HHL for tensor-decomposable matrices

Cezary Pilaszewicz[1][0009−0006−6162−3676] and Marian Margraf[1]

Freie Universität Berlin, Takustr. 9, 14195 Berlin, Germany
firstname.lastname@fu-berlin.de

Abstract. We use the HHL algorithm to retrieve a quantum state holding the algebraic normal formal 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 corresponds 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 how some information about the ANF can be extracted from such a state.

1 Introduction

The HHL algorithm, proposed in 2009, is a quantum procedure that computes some given input's preimage. Unlike the classical approaches, it does not linearly depend on the size of the matrix. Rather its runtime depends on the sparseness and condition number of the matrix, and only logarithmically on the matrix's size. Since most nowadays used ciphers can be described as some form of exponentially big equation system, the idea of using HHL to cryptanalyse ciphers came forward [6] 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, however, instead of inverting an exponential-size matrix, we will first decompose the matrix into a tensor product and invert each of the sub-matrices. The matrix to be inverted is the matrix T_n describing the Boolean Möbius transform. We decided to decompose the matrix since in its full form it has high sparsity. This means that the HHL runtime for such a matrix would be comparable to the classical approach. Instead, each of the sub-matrices is small (2×2 before Booleanization) and therefore easily invertable.

By computing the preimage of the input under the Boolean Möbius transform, we manage to encapsulate the algebraic normal form of the function under attack in the quantum register. The result is:

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

where c_I is the coefficient of the monomial x^I. This can be used to both retrieve the ANF of the function, as well as to estimate the Hamming weight of $\mathcal{A}_f$.

We use the HHL algorithm to overcome the fact that the matrices to be inverted are not unitary, a classical requirement for quantum computation. The HHL is a ready-to-use framework, which can be used for this operation. However, we also notice that the algorithm could be improved. Since the T_n is self-inverse, we don't need to compute the preimage, we could instead apply the matrix T_n to our input.

We introduce the notation used through this paper in Section 1.1. Chapter 2 focuses on the HHL algorithm. We estimate the runtime, show how to Booleanize a matrix and how the HHL algorithm is used in cryptanalysis. In chapter 3 we discuss our new approach to apply HHL to a tensor product. Section 4 discusses the cryptographic significance of the new HHL. We introduce the Boolean Möbius transform and its tensor decomposition. In Section 4.2 we combine the two ideas. We explain why it's 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 talk about how the resulting state can be used to analyse the algebraic normal form of a Boolean function.

1.1 Notation

We will shortly introduce the common notation used through 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 $f(x) \in \mathbb{F}_2[x_1, \ldots, x_n]$ [19]. Let $c_I \in \mathbb{F}_2$ and $x^I := \prod_{i \in I} x_i$. 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:

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

where $\mathrm{supp}(x) := \{i : x_i \neq 0\}$ [19]. In other words, we can compute the coefficient c_I by adding (over $\mathbb{F}_2$) the images of all inputs dominated by I.

The Hamming weight of a vector $hw(x)$ is its number of non-zero entries. The degree of a coefficient can be computed using the Hamming weight:

$$deg(c_I) = hw(I)$$

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 ith component will be written as $s|_i$.

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 [12] as an efficient algorithm to solve exponentially big systems of linear equations, outperforming any classical algorithm. Its cryptographic significance got quickly discovered and [6] proposed the first attack scenario. Since then, a series of cryptographic publications have considered the HHL setting.

The algorithm takes as input a Hermitian matrix $A \in \mathbb{C}^{N \times N}$ and a vector $\overrightarrow{b} \in \mathbb{C}^N$ and computes the value $\overrightarrow{x}$ such that:

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

To do this, we need to represent $\overrightarrow{b}$ as a quantum register $|b\rangle = \sum_{i=0}^{N-1} b_i |i\rangle$. Then, A is transformed into a unitary operator e^{iAt}. Next, e^{iAt} is applied to $|b\rangle$ using the Hamiltonian simulation technique [3]. This is equivalent to decomposing $|b\rangle$ to an eigenbasis of A and determining the corresponding eigenvalues. The register after this transformation is in the state:

