod_{i=0}^{t} n_i}$. Then:*

$$M \cdot |x\rangle = \prod_{j=1}^{t} \left(\bigotimes_{i=1}^{j-1} E_i \otimes M_j \otimes \bigotimes_{i=j+1}^{t} E_i \right) \cdot |x\rangle$$

Proof. Since both the product and the tensor of two matrices are matrices again, it suffices to show this property for two matrices. The rest follows from the associativity of matrix products. Let $M_1, E_1 \in \mathbb{C}^{n_1 \times n_1}$, $M_2, E_2 \in \mathbb{C}^{n_2 \times n_2}$. Further, let $M = M_1 \otimes M_2$. Then, by standard tensor properties, we know the following:

$$\begin{aligned} (M_1 \otimes E_2) \cdot (E_1 \otimes M_2) &= (M_1 \cdot E_1) \otimes (M_2 \cdot E_2) \\ &= M_1 \otimes M_2 \\ &= M \end{aligned}$$

Since $(M_1 \otimes E_2) \cdot (E_1 \otimes M_2)$ is indifferent to M, it does not make a difference which one we apply to a vector $|x\rangle$. □

Again, this is a simple fact from linear algebra, but it will constitute a major stepping stone for our algorithm. We want to highlight that the vector $M \cdot |x\rangle$ from Lemma 3 does not have to be a valid quantum state even if $|x\rangle$ is. Since the matrix M is not unitary, it means the result does not have to be normed.

We will now use the observation from Lemma 3 to build a procedure to solve Problem 2 for a matrix $M \in \mathbb{C}^{2^n \times 2^n}$ with the property $M = \bigotimes_{i=1}^{t} M_i$. Instead of applying HHL to the whole matrix M, we will consider its vertical decomposition and apply HHL to the sub-matrices. We highlight that finding such a decomposition is an NP-hard problem [16], but we assume it is given to us. The idea is depicted in Figure 2. The resulting state will be identical to that of HHL applied to the matrix M.

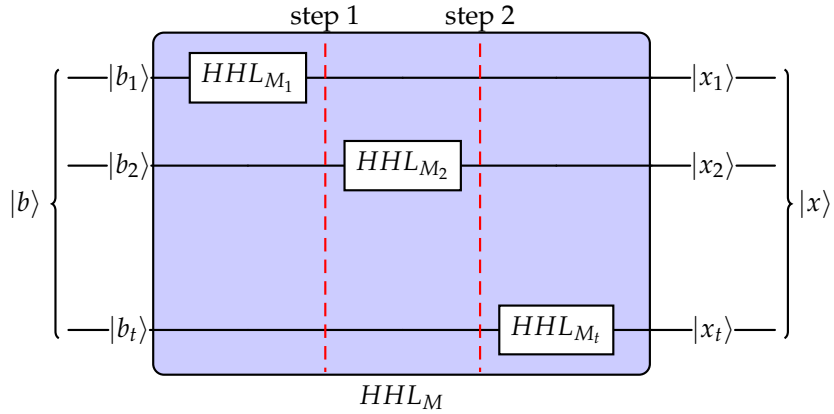


Figure 2. Vertical HHL scheme. We note that we actually apply a single HHL_{M_i} at a time in t steps. The register after step 1 is $(HHL_{M_1} \otimes \bigotimes_{i=2}^{t} E_i)\,|b\rangle$. The state in step 2 is $(E_1 \otimes HHL_{M_2} \otimes \bigotimes_{i=3}^{t} E_i) \circ (M_1 \otimes \bigotimes_{i=2}^{t} E_i)^{-1}\,|b\rangle$.

We begin with a Lemma showing that, for a simple-structured tensor product (a tensor product of mostly diagonal matrices), applying the HHL algorithm on the combined state or the horizontal decomposition delivers the same result and has almost identical runtime.

Lemma 4. *Let $M \in \mathbb{C}^{N\times N}$ be a Hermitian matrix and HHL_M be the HHL algorithm from Lemma 1. Further, let M have the following structure:*

$$M = \bigotimes_{i=1}^{j-1} E_i \otimes M_j \otimes \bigotimes_{i=j+1}^{t} E_i$$

for a Hermitian M_j. Here, E_i's are identity matrices of appropriate dimension. Then, for a given input $|b\rangle$, the following equality holds:

$$HHL_M |b\rangle = \bigotimes_{i=1}^{j-1} E_i \otimes HHL_{M_j} \otimes \bigotimes_{i=j+1}^{t} E_i |b\rangle$$

Proof. For a matrix $M = \bigotimes_{i=1}^{j-1} E_i \otimes M_j \otimes \bigotimes_{i=j+1}^{t} E_i$, the HHL algorithm first uses the Hamiltonian simulation technique to apply e^{iMn} to the register $|b\rangle$. Observe that we can rewrite M as follows:

$$\begin{aligned} M &= \bigotimes_{i=1}^{j-1} E_i \otimes M_j \otimes \bigotimes_{i=j+1}^{t} E_i \\ &= E_1 \otimes M_j \otimes E_2 \end{aligned}$$

for $E_1 = \bigotimes_{i=1}^{j-1} E_i$ and $E_2 = \bigotimes_{i=j+1}^{t} E_i$. Then, the following equality holds:

$$\begin{aligned} e^M &= e^{E_1 \otimes M_j \otimes E_2} \\ &= \sum_{j=0}^{\infty} \frac{(E_1 \otimes M_j \otimes E_2)^j}{j!} \\ &= (E_1 \otimes M_j \otimes E_2)^0 + (E_1 \otimes M_j \otimes E_2) + \frac{1}{2}(E_1 \otimes M_j \otimes E_2)^2 + \dots \\ &= (E_1^0 \otimes M_j^0 \otimes E_2^0) + (E_1 \otimes M_j \otimes E_2) + \frac{1}{2}(E_1^2 \otimes M_j^2 \otimes E_2^2) + \dots \\ &= (E_1 \otimes M_j^0 \otimes E_2) + (E_1 \otimes M_j \otimes E_2) + (E_1 \otimes \frac{1}{2} M_j^2 \otimes E_2) + \dots \\ &= E_1 \otimes \left(M_j^0 + M_j + \frac{1}{2} M_j^2 + \dots \right) \otimes E_2 \\ &= E_1 \otimes e^{M_j} \otimes E_2 \end{aligned}$$

As the remaining steps of the HHL algorithm do not depend on the matrix M, the claim follows. □

Because of the simple structure of M from Lemma 4, the runtime of HHL_M will be $\tilde{\mathcal{O}}\left(\kappa_{M_j} \cdot s_{M_j} \cdot poly(\log N)\right)$. This accounts for an overhead of $\tilde{\mathcal{O}}(\log N)$ when compared to HHL_{M_j} (under the assumption that E_i's and M_j are of relatively similar size).

Theorem 2. *Let $M = \bigotimes_{i=1}^{t} M_i$ be a vertical decomposition of a Hermitian M. Given M_i's are Hermitian matrices as well, let HHL_M and HHL_{M_i} be the unitary HHL algorithm circuit for a matrix M and M_i, respectively. Then for all quantum states $|b\rangle$:*

$$HHL_M |b\rangle = \prod_{j=1}^{t} \left(\bigotimes_{i=1}^{j-1} E_i \otimes HHL_{M_j} \otimes \bigotimes_{i=j+1}^{t} E_i \right) |b\rangle$$

and the result is a valid quantum state $|x\rangle$ from Problem 2.

