Quantum Information and Computation, Vol. 22, No. 3&4 (2022) 0209–0240

QUANTUM SECURITY OF AES-128 UNDER HHL ALGORITHM

JUNTAO GAO HAO LI BAOCANG WANG
State Key Laboratory of Integrated Services Networks, Xidian University,Xi'an, 710071, Shaanxi, China

XUELIAN LI
School of Mathematics and Statistics, Xidian University, Xi'an, 710126 Shaanxi, China

Received September 18, 2021
Revised January 1, 2022

There have been a lot of researches about algebric analysis of AES. In this paper, we turned to quantum algorithm to analyze security of AES-128 against the modified HHL algorithm, which is a quantum algorithm used to get classical solutions of multivariate equation system. We constructed two types of equation systems of AES, and solved them with several variants of HHL algorithms respectively. The resulting complexities involved the condition number are given. We analyzed the reasons for the different complexity of the two equation systems and their solution methods, and pointed out that the combination of the boolean equation system and (the improved) BoolSol is more threatening to AES. With a lower bound on the condition number presented by Ding et al., we show that, for AES-128, HHL algorithm is difficult to achieve better attack