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

where $|b\rangle = \sum_{j=0}^{N-1} \beta_j |u_j\rangle$. The eigenvalues $|\lambda_j\rangle$ are then used to invert the matrix-application of A, and the register ends in the state:

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

2.1 HHL Runtime

In [12], authors show how to run the HHL algorithm for a matrix $A \in \mathbb{C}^{N \times N}$ in time $\tilde{\mathcal{O}}\left(\kappa^2 \cdot s^2 \cdot poly(\log N)\right)$, where s is the sparseness of the matrix A, 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 $\log N$). They require that A is efficiently row-computable, or access to an oracle that returns the indices of non-zero matrix entries given a row-index. A significant portion of the runtime is covered by the phase estimation algorithm, which for an s-sparse matrix takes $\tilde{\mathcal{O}}(ts^2)$. [1] improved the HHL algorithm by lowering the condition number dependence from quadratic to almost linear. [8] proposed a further improvement to achieve the runtime:

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

This will be the complexity that we will use in this paper. However, 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

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 needed for 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 matrix the Booleanization of M, or a Booleanized matrix. For a Boolean equation $f : \mathbb{F}_2^n \to \mathbb{F}_2$, $\#f$ is the number of non-zero coefficients in its ANF. We could also see it as the Hamming weight of the vector $\mathcal{A}_f$. The total sparseness of a matrix is the sum of entries in all rows of said matrix.

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

$$f_i(x_1, ..., x_n) - z_i = 0 \qquad \forall i = 1, ..., m \tag{2}$$
$$z_i/2 \in \mathbb{Z} \qquad \forall i = 1, ..., m \tag{3}$$
$$x_j - x_j^2 = 0 \qquad \forall j = 1, ..., n \tag{4}$$

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 $x_1, ..., x_n$ are from $\mathbb{F}_2$. An important observation here is also that $\forall i = 1, ..., m\ 0 \leq z_i \leq \#f_i$.

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 (3) 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:

$$f_i(x_1, ..., x_n) - \sum_{b=1}^{\log \#f_i} 2^b z_{i_b} = 0 \qquad \forall i = 1, ..., m$$

$$z_{i_b} - z_{i_b}^2 = 0 \qquad \forall i = 1, ..., m, \forall b = 1, ..., \log \# f_i$$
$$x_j - x_j^2 = 0 \qquad \forall j = 1, ..., n$$

Similarly as in (4), the equation $z_{i_b} - z_{i_b}^2 = 0$ forces the values of $z_{i_b} \in \mathbb{F}_2$.

The adjustments mentioned in this section increased the number of needed variables from n to $n + m \cdot \log \# f_i$ and the number of equations from m to $m \cdot (1 + \log \# f_i) + n$.

Further possible improvements
In [9] authors suggest further improvements to the above construction. They use the Valiant-Vazirani affine hashing method [22] to ensure that the polynomial system has only one solution. To achieve this, for a system with S many solutions, they introduce $\mathcal{O}(\log S)$ random linear equations. They also rewrite the equation system in such a way, that only one of the equations has a constant term equal to -1. The two techniques increase the number of equations and the sparseness only by a small factor but should result in a polynomial system that is easier to handle [9].

2.3 State-of-the-art of HHL in Cryptanalysis

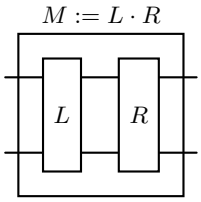
The cryptographic appeal of the HHL algorithm is clear. Most good ciphers are characterized by complicated equation systems needed to describe them. The first attempt was introduced in [6], and used the so-called Macaulay matrix - an exponentially big and sparse matrix which can be solved for the Boolean solution of a polynomial system. The next proposal targeted the NTRU system [13] and used the HHL algorithm to solve a polynomial system with noise [7]. Finally, in [16], the Grain-128 and Grain-128a ciphers were cryptanalysed using techniques based on [6,7]. [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 [17], were analysed under the HHL algorithm.