Proof. Let HHL_M and HHL_{M_i} be the unitary circuit which implements the application of the HHL algorithm with input matrix M and M_i, respectively. Then, by the Lemmas 3 and 4, the following constructions are equal:

$$\begin{aligned} HHL_M &= \prod_{j=1}^{t} \left(HHL_{(\bigotimes_{i=1}^{j-1} E_i \otimes M_j \otimes \bigotimes_{i=j+1}^{t} E_i)} \right) \\ &= \prod_{j=1}^{t} \left(\bigotimes_{i=1}^{j-1} E_i \otimes HHL_{M_j} \otimes \bigotimes_{i=j+1}^{t} E_i \right) \end{aligned}$$

Further, since $\prod_{j=1}^{t} \left(\bigotimes_{i=1}^{j-1} E_i \otimes HHL_{M_j} \otimes \bigotimes_{i=j+1}^{t} E_i \right)$ is a valid quantum circuit, the output of the algorithm is a proper quantum state. Finally, as described in Section 2, the output of the algorithm is a vector $|x\rangle$ such that the following is true:

$$A \cdot \overrightarrow{x} = \overrightarrow{b}$$

This is exactly the solution to Problem 2. □

As the vertical construction is well-parallelizable, the runtime of the vertical HHL will be upper bound by the complexity of the most time-intensive inversion.

Theorem 3. *Let $M = \bigotimes_{i=1}^{t} M_i$ be a horizontal decomposition of a Hermitian matrix M. Further, for all $i = 1, \ldots, t$, let $M_i \in \mathbb{C}^{N_i \times N_i}$ be a Hermitian matrix with κ_i its condition number and s_i its sparseness. The time complexity of the vertical HHL from Theorem 2 for the matrix M is:*

$$\tilde{\mathcal{O}} \left(t \cdot (\max_i \kappa_i \cdot s_i \cdot poly(\log N_i)) \right)$$

Proof. The time complexity of applying each HHL_{M_i} on a subregister is upper-bound by applying the most cost-intensive one. As presented in Section 2.1, this corresponds to a time complexity as follows:

$$\tilde{\mathcal{O}} \left(\max_i \kappa_i \cdot s_i \cdot \text{poly}(\log N_i) \right) \tag{6}$$

Further, applying E_i to any subregister is equivalent to not evolving the register in any way. Therefore in each step, we apply HHL_{M_i} to one of the subregisters and leave the other registers constant. Since we need to invert t sub-matrices, we have t steps, each upper-bound by Equation (6). This results in the runtime as follows:

$$\tilde{\mathcal{O}} \left(t \cdot (\max_i \kappa_i \cdot s_i \cdot \text{poly}(\log N_i)) \right)$$

In our analysis, we implicitly hide the fact that, as mentioned in Section 2.1, in the HHL algorithm we need to perform a measurement of the ancilla register which, with a certain probability, might fail. However, using the amplitude amplification we can boost the probability of measuring the correct state to be almost 1 in $\mathcal{O}(\kappa_i)$ steps for each HHL-subroutine. In [13], the authors describe a method how to slow the rotation down, when the amplitude of the correct state is big enough. In fact, for each subroutine, with $\mathcal{O}(\max_i \kappa_i)$ steps we can obtain a probability of measuring the incorrect state upper-bound by $\frac{1}{2t}$.

Then, for the state $|x\rangle$, which is the result of the vertical HHL algorithm, the probability of measuring at least a single $|0\rangle$ can be lower-bound by the following:

$$\begin{aligned}\Pr\left(\frac{M|x\rangle}{\langle x|M^\dagger M|x\rangle} \neq |1\rangle^{\otimes t}\right) &\leq \sum_{i=1}^{t} \Pr\left(\frac{M|x\rangle}{\langle x|M^\dagger M|x\rangle} \neq \left(\bigotimes_{j=1}^{i-1}|1\rangle\right)|0\rangle\left(\bigotimes_{j=i+1}^{t}|1\rangle\right)\right)\\ &\leq \frac{1}{2t} + \ldots + \frac{1}{2t}\\ &= \frac{1}{2}\end{aligned}$$

Therefore, with at least $\frac{1}{2}$ probability the vertical HHL algorithm will succeed. It should be noted that the ancilla registers indicate when one of the matrix inversions has failed. □

Finally, we want to highlight that, in our considerations, all the subroutines HHL_{M_j} are not executed at the exact same time. Instead, they are sequentially applied to each of the registers. We believe that simultaneously applying all the routines would not drastically change the result and could lead to a speed-up factor of $\tilde{\mathcal{O}}(t)$ compared to Theorem 3. We keep it as future research to formally prove the claim.

4. HHL for Möbius Transform

In this section, we introduce the Boolean Möbius transform, a building block of many cryptanalytical tools. Further, we present how the method from Section 3 can be used for the Möbius transform.

4.1. Möbius Transform of Boolean Functions

The Boolean Möbius transform is the mapping $\mu : \mathbb{F}_2^{2^n} \to \mathbb{F}_2^{2^n}$. It acts as the bijective transform between the ANF of a Boolean function f, and its truth table $\mathcal{T}_f$. It is defined in the following way:

$$\forall x \in \mathbb{F}_2^n : f(x) = \bigoplus_{I \in \mathbb{F}_2^n} \mu(f)(I)x^I,$$

where $\mu(f)$ is the Boolean function defined through the coefficients [17] (cf. Equation (1)). For a Boolean function f, we can compute its Möbius transform in terms of matrices.

Claim 1. *Let T_n be a Boolean matrix recursively defined as:*

1. $T_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$
2. $T_n = \begin{pmatrix} T_{n-1} & O_{n-1} \\ T_{n-1} & T_{n-1} \end{pmatrix}$

where O_{n-1} is a $2^{n-1} \times 2^{n-1}$ 0-matrix.

For a Boolean function f, T_n can serve as the Möbius transform acting on the truth table $\mathcal{T}_f$ and the corresponding ANF as follows:

- $\mu(\mathcal{A}_f) = \mathcal{T}_f$
- $\mu(\mathcal{T}_f) = \mathcal{A}_f$.

A direct consequence is that T_n is self-inverse.

The above claim is thoroughly discussed in [18,19]. As a consequence, we can describe the Möbius transform as a tensor product of polynomial terms:

Lemma 5. *We can describe T_n using the tensor product as:*

$$T_n = T_1 \otimes T_{n-1}$$

Proof. Using the definition of tensor products for matrices, we have the following:

$$\begin{aligned} T_1 \otimes T_{n-1} &= \begin{pmatrix} 1 \cdot T_{n-1} & 0 \cdot T_{n-1} \\ 1 \cdot T_{n-1} & 1 \cdot T_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} T_{n-1} & O_{n-1} \\ T_{n-1} & T_{n-1} \end{pmatrix} \\ &= T_n \end{aligned}$$

□

Lemma 6. *Let $\mathcal{T}_f \in \mathbb{F}_2^{2^n}$ be the truth table of a Boolean function f. Then we can compute the algebraic normal form of f as:*

$$\mathcal{A}_F = T_n \mathcal{T}_f = \prod_{j=1}^{t} \left(\bigotimes_{i=1}^{j-1} E_1 \otimes T_1 \otimes \bigotimes_{i=j+1}^{t} E_1 \right) \mathcal{T}_f$$

Proof. By Claim 1 and Lemma 5, we know T_n can be described as follows:

$$T_n = \bigotimes_{i=1}^{n} T_1$$

Further, by Lemma 3, we know that the constructions are equivalent:

$$\prod_{j=1}^{t} \left(\bigotimes_{i=1}^{j-1} E_1 \otimes T_1 \otimes \bigotimes_{i=j+1}^{t} E_1 \right) \mathcal{T}_f = \left(\bigotimes_{i=1}^{n} T_1 \right) \mathcal{T}_f = T_n \mathcal{T}_f$$

Finally, by using Claim 1, we know that $T_n \mathcal{T}_f = \mathcal{A}_f$ proving the statement. □

Example 1. *Let $f(X_1, X_2) = X_1 \oplus X_1 X_2$. Then, the truth table of f is $\mathcal{T}_f = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$. We will compute the ANF of f using the procedure mentioned in Lemma 6:*

$$\begin{aligned} T_2 \mathcal{T}_f &= (T_1 \otimes E_1) \cdot (E_1 \otimes T_1) \mathcal{T}_f \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \end{aligned}$$

$$= \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$
$$= \mathcal{A}_f$$

We highlight that the formula for the ANF of f from Lemma 6 resembles the formula mentioned in Theorem 2. We will next show that by using the HHL algorithm we can mimic the application of T_n and the result will be a quantum state which represents the $\mathcal{A}_f$.

4.2. Möbius Transform Using HHL

In this section, we show how to tackle a specific initialization of Problem 2. We will compute the preimage of the matrix T_n describing the Möbius transform.

Problem 3. *Given a matrix $T_n \in \mathbb{F}_2^{2^n \times 2^n}$ defined in Claim 1, and an input state*

$$|b\rangle = \sum_{i=0}^{2^n-1} b_i |i\rangle$$

with $\overrightarrow{b} = (b_0, \ldots, b_{2^n-1}) \in \mathbb{C}^{2^n}$, find the state $|x\rangle = \sum_{i=0}^{2^n-1} x_i |i\rangle$ such that:

$$T_n \cdot \overrightarrow{x} = \overrightarrow{b}$$

As we have seen in Section 2, inverting the matrix T_n would solve Problem 3. However, we cannot simply apply the HHL algorithm for matrix T_n to an arbitrary register $|b\rangle$. First, observe that T_n is defined as an $\mathbb{F}_2$-matrix, while HHL operates over $\mathbb{C}$. Instead, we need to use the reduction mentioned in Section 2.2 to create a different matrix Γ_n, which delivers the same result as T_n over $\mathbb{F}_2$. We need additional variables and equations to force the solution to be an $\mathbb{F}_2$-vector and ensure that the addition modulo 2 is correct. It must be noted that this procedure will expand the size of the matrix only polynomially.

Second, the HHL algorithm requires the matrix to be inverted to be Hermitian. The resulting matrix Γ_n is not. We can use the procedure already mentioned in Section 2 by [3]. The main idea is that, for an arbitrary matrix M that is not Hermitian, we can construct a new matrix C_M:

$$C_M := \begin{pmatrix} 0 & M \\ M^\dagger & 0 \end{pmatrix},$$

and the matrix C_M will be Hermitian. Since, for our target matrix Γ_n, all the entries are real, it is clear that $\Gamma_n^\dagger = \Gamma_n^T$. No conjugation needs to be calculated to provide oracle access to $\Gamma_n^\dagger$. Instead, we now also need the column-oracle of the matrix (cf. Section 2.1). The reason for this is the observation that the bottom rows of $C = \begin{pmatrix} 0 & \Gamma_n \\ \Gamma_n^\dagger & 0 \end{pmatrix}$ are in fact the rows of $\Gamma_n^\dagger = \Gamma_n^T$, which are actually the columns of Γ_n. For matrices of exponentially big sparsity, this is an additional requirement.

Further, the input vector must be adjusted. While, for Γ_n, the input was a vector of the form $\overrightarrow{b}$, for the new matrix C we need the vector $\tilde{b} := \begin{pmatrix} \overrightarrow{b} \\ 0 \end{pmatrix}$. Also, the output will have a different form, $\tilde{x} := \begin{pmatrix} 0 \\ \overrightarrow{x} \end{pmatrix}$, due to the following:

$$\begin{aligned} C \cdot \tilde{x} &= \begin{pmatrix} 0 & \Gamma_n \\ \Gamma_n^\dagger & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ \overrightarrow{x} \end{pmatrix} \\ &= \begin{pmatrix} \Gamma_n \cdot \overrightarrow{x} \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} \overrightarrow{b} \\ 0 \end{pmatrix} \\ &= \tilde{b} \end{aligned} \tag{7}$$

There is one problem when we try to apply this approach to the matrix T_n. The sparsity plays a crucial role in the runtime of the HHL. As the Boolean Möbius transform always has one full column, the sparseness of T_n is exponential:

$$s(T_n) = 2^n.$$

We will overcome the runtime implications of the high sparsity of T_n as input to HHL using the decomposition technique.

4.3. Booleanized T_n vs. Booleanized T_1

To overcome the exponentially big runtime of HHL_{T_n}, we will instead call the HHL with the Boolean Möbius transform for each single qubit. Again, we want to highlight that the HHL algorithm requires a Hermitian matrix, which neither T_1 nor T_n are. In the construction above, we Booleanized the matrix T_n to guarantee that the addition is performed modulo 2 and the solution is an $\mathbb{F}_2$-vector. However, it is not guaranteed that the matrix Γ_n can be decomposed into a tensor of small matrices.

Instead, for the vertical HHL, we will first decompose the matrix T_n, and then perform the Booleanization on each sub-matrix T_1. Refs. [7,8] mentioned a set of additional or alternative equations, which when incorporated into the equation system, allow $\mathbb{F}_2$-computation. In the example below, we present one candidate for such a matrix.

Example 2. *Using the techniques mentioned in Section 2.2, we contruct a matrix $\Gamma_1' \in \mathbb{C}^{5\times 5}$. For $a, b \in \mathbb{F}_2$, the preimage under Γ_1' of the vector $\overrightarrow{b} = \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ has the form:*

$$\overrightarrow{x} = \begin{pmatrix} T_1^{-1}\begin{pmatrix} b_0 \\ b_1 \end{pmatrix}|_0 \\ T_1^{-1}\begin{pmatrix} b_0 \\ b_1 \end{pmatrix}|_1 \\ * \\ * \\ * \end{pmatrix} = \begin{pmatrix} T_1^{-1}\begin{pmatrix} b_0 \\ b_1 \end{pmatrix} \\ * \\ * \\ * \end{pmatrix}$$

We define Γ_1' *as:*

$$\Gamma_1' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -2 & -2 & 2 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & -1 \end{pmatrix} \tag{8}$$

such, that:

$$\Gamma_1' \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \Gamma_1' \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \Gamma_1' \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \Gamma_1' \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The construction uses a handful of extra qubits for each HHL application, however, the number of needed qubits will stay polynomial in terms of the size of the register holding the state $|b\rangle$. The matrix must be Hermitianized before being plugged into the HHL (see Equation (7)). The result will be a matrix Γ_1. We want to highlight that the sparsity in Example 2 does not depend on the number n:

$$s(\Gamma_1) = 5$$

and the condition number κ_{Γ_1} is constant.