3 HHL for tensors of matrices

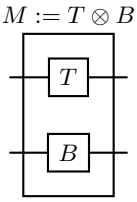
Usually, when considering the use-case of HHL we want to leverage the exponential speed-up in regard to the size of the matrix. This is the natural direction as that's the obvious runtime difference between the HHL and the classical approach. The hindrance is, however, that even though the matrix can 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 chapter, we present an approach to overcome the above-mentioned obstacles for a special set of matrices. We show, that if a matrix M can be decomposed as $M = M_1 \otimes M_2 \otimes ... \otimes M_n$, where each M_i has a small size,

applying the HHL approach to each M_i has low runtime and delivers the same result. This allows overcoming the exponential cost of computing the preimage given a matrix M and some output $|b\rangle$. A similar technique is a building block of known quantum algorithms like Shor's algorithm or Simon's algorithm. There, e.g., instead of applying a matrix H_n to an n-long register $|x\rangle$, we apply H_1 to each of n qubits of $|x\rangle$.



(a) Horizontal decomposition



(b) Vertical decomposition

We start with a simple Lemma saying that applying a vertical (tensor) decomposition of a matrix M to quantum sub-registers results in the same state as applying the matrix M to the whole register:

Lemma 1. *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.

Lemma 2. *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 product and tensor of two matrices is a matrix, we will show the property just for two matrices. The rest follows by 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:

$$(M_1 \otimes E_2) \cdot (E_1 \otimes M_2) = (M_1 \cdot E_1) \otimes (M_2 \cdot E_2)$$

$$= M_1 \otimes M_2$$
$$= M$$

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

We want to highlight that the result vector of applying the matrix from Lemma 2 does not have to be a valid quantum state. Since the matrix M is not unitary, it means the result does not have to be normed. Instead, we will move to the HHL setting where the state to be reverted is encoded into the amplitudes of the quantum register:

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 frame this in a quantum setting:

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

We will now use the result of Lemma 2 to solve Problem 2 for a matrix $M \in \mathbb{C}^{2^n \times 2^n}$ such that $M = \bigotimes_{i=1}^{t} M_i$. Instead of applying HHL to the whole matrix M, we will consider its vertical decomposition and apply HHL to the submatrices. The idea is depicted in Figure 2. The resulting state will be identical to that of HHL applied to the matrix M.

Theorem 1. *Let $M = \bigotimes_{i=1}^{t} M_i$ be a vertical decomposition of M. Further, 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.

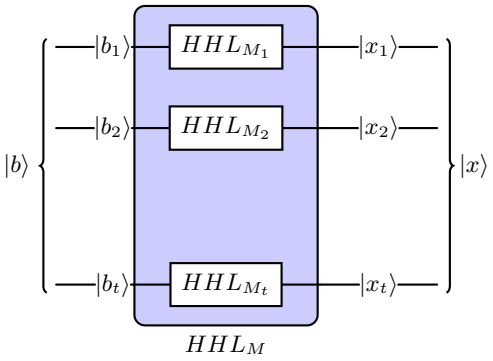


Fig. 2: Vertical HHL

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. Then, by the Lemma 2, the following constructions are equal:

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

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:

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

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 2. *Let $M = \bigotimes_{i=1}^{t} M_i$ be a horizontal decomposition of M. Further, $\forall i$ let κ_i be the condition of the matrix $M_i \in \mathbb{C}^{N_i \times N_i}$ and s_i its sparseness. The time complexity of the vertical HHL for 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 we saw in Section 2.1, this corresponds to a time complexity of:

$$\tilde{\mathcal{O}}\left(\max_i \kappa_i \cdot s_i \cdot poly(\log N_i)\right) \tag{5}$$

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 (5). This results in the runtime of:

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

4 Cryptographic Use-cases

4.1 Möbius Transform of Boolean Functions

In this chapter, we introduce the Boolean Möbius transform $\mu : \mathbb{F}_2^{2^n} \to \mathbb{F}_2^{2^n}$. It acts as the bijective mapping between the ANF of a Boolean function f, and its truth table $\mathcal{T}_f$. It acts 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 [2] (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 [5,20].