With the machinery in place, we can solve Problem 3 using the algorithm from Theorem 2. Further, we can estimate the runtime of the algorithm:

Theorem 4. *Let* $T_n = \bigotimes_{i=1}^{n} T_1$ *be a horizontal decomposition of* T_n*. Further, let* $\Gamma_1 \in \mathbb{C}^{\mathcal{O}(1)\times\mathcal{O}(1)}$ *be the Booleanized version of* T_1*. Then, using the vertical HHL algorithm with input* Γ_1*, for an arbitrary quantum state* $|b\rangle$*, we can solve Problem 3 in* $\tilde{\mathcal{O}}(n)$ *time.*

Proof. As seen in Theorem 1, for a matrix decomposition $M = \bigotimes_{i=1}^{t} M_i$, the runtime is as follows:

$$\tilde{\mathcal{O}}\left(t \cdot (\max_i \kappa_i \cdot s_i \cdot \text{poly}(\log N_i))\right)$$

In our case, all M_i's correspond to the Booleanized version of the T_1 matrix. We can use the techniques mentioned in Section 2.2 to create a Booleanized Γ_1. All the reduction steps keep the sparsity, the condition number, and the size of the resulting matrix polynomial in terms of the size of T_1 [7]. Let $s_{\Gamma_1}, \kappa_{\Gamma_1}, N_{\Gamma_1}$ be the sparsity, condition number, and size of the matrix Γ_1, respectively. Plugging the values into the upper-bound, we obtain the collected runtime for the algorithm:

$$\tilde{\mathcal{O}}\left(t \cdot (\max_i \kappa_i \cdot s_i \cdot \text{poly}(\log N_i))\right) = \tilde{\mathcal{O}}\left(n \cdot (\kappa_{\Gamma_1} \cdot s_{\Gamma_1} \cdot \log(N_{\Gamma_1}))\right) = \tilde{\mathcal{O}}\left(n\right)$$

Keeping in mind that Problem 3 is an initialization of Problem 2, as well as Theorem 4 is an initialization of Theorem 2, the claim follows. □

4.4. How to Generate Input

We propose two approaches to generate the input $|b\rangle$ as described in Problem 3. They will consider two different settings and target different encryption scheme types. We first introduce the Q1 model, the more realistic yet less powerful one. In Q1 model, the attacker has access to a quantum computer and some classical oracle. This is the standard model used for Shor's or Grover's algorithm. In the Q2 model, we assume a quantum oracle to which the attacker has access. A quantum oracle allows superposition queries. While this attack scenario is based on a very strong assumption, it might still lead to interesting insights into ciphers structure [20,21].

Our Q1 attack will target plaintext-independent constructions, like classical stream ciphers or block ciphers in counter-mode with fixed IVs. Since for a fixed cipher f and a fixed IV value we can build a deterministic classical circuit, we are also able to build its unitary version U_f [22]. Using U_f and an equally distributed superposition of all states, we can prepare the following state in polynomial time:

$$|x\rangle |y\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle |f(i)\rangle .$$

Now, when we measure a $|1\rangle$ in the second register, only values i for which $f(i) = 1$ will remain with a non-zero amplitude in the superposition:

$$|x\rangle |y\rangle = \frac{1}{\sqrt{\mathrm{hw}(\mathcal{T}_f)}} \sum_{\substack{i=0 \\ f(i)=1}}^{2^n-1} |i\rangle |1\rangle .$$

The register $|x\rangle$ has now exactly the desired form of $|b\rangle$ from Problem 3, and each b_i corresponds to the bit output for a given key i. Obviously, the $|1\rangle$ measurement only happens with a certain probability. However, as in the cryptographic setting, we mostly work with functions that are almost balanced [2], and we expect the probability to measure the desired state to be fairly high. For a Boolean function f, the said probability is the ratio of the Hamming weight of $\mathcal{T}_f$ and the size of its truth table, as it is for a balanced function $\frac{1}{2}$. We show the exact procedure in Algorithm 1.

Algorithm 1: Q1 model input preparation for Problem 3

Input: Quantum implementation U_f of a keyed cipher $f : \{0,1\}^n \to \{0,1\}$

1 $|x\rangle |y\rangle \leftarrow |0\rangle^{\otimes n} |0\rangle$;
2 $|x\rangle |y\rangle \leftarrow H_n |x\rangle |y\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle |0\rangle$;
3 $|x\rangle |y\rangle \leftarrow U_f |x\rangle |y\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle |f(i)\rangle$;
4 Measure register $|y\rangle$;
5 If $|y\rangle == |0\rangle$ go to step 1;
6 **return** $|x\rangle$;

Output: $|b\rangle = \sum_{i=0}^{2^n-1} b_i |i\rangle$, where b_i is the value of $f(i)$ scaled by the factor $\frac{1}{\sqrt{\mathrm{hw}(\mathcal{T}_f)}}$ to produce a valid quantum state.

In the Q2 model, we can perform a similar trick to retrieve the superposition of all outputs of a function with a fixed key. The superposition, however, encapsulates the possible inputs to an encryption function with a secret key. Especially in the case where there are weak keys, this might allow retrieving some additional information about the key and some encrypted plaintext. The Q2 attack resembles the previous one and is presented in Algorithm 2.

Algorithm 2: Q2 model input preparation for Problem 3

Input: Quantum oracle $\mathcal{O}_{f_k}$ for a cipher $f_k : \{0,1\}^n \to \{0,1\}$

1 $|x\rangle |y\rangle \leftarrow |0\rangle^{\otimes n} |0\rangle$;

2 $|x\rangle |y\rangle \leftarrow H_n |x\rangle |y\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle |0\rangle$;

3 $|x\rangle |y\rangle \leftarrow U_{f_k} |x\rangle |y\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle |f_k(i)\rangle$;

4 Measure register $|y\rangle$;

5 If $|y\rangle == |0\rangle$ go to step 1;

6 **return** $|x\rangle$;

Output: $|b\rangle = \sum_{i=0}^{2^n-1} b_i |i\rangle$, where b_i is the value of $f_k(i)$ scaled by the factor $\frac{1}{\sqrt{\mathrm{hw}(\mathcal{T}_{f_k})}}$ to produce a valid quantum state.

5. Cryptographic Use-Case

In this section, we want to use the HHL algorithm for Möbius transform to perform cryptanalysis of Boolean functions. We begin with an approach that allows creating the state $|\mathcal{A}_f\rangle$, which might be polynomially reconstructible for sparse functions.

5.1. Retrieving the Algebraic Normal Form

In Section 4.2, we showed how to generate the input corresponding to the $\mathcal{T}_f$ for a Boolean function f. The procedure runs in polynomial time and the result is a vector of the following form:

$$|\mathcal{T}_f\rangle = \frac{1}{\sqrt{\mathrm{hw}(\mathcal{T}_f)}} \sum_{\substack{i=0 \\ f(i)=1}}^{2^n-1} |i\rangle = \frac{1}{\sqrt{\mathrm{hw}(\mathcal{T}_f)}} \cdot \mathcal{T}_f$$

We also showed how to construct the matrix input Γ_1 for the HHL algorithm, which corresponds to the Boolean Möbius transform. The matrix is sparse and of low condition number and a constant size. The HHL algorithm would take constant time to run for a single instance of Γ_1, and $\tilde{\mathcal{O}}(n)$ time to apply the Möbius transform to the whole register (cf. Theorem 4).

As seen in Section 4.1, applying T_n to the vector $\mathcal{T}_f$ results in a vector which describes the algebraic normal form of the function f. A similar result is obtained when the vertical HHL algorithm with Γ_1 input is applied to the register $|\mathcal{T}_f\rangle$.

Theorem 5. *Let HHL_{Γ_n} be the vertical application of HHL_{Γ_1} to n qubits:*

$$HHL_{\Gamma_n} |b\rangle |ancilla\rangle := \prod_{j=1}^{n} \left(\bigotimes_{i=1}^{j-1} E_1 \otimes HHL_{\Gamma_1} \otimes \bigotimes_{i=j+1}^{n} E_1 \right) |b\rangle |ancilla\rangle$$

The ancilla register is needed to cover for the special Booleanized construction of Γ_1. Then, applying the HHL_{Γ_n} to the state $|\mathcal{T}_f\rangle$ results in the following register:

$$HHL_{\Gamma_n} |\mathcal{T}_f\rangle |ancilla\rangle = |\mathcal{A}_f\rangle |ancilla\rangle = \left(\frac{1}{\sqrt{hw(\mathcal{A}_f)}} \cdot \mathcal{A}_f \right) |ancilla\rangle$$

With an additional measurement of some helper register we can obtain the pure state $|\mathcal{A}_f\rangle$.

Proof. As seen in Theorem 4, applying the vertical HHL approach with Γ_1 input allows for computing of the preimage of the input vector $|b\rangle$. The result is equivalent to the preimage

of $\overrightarrow{b}$ under the Boolean Möbius transformation T_n. As seen in Claim 1, the result is a vector $\mathcal{A}_f$ holding the algebraic normal form of f scaled by a factor dependent on the superposition. Finally, we can discard the ancilla register (or perform some additional conditional measurement) and only consider the register $|\mathcal{A}_f\rangle$. □

The resulting register corresponds to the vector description of the ANF of f, scaled by a factor. The important observation is that only the coefficients which are one have a non-zero amplitude.

5.1.1. Sparse ANF Representation

Classically, reconstruction of an unknown Boolean function is a difficult problem. One of the approaches can be obtained from the learning theory and is based on the Fourier spectrum [2]. One can also use the Formula (1) mentioned in Section 1.2 to reveal the ANF. While obtaining the coefficients of low degree coefficients is not problematic, the complexity grows exponentially with the degree of the coefficient. On the other hand, if the degree of the coefficient is too high, it affects a small number of possible outputs and has a low influence on the truth table of the function.

When considering the result of the algorithm described in Theorem 5, we want to highlight that the state $|\mathcal{A}_f\rangle$ is an equally distributed superposition of all the base states which contribute to the ANF. Upon measurement, we obtain any of the ANF's active (non-zero) coefficients with the same probability. The difference, opposing the classical scenario, is that the number of steps we need for each coefficient is independent of the coefficient's degree. Especially for Boolean functions with sparse representation (polynomial number of coefficients with respect to n) and many high-order coefficients, the ANF reconstruction can be significantly sped up. Simple measurements can be combined with techniques like amplitude amplification to guarantee that the whole ANF is properly recovered in polynomial time [13].

In the Q1 model, retrieving the $\mathcal{A}_f$ can give us information about the secret key that is being used to generate some bit-stream. By combining information from multiple Boolean functions (e.g., multiple one-bit projections generated by the stream cipher) we could build a polynomial-size equation system which can be solved for the secret key. The ANF obtained in the Q2 model does not give us direct information about the key, but could be used to decrypt future ciphertexts.

5.1.2. Estimating the Number of Terms

In some cases, especially when $\mathrm{hw}(\mathcal{A}_f)$ is super-polynomial, we might consider other properties of f rather than its ANF. The state $|\mathcal{A}_f\rangle$ allows us to estimate the number of non-zero coefficients in the algebraic normal form of the Boolean function. We can achieve this using a technique called *Quantum Amplitude Estimation* [13].

Quantum amplitude estimation deals with a problem connected to the Grover's algorithm setting. However, instead of searching for an $\{x \in X : f(x) = 1\}$, we are interested in the size of said set. Similarly, as in Grover's algorithm, the algorithm divides the space into two subspaces following some indicator function f:

$$|\Psi\rangle = |\Psi_1\rangle + |\Psi_0\rangle$$

We call $|\Psi_1\rangle$ the good components and $|\Psi_0\rangle$ the bad components. We want to estimate the probability of measuring a good component. To achieve it, the algorithm uses the famous Grover's operator combined with phase estimation. After [13] was published, other potential candidates to solve the same problem were constructed [23–25].

We begin with a straightforward observation that the state $|\mathcal{A}_f\rangle$ is an equal superposition of all states which correspond to the non-zero coefficients of $\mathcal{A}_f$. This means that the amplitudes of $|\mathcal{A}_f\rangle$ are only zeros and ones scaled by a factor of $\frac{1}{\sqrt{\mathrm{hw}(\mathcal{A}_f)}}$. If we were able to estimate the amplitude of one of the active entries, we could determine the Hamming weight $\mathrm{hw}(\mathcal{A}_f)$. This is exactly the sparsity of the function f. Further, since all non-zero amplitudes are equal, we just need to find the amplitude of a single element from the ANF (e.g., by measuring $|\mathcal{A}_f\rangle$ we find the element from $\mathcal{A}_f$ and subsequently estimate its amplitude). Sometimes, we might not be interested in finding the exact number but rather proving that it is above a certain threshold. Since most of the amplitude estimation algorithms have their runtime dependent on an error threshold, we could verify if the amplitude is only smaller than some value, which guarantees the desired ANF sparseness.

We could iterate the above ideas to prove additional properties. By carefully choosing the function used for amplitude estimation, we could check how many higher-order terms are in the ANF. The *good states* (as defined in [13]) could be the terms with a degree greater than some threshold. This task would take exponential time in the classical scenario, but could be efficiently performed on a quantum computer.

5.1.3. Comparison to Classical Approaches

Retrieving the algebraic normal form of a Boolean function is a concept closely related to computational learning theory. In the classical scenario, given a set of samples of the form $\Big(x, f(x)\Big)_{x\in\mathcal{S}}$, we want to compute a function $\tilde{f}$ such that the following is true:

$$\Pr_{x\in\{0,1\}^n}\Big(\tilde{f}(x) = f(x)\Big) \geq c$$

for some non-negligible constant c. Obviously, reconstructing the original function f would solve this problem. However, since higher-order terms of $\mathcal{A}_f$ only influence a few outputs, ignoring terms with orders bigger than some predetermined value t might be beneficial. How many such high-order terms might be ignored depends on the value of c.