Lemma 3. *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:

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

Lemma 4. *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 3, we know T_n can be described as:

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

Further, by Lemma 2, 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:

Let $f(X_1, X_2) = X_1 \oplus X_1X_2$. 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 4:

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

It should be clear that the formula for the ANF of f from Lemma 4 resembles the formula mentioned in Theorem 1. We will next show that 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 chapter, we show how to tackle a specific initialization of Problem 2. We will try to 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 $\mathbb{F}_2$. To do this, 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. The key point here is, however, that this 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. Luckily, we can use the procedure proposed by the original authors of the HHL algorithm to cover this case [12]. The main idea is, if a matrix A is not Hermitian, we can construct a new matrix C:

$$C := \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix},$$

and the matrix C will be Hermitian. Here the $A^\dagger$ is the conjugate transpose of A. Since all the entries of A are real, $A^\dagger = A^T$. For matrices of exponentially big sparsity, this is an additional requirement. Instead of just requiring an oracle for the entries in a row, we now also need the column-oracle of the matrix A (cf. Section 2.1).

Further, the input vector must be adjusted. While for A, 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, $\tilde{x} := \begin{pmatrix} 0 \\ \overrightarrow{x} \end{pmatrix}$. To see this, consider:

$$\begin{aligned} C \cdot \tilde{x} &= \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ \overrightarrow{x} \end{pmatrix} \\ &= \begin{pmatrix} A \cdot \overrightarrow{x} \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} \overrightarrow{b} \\ 0 \end{pmatrix} \\ &= \tilde{b} \end{aligned} \tag{6}$$

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. However, since there is always one full column, the sparseness of T_n is exponential:

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

To circumvent that, instead of using T_n as input to HHL, we will first vertically decompose it.

Booleanized T_n vs Booleanized T_1

To overcome the exponentially big runtime of HHL_{T_n}, we will instead call the HHL_{T_1} procedure for each 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 to be inverted, and then perform the reduction on each sub-matrix. [7] and [16] mentioned a set of additional/alternative equations, which when incorporated into the equation system, allow for $\mathbb{F}_2$-computation. In the example below, we present one candidate for such a matrix.

Example:
Using the techniques mentioned in Section 2.2, we contruct a matrix $\Gamma \in \mathbb{C}^{4\times 5}$.

For $a, b \in \mathbb{F}_2$, the preimage under Γ of the vector $\overrightarrow{b} = \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ 0 \end{pmatrix}$ 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 as:

$$\Gamma = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -2 & -2 & 2 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

such, that:

$$\Gamma \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ 0 \\ 0 \end{pmatrix}, \quad \Gamma \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{1} \\ \mathbf{1} \\ 0 \\ 0 \end{pmatrix}, \Gamma \begin{pmatrix} \mathbf{0} \\ \mathbf{1} \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{1} \\ 0 \\ 0 \end{pmatrix}, \quad \Gamma \begin{pmatrix} \mathbf{1} \\ \mathbf{1} \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \\ 0 \\ 0 \end{pmatrix}$$

With further improvements mentioned in Section 2.2, like the hashing technique, we can guarantee that the preimages are always the ones shown here.

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 also needs to be Hermitianized before being plugged into the HHL (see Equation (6)). The result will be a matrix Γ. We want to also highlight that the sparsity in the Example above does not depend on the number of qubits:

$$s(\Gamma) = 5$$

and the condition number κ_Γ is constant.

With the machinery in place, we can solve Problem 3 using the algorithm from Theorem 1. Further, we can estimate the runtime of the algorithm:

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

Proof.
As seen in Theorem 2, for a matrix decomposition $M = \bigotimes_{i=1}^{t} M_i$, the runtime is:

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

In this case, all M_i's correspond to the Booleanized T_1 matrix. We can use the techniques mentioned in Section 2.2 to create a Booleanization of T_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 $s_\Gamma, \kappa_\Gamma, N_\Gamma$ be the sparsity, condition number and size of the matrix Γ respectively. Combining the above with the upper-bound, we get the collected run-time for the algorithm:

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

Keeping in mind, that Problem 3 is an initialization of Problem 2, as well as Theorem 3 is an initialization of Theorem 1, the claim follows.

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 [15,14].