In [2], a learning approach is proposed, which relies on determining the Fourier coefficients of f instead of its ANF. As there are 2^n such coefficients, and computing each one is not a trivial task; for bigger n-values, this is not a manageable approach. The corollary below estimates the runtime of such an approach for a specific set of with total influence bounded by some value t.

Corollary 1 ([2]). *Let $\boldsymbol{I}(f)$ be the total influence of a Boolean function f, defined as:*

$$\boldsymbol{I}(f) = \sum_{S\subseteq[n]} |S|\hat{f}(S)^2,$$

where $\hat{f}(S)$ are the Fourier coefficients of f.

For $t \geq 1$, let $\mathcal{C} = \{f : \{-1,1\}^n \to \{-1,1\} | \boldsymbol{I}(f) \leq t\}$. Then $\mathcal{C}$ is learnable from random examples with error ϵ in time $n^{\mathcal{O}(t/\epsilon)}$.

An alternate approach would be learning the actual ANF coefficients of the function f. As mentioned in Section 1.2, the formula for a coefficient of the term x^I is as follows:

$$c_I = \sum_{\mathrm{supp}(x)\subseteq I} f(x).$$

So, to compute a coefficient of order $deg(I) = t$, we need to evaluate the function f at 2^t inputs and add them together. For higher-order terms, the number of computational steps is thus $\mathcal{O}(2^t)$.

In comparison, our approach depends solely on n and the Hamming weight of the algebraic normal form of f. The high-order coefficients are as easy to determine as the low-order terms. Furthermore, we are not aware of any classical algorithm other than trivial counting, which determines the Hamming weight of the ANF of a function.

Historically, certain sparse but high-order functions were used in cryptography. A famous example, the Toyocrypt cipher, was attacked in [26]. Even though the polynomial used there was not secret, some of the currently used ciphers may use functions with sparse polynomial representation.

5.2. Algebraic Transition Matrix

Another approach using a concept closely related to the Boolean Möbius Transform was proposed in [5]. For a vectorial Boolean function $F : \{0,1\}^n \to \{0,1\}^m$, ref. [5] define a linear operator mapping from δ_x to $\delta_{F(x)}$, where δ_x is the Kronecker delta function. The operator is called a transition matrix:

Definition 1 (Transition matrix [5]). *Let $\mathbb{K}$ be a field and let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function. Define $T^F : \mathbb{K}^{\mathbb{F}_2^n} \to \mathbb{K}^{\mathbb{F}_2^m}$ as the unique linear operator that maps δ_x to $\delta_{F(x)}$. The transition matrix of F is the coordinate representation of T^F with respect to the Kronecker delta bases of $\mathbb{K}^{\mathbb{F}_2^n}$ and $\mathbb{K}^{\mathbb{F}_2^m}$.*

The transition matrix has a set of useful properties which allow combining smaller transformations to a bigger, more complex one [5]. However, in this paper, we will not focus on those and rather will show that, in the Q2 model, we can obtain oracle access to the transition matrix.

Example 3 ([5]). *Consider a function $F : \mathbb{F}_2^2 \to \mathbb{F}_2^2$ defined as*

$$F(x_1, x_2) = (F_1(x_1, x_2), F_2(x_1, x_2)) \mapsto (x_2 + 1, x_1 + x_1 x_2)$$

The truth table of such a function is as follows:

$$\begin{aligned} F :(0,0) &\mapsto (1,0) \\ (1,0) &\mapsto (1,1) \\ (0,1) &\mapsto (0,0) \\ (1,1) &\mapsto (0,0) \end{aligned}$$

The transition matrix T^F is then: as follows

$$T^F = \begin{array}{c} \\ (0,0) \\ (1,0) \\ (0,1) \\ (1,1) \end{array} \begin{array}{c} \begin{array}{cccc} (0,0) & (1,0) & (0,1) & (1,1) \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

We want to highlight that, given a column index i, the only non-zero entry appears in the $F(i)$'th row. If we allow superposition queries to the function F (as is assumed in the Q2 model), we have a quantum column-oracle for the transition matrix. To obtain the row-oracle which, given a row index, returns the indices of non-zero entries in that

row, we additionally need an oracle for the F^{-1} function. For a permutation, the oracle acts as a bijection. However, as seen in the example above, this must not be the case in general. With some assumptions about the maximal number of elements ν that map to the same value, we can still solve this problem with a bigger output register. We also want to highlight that such a matrix is well-suited for the HHL algorithm—the sparseness and condition number can be upper-bound by $s = \mathcal{O}(\nu) = \kappa$. Especially for F, which is a permutation, the runtime of HHL with T^F is $\tilde{\mathcal{O}}(n+m)$.

While the transition matrix is not necessarily useful in the HHL setting, they are a building block of a more interesting construction. We will first introduce a dual to the Boolean Möbius transform, as defined in [5], and then use it to build an algebraic transition matrix.

Definition 2 (Dual Boolean (binary) Möbius Transformation [5])**.** *The dual Boolean Möbius transformation of a Boolean function $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$ is the Boolean function $\mathcal{P}_n(f) = \tilde{f} \in \mathbb{F}_2^{\mathbb{F}_2^n}$, where $\tilde{f}(u)$ is the coordinate of f corresponding to π_u in the precursor basis. The precursor basis vectors π_u are the indicator function of precursor sets:*

$$\pi_u = \mathbb{1}_{\{supp(x) \subseteq u\}}$$

The matrix representation of $\mathcal{P}_n$ has the following form:

$$\mathcal{P}_n = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^{\otimes n}$$

Observe that the dual Boolean Möbius transform $\mathcal{P}_n$ is actually the transpose of T_n defined in Section 4.1. By techniques similar to the ones mentioned above, we can also Booleanize $\mathcal{P}_n$. With the definitions above, we can finally introduce the algebraic transition matrices:

Definition 3 (Algebraic Transition Matrix [5])**.** *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function. Define $A^F : \mathbb{F}_2^{\mathbb{F}_2^n} \to \mathbb{F}_2^{\mathbb{F}_2^m}$ as the dual Boolean Möbius transformation of T^F. That is:*

$$A^F = \mathcal{P}_m T^F \mathcal{P}_n^{-1}$$

The algebraic transition matrix of F is the coordinate representation of T^F with respect to the precursor bases of $\mathbb{F}_2^{\mathbb{F}_2^n}$ and $\mathbb{F}_2^{\mathbb{F}_2^m}$.

The algebraic transition matrix can be computed as a horizontal decomposition of three matrices, two dual Möbius transforms and a transition matrix. We observe that like the Boolean Möbius transform, its dual is self-inverse. So in fact, we can compute A^F as follows:

$$A^F = \mathcal{P}_m T^F \mathcal{P}_n$$

The HHL algorithm for each of those matrices runs in time $\mathcal{O}(m+n)$. The two dual transforms can be vertically decomposed (cf. Figure 1b) using the technique from Section 4.2, while T^F is well-suited for HHL as we argued above. We can combine the three HHL routines by straight concatenation:

$$HHL_{A^F} |b\rangle = HHL_{\mathcal{P}_m} \circ HHL_{T^F} \circ HHL_{\mathcal{P}_n} |b\rangle$$

As a result, we can compute the preimage of a quantum state $|b\rangle$ under the algebraic transition matrix in polynomial time by combining the classical HHL algorithm with the vertical HHL.

5.2.1. Interpretation of ATMs Mappings

The algebraic transition matrix gives us deep insights into the vectorial Boolean function which is being investigated. In [5], the authors show that, for $I = (i_1, \ldots, i_m)$, $J = (j_1, \ldots, j_n)$, the matrix A^F following holds:

$$(A^F)_{(I,J)} = 1 \iff \mathcal{A}_{(\prod_{k=1}^{m} F_k^{i_k})}|_J = 1,$$

where $\mathcal{A}_{(\prod_{k=1}^{m} F_k^{i_k})}|_J$ is the Jth coefficient c_J of the algebraic normal form of the Boolean function f as defined in Section 1.2.

Using the HHL algorithm does not give us direct access to the algebraic transition matrix. However, ref. [5] comes with a theorem which explains the result of multiplying A^F by some vector $g \in \mathbb{F}_2^{\mathbb{F}_2^m}$:

Theorem 6 (Property of Algebraic Transition Matrices [5])**.** *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function and A^F its algebraic transition matrix. Further, let $g : \mathbb{F}_2^m \to \mathbb{F}_2$ be a Boolean function with m inputs. Then:*

$$\mathcal{A}_{g \circ F} = (A^F)^T \mathcal{A}_g$$

In other words, for a given Boolean function of the form $h = g \circ F$, using the HHL algorithm with input matrix A^F and a vector $|h\rangle$, we can find the state $|g\rangle$ which encapsulates the searched function.

Example 4 ([5])**.** *Consider a function $F : \mathbb{F}_2^2 \to \mathbb{F}_2^2$ defined as follows*

$$F(x_1, x_2) = (F_1(x_1, x_2), F_2(x_1, x_2)) \mapsto (x_2 + 1, x_1 + x_1 x_2)$$

The algebraic transition matrix A^F is then as follows:

$$A^F = \begin{array}{c} \\ 1 \\ F_1 \\ F_2 \\ F_1 \cdot F_2 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & x_1 & x_2 & x_1 x_2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

The rows correspond to products of the projection functions F_i's, while the columns indicate whether the coefficient of each x^I is one or zero.

The algebraic transition matrix also allows quick computation of the Boolean function obtained by composition of F with any Boolean function. For example for the Boolean function $g(x_1, x_2) = x_1 + x_2$, we can compute $g \circ F$:

$$(A^F)^T \cdot \mathcal{A}_g = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}^T \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$
\begin{aligned}
&= \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \\
&= 1 + x_1 + x_2 + x_1 x_2 \\
&= g \circ F
\end{aligned}
$$

A possible attack scenario includes finding a Boolean form to compute the value of each x_i from the output of the function F. For example, to find the formula for x_1, the input vector to the HHL algorithm would have to be as follows:

$$
|b\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{matrix} 1 \\ x_1 \\ x_2 \\ \\ x_1 x_2 \\ \\ \prod_{i=1}^{m} x_i \end{matrix}
$$

We highlight that the matrix to be inverted is not A^F, but $(A^F)^T$. This is not a problem as we can just switch the row- and column-oracles to obtain the transpose of the matrix T^F and use the following fact:

$$
\begin{aligned}
(A^F)^T &= (\mathcal{P}_m T^F \mathcal{P}_n)^T \\
&= \mathcal{P}_n^T \cdot (T^F)^T \cdot \mathcal{P}_m^T \\
&= \mathcal{P}_n \cdot (T^F)^T \cdot \mathcal{P}_m
\end{aligned}
$$

For each low-order term x^I, we can use the techniques used in Section 5.1.2 to verify whether the preimage has a small Hamming weight. If such an input is found, there exists a sparse Boolean function that allows the computation of the value of x^I. We can recover such a function and use it to deduce the values of x_i for $i \in I$. For example, we can use the fact that if $x^I = 1$, then $\forall i \in I$ follows $x_i = 1$.

5.2.2. Comparison to Classical Approaches

In the classical setting, to find a formula to compute certain bits, the attacker would have to analyze the ciphers algebraic properties. The concept is similar to algebraic cryptanalysis, where the cipher is rewritten as a set of equations for which the solution is the secret key [27].

We are, however, not interested in building a huge equation system, but rather want to find a function which, combined with the original function, leaks secret bits. The idea resembles the concept of annihilators, introduced in [26]. An annihilator of a Boolean function $f(x) : \mathbb{F}_2^n \to \mathbb{F}_2$ is another Boolean function $g(x) : \mathbb{F}_2^n \to \mathbb{F}_2$ such that the following is true:

$$
f(x) \cdot g(x) = 0 \ \ \forall x \in \mathbb{F}_2^n
$$

The annihilators allow reducing the degree of algebraic equations and, as a consequence, finding some of the secret key bits used in encryption. In our approach, we are

rather interested in analyzing a vectorial Boolean function $F(x) : \mathbb{F}_2^n \to \mathbb{F}_2^m$ using a Boolean function $g(x) : \mathbb{F}_2^m \to \mathbb{F}_2$ such that the following is true:

$$g \circ F(x) = x^I \quad \forall x \in \mathbb{F}_2^n,$$

but the desired outcome is similar.

6. Conclusions

In this paper, we have shown how tensor-decomposable matrices can be used in the HHL setting to improve the runtime of the linear system solving algorithm proposed in [3]. We have proven that a vertical decomposition has a direct influence on the run-time, and in the quantum setting allows preimage computation even if the input to be inverted is not tensor-decomposable. This is a direct improvement over the classical computation.

Next, we have shown a polynomial-time procedure to create a quantum state which describes the algebraic normal form of a Boolean function f. The register has the form $|\mathcal{A}_f\rangle = \sum_{i=0}^{2^n-1} \frac{1}{\sqrt{\mathrm{hw}(\mathcal{A}_f)}} \cdot \mathcal{A}_f$, where $\mathcal{A}_f$ represents the coefficient vector of f. We further show how $|\mathcal{A}_f\rangle$ could be used to extract some information about the ANF. In most straightforward scenarios, a distinguisher attack could be performed to differentiate a cryptographic function from a random function. The Hamming weight of the ANF of a random function will be roughly $N/2$. Meanwhile, a cryptographic function is likely to deviate from this value. The attack could also be used as a testing method for new ciphers. It would allow a quicker verification of the sparsity of ANF representations.

We show how a construct called an algebraic transition matrix can be used in the HHL setting to improve the cryptanalysis of Boolean functions. By using the polynomial-time HHL algorithm for all three parts of the horizontal decomposition of A^F, with specific input, we can find formulas for the values of the key used in the encryption process.