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 [18]. 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:

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

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. We show the exact algorithm 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 $\frac{1}{\sqrt{hw(\mathcal{T}_f)}}$ 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 $\frac{1}{\sqrt{hw(\mathcal{T}_{f_k})}}$ to produce a valid quantum state.

4.3 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 form:

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

We also showed how to construct the matrix input Γ for the HHL algorithm, which corresponds to the Boolean Möbius transform. The matrix is sparse, of low condition number and constant size. This means the HHL algorithm would take constant time to run for a single instance of Γ, and $\tilde{\mathcal{O}}(n)$ time to apply the Möbius transform to the whole register (cf. Theorem 3).

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 input is applied to the register $|\mathcal{T}_f\rangle$.

Theorem 4. *Let HHL_{Γ_n} be the vertical application of HHL_Γ to n qubits:*

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

The ancilla register is needed to cover for the special Booleanized construction of Γ. 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 3, applying the vertical HHL approach with Γ input allows computing the preimage of the input vector $|b\rangle$. This is equivalent to computing 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. This is important, as only the coefficients which are one have a non-zero amplitude.

5 ANF Analysis

In the last chapter, we saw how to efficiently encapsulate information about the algebraic normal form of a Boolean function into a register. However, as usually with quantum algorithms, extracting the information from the quantum register is not a trivial task. In this chapter, we want to go through possible approaches and scenarios in which they might work.

5.1 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 [19]. One can also use the formula (1) mentioned in Section 1.1 to reveal the ANF. While getting 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 4, we want to highlight that state $|\mathcal{A}_f\rangle$ is an equally distributed superposition of all the base states which contribute to the ANF. This means that 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 amount of steps we need is independent of the degree of the coefficient. This might be especially useful when a Boolean function f with a sparse representation (polynomial number of coefficients with respect to n), and many high-order coefficients is being learned. Simple measurements can be combined with techniques like amplitude amplification to guarantee that the whole ANF is properly recovered in polynomial time [4].

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.

5.2 Estimating the Number of Terms

In some cases, especially when $hw(\mathcal{A}_f)$ 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 do this using a technique called *Quantum Amplitude Estimation* [4].

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 [4] was published, other potential candidates to solve the same problem were constructed[21,23,11].

We begin with a simple 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 $\frac{1}{\sqrt{hw(\mathcal{A}_f)}}$. If we were able to estimate the amplitude of one of the active entries, we could determine the Hamming weight $hw(\mathcal{A}_f)$. This is exactly the sparsity of the function f. Further, since all non-zero amplitudes are equal, we just need to find a single element from the ANF (e.g., by measuring $|\mathcal{A}_f\rangle$). Sometimes, we might not be interested in finding the exact number but rather prove that it's 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 further 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 [4]) could be the terms with a degree greater than some threshold. This is a task that would take exponential time in the classical scenario but could be efficiently done on a quantum computer.

6 Conclusion

In this paper, 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 $|\mathcal{A}_f\rangle = \sum_{i=0}^{2^n-1} \frac{1}{\sqrt{hw(\mathcal{A}_f)}} \cdot \mathcal{A}_f$, 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 the simplest scenario, 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 also point to the fact that the Boolean Möbius transform T_n is self-inverse. This means that it is not necessary to compute the preimage under the Möbius transformation. We could also compute the value $T_n \cdot |b\rangle$ and get the same result. This would mean we do not need to invert the eigenvalue λ_k, but use it as a factor. It remains an open question how much that would reduce the runtime of the resulting algorithm. We believe for some use-cases it might pose an interesting question.