We finally point to the fact that the Boolean Möbius transform T_n is self-inverse. Therefore, it is not necessary to compute the preimage under the Möbius transformation. We could compute the value $T_n \cdot |b\rangle$ and obtain the same result. We would not have to invert the eigenvalue λ_k, but use it as a factor. It remains an open question how significantly such an alteration improves the runtime. We believe that, for some use cases, it might pose an interesting question.

Author Contributions: Conceptualisation, methodology, formal analysis, writing—original draft preparation, writing—review and editing C.P., supervision, project administration and proof-reading M.M. All authors have read and agreed to the published version of the manuscript.

Funding: Supported by Open Access funds of Freie Universität Berlin.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hosoyamada, A. Quantum Speed-Up for Multidimensional (Zero Correlation) Linear Distinguishers. In *Advances in Cryptology, Proceedings of the ASIACRYPT 2023, Guangzhou, China, 4–8 December 2023* ; Springer: Singapore, 2023; pp. 311–345. [CrossRef]
2. O'Donnell, R. *Analysis of Boolean Functions*; Cambridge University Press: Cambridge, UK, 2014. [CrossRef]
3. Harrow, A.W.; Hassidim, A.; Lloyd, S. Quantum Algorithm for Solving Linear Systems of Equations. *Phys. Rev. Lett.* **2009**, *103*, 150502. [CrossRef] [PubMed]
4. Chen, Y.A.; Gao, X.S. Quantum Algorithm for Boolean Equation Solving and Quantum Algebraic Attack on Cryptosystems. *J. Syst. Sci. Complex.* **2022**, *35*, 373–412. [CrossRef]
5. Beyne, T.; Verbauwhede, M. Integral Cryptanalysis Using Algebraic Transition Matrices. *IACR Trans. Symmetric Cryptol.* **2023**, *2023*, 244–269. [CrossRef]

6. Hoffstein, J.; Pipher, J.; Silverman, J.H. NTRU: A Ring-Based Public Key Cryptosystem. In *Algorithmic Number Theory*; Goos, G., Hartmanis, J., Van Leeuwen, J., Buhler, J.P., Eds.; Springer: Berlin/Heidelberg, Germany, 1998; Volume 1423, pp. 267–288. [CrossRef]
7. Chen, Y.A.; Gao, X.S.; Yuan, C.M. Quantum Algorithm for Optimization and Polynomial System Solving over Finite Field and Application to Cryptanalysis. *arXiv* **2018**, arXiv:1802.03856v2. [CrossRef]
8. Liu, W.; Gao, J. Quantum Security of Grain-128/Grain-128a Stream Cipher against HHL Algorithm. *Quantum Inf. Process.* **2021**, *20*, 343. [CrossRef]
9. Ding, J.; Gheorghiu, V.; Gilyén, A.; Hallgren, S.; Li, J. Limitations of the Macaulay Matrix Approach for Using the HHL Algorithm to Solve Multivariate Polynomial Systems. *Quantum* **2023**, *7*, 1069. [CrossRef]
10. Gao, J.; Li, H.; Wang, B.; Li, X. AES-128 under HHL Algorithm. *Quant. Inf. Comput.* **2022**, *22*, 0209–0240. [CrossRef]
11. Murphy, S.; Robshaw, M.J. Essential Algebraic Structure within the AES. In *Advances in Cryptology, Proceedings of the CRYPTO 2002, Santa Barbara, CA, USA, 18–22 August 2022*; Goos, G., Hartmanis, J., Van Leeuwen, J., Yung, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2442, pp. 1–16. [CrossRef]
12. Berry, D.W.; Ahokas, G.; Cleve, R.; Sanders, B.C. Efficient Quantum Algorithms for Simulating Sparse Hamiltonians. *Commun. Math. Phys.* **2007**, *270*, 359–371. [CrossRef]
13. Brassard, G.; Hoyer, P.; Mosca, M.; Tapp, A. Quantum Amplitude Amplification and Estimation. *AMS Contemp. Math. Ser.* **2000**, *305*. [CrossRef]
14. Ambainis, A. Variable Time Amplitude Amplification and a Faster Quantum Algorithm for Solving Systems of Linear Equations. *arXiv* **2010**, arXiv:1010.4458. [CrossRef]
15. Childs, A.M.; Kothari, R.; Somma, R.D. Quantum Algorithm for Systems of Linear Equations with Exponentially Improved Dependence on Precision. *SIAM J. Comput.* **2017**, *46*, 1920–1950. [CrossRef]
16. Gurvits, L. Classical Deterministic Complexity of Edmonds' Problem and Quantum Entanglement. In Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, San Diego, CA, USA, 9–11 June 2003; pp. 10–19. [CrossRef]
17. Barbier, M.; Cheballah, H.; Le Bars, J.M. On the Computation of the Möbius Transform. *Theor. Comput. Sci.* **2020**, *809*, 171–188. [CrossRef]
18. Carlet, C. Boolean Functions for Cryptography and Error-Correcting Codes. In *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, 1st ed.; Crama, Y., Hammer, P.L., Eds.; Cambridge University Press: Cambridge, UK, 2010; pp. 257–397. [CrossRef]
19. Pieprzyk, J.; Wang, H.; Zhang, X.m. Möbius Transforms, Coincident Boolean Functions and Non-Coincidence Property of Boolean Functions. *Int. J. Comput. Math.* **2011**, *88*, 1398–1416. [CrossRef]
20. Kuwakado, H.; Morii, M. Quantum Distinguisher between the 3-Round Feistel Cipher and the Random Permutation. In Proceedings of the 2010 IEEE International Symposium on Information Theory, Austin, TX, USA, 13–18 June 2010; pp. 2682–2685. [CrossRef]
21. Kaplan, M.; Leurent, G.; Leverrier, A.; Naya-Plasencia, M. Breaking Symmetric Cryptosystems Using Quantum Period Finding. In *Advances in Cryptology, Proceedings of the CRYPTO 2016, Santa Barbara, CA, USA, 14–18 August 2016*; Robshaw, M., Katz, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 207–237. [CrossRef]
22. Nielsen, M.A.; Chuang, I.L. *Quantum Computation and Quantum Information*, 10th ed.; Cambridge University Press: Cambridge, UK, 2010. [CrossRef]
23. Suzuki, Y.; Uno, S.; Raymond, R.; Tanaka, T.; Onodera, T.; Yamamoto, N. Amplitude Estimation without Phase Estimation. *Quantum Inf. Process.* **2020**, *19*, 75. [CrossRef]
24. Vazquez, A.C.; Woerner, S. Efficient State Preparation for Quantum Amplitude Estimation. *Phys. Rev. Appl.* **2021**, *15*, 034027. [CrossRef]
25. Grinko, D.; Gacon, J.; Zoufal, C.; Woerner, S. Iterative Quantum Amplitude Estimation. *Npj Quantum Inf.* **2021**, *7*, 52. [CrossRef]
26. Courtois, N.T.; Meier, W. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology, Proceedings of the EUROCRYPT 2003, Warsaw, Poland, 4–8 May 2003*; Biham, E., Ed.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 345–359. [CrossRef]
27. Bard, G.V. *Algebraic Cryptanalysis*; Springer: Boston, MA, USA, 2009. [CrossRef]