References

1. Ambainis, A.: Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations. https://arxiv.org/abs/1010.4458v2 (Oct 2010)
2. Barbier, M., Cheballah, H., Le Bars, J.M.: On the computation of the Möbius transform. Theoretical Computer Science **809**, 171–188 (Feb 2020). https://doi.org/10.1016/j.tcs.2019.12.005
3. Berry, D.W., Ahokas, G., Cleve, R., Sanders, B.C.: Efficient Quantum Algorithms for Simulating Sparse Hamiltonians. Communications in Mathematical Physics **270**(2), 359–371 (Mar 2007). https://doi.org/10.1007/s00220-006-0150-x
4. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum Amplitude Amplification and Estimation. AMS Contemporary Mathematics Series **305** (Jun 2000). https://doi.org/10.1090/conm/305/05215
5. Carlet, C.: Boolean Functions for Cryptography and Error-Correcting Codes. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 257–397. Cambridge University Press, 1 edn. (Jun 2010). https://doi.org/10.1017/CBO9780511780448.011
6. Chen, Y.A., Gao, X.S.: Quantum Algorithm for Boolean Equation Solving and Quantum Algebraic Attack on Cryptosystems. Journal of Systems Science and Complexity **35**(1), 373–412 (Feb 2022). https://doi.org/10.1007/s11424-020-0028-6
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 (Oct 2018)
8. Childs, A.M., Kothari, R., Somma, R.D.: Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. SIAM Journal on Computing **46**(6), 1920–1950 (Jan 2017). https://doi.org/10.1137/16M1087072
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 **7**, 1069 (Jul 2023). https://doi.org/10.22331/q-2023-07-26-1069
10. Gao, J., Li, H., Wang, B., Li, X.: AES-128 under HHL algorithm. Quant. Inf. Comput. **22**(3-4), 0209–0240 (2022). https://doi.org/10.26421/QIC22.3-4-2
11. Grinko, D., Gacon, J., Zoufal, C., Woerner, S.: Iterative Quantum Amplitude Estimation. npj Quantum Information **7**(1), 52 (Mar 2021). https://doi.org/10.1038/s41534-021-00379-1
12. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for solving linear systems of equations. Physical Review Letters **103**(15), 150502 (Oct 2009). https://doi.org/10.1103/PhysRevLett.103.150502
13. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: Goos, G., Hartmanis, J., Van Leeuwen, J., Buhler, J.P. (eds.) Algorithmic Number Theory, vol. 1423, pp. 267–288. Springer Berlin Heidelberg, Berlin, Heidelberg (1998). https://doi.org/10.1007/BFb0054868
14. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking Symmetric Cryptosystems Using Quantum Period Finding. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016. pp. 207–237. Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_8
15. Kuwakado, H., Morii, M.: Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In: 2010 IEEE International Symposium on Information Theory. pp. 2682–2685 (Jun 2010). https://doi.org/10.1109/ISIT.2010.5513654
16. Liu, W., Gao, J.: Quantum security of Grain-128/Grain-128a stream cipher against HHL algorithm. Quantum Information Processing **20**(10), 343 (Oct 2021). https://doi.org/10.1007/s11128-021-03275-x
17. Murphy, S., Robshaw, M.J.: Essential Algebraic Structure within the AES. In: Goos, G., Hartmanis, J., Van Leeuwen, J., Yung, M. (eds.) Advances in Cryptology — CRYPTO 2002, vol. 2442, pp. 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_1
18. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition (Dec 2010). https://doi.org/10.1017/CBO9780511976667
19. O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press, Cambridge (2014). https://doi.org/10.1017/CBO9781139814782
20. Pieprzyk, J., Wang, H., Zhang, X.m.: Möbius transforms, coincident Boolean functions and non-coincidence property of Boolean functions. Int. J. Comput. Math. **88**, 1398–1416 (May 2011). https://doi.org/10.1080/00207160.2010.509428
21. Suzuki, Y., Uno, S., Raymond, R., Tanaka, T., Onodera, T., Yamamoto, N.: Amplitude estimation without phase estimation. Quantum Information Processing **19**(2), 75 (Jan 2020). https://doi.org/10.1007/s11128-019-2565-2
22. Valiant, L.G., Vazirani, V.V.: NP is as easy as detecting unique solutions. Theoretical Computer Science **47**, 85–93 (Jan 1986). https://doi.org/10.1016/0304-3975(86)90135-0
23. Vazquez, A.C., Woerner, S.: Efficient State Preparation for Quantum Amplitude Estimation. Physical Review Applied **15**(3), 034027 (Mar 2021). https://doi.org/10.1103/PhysRevApplied.15.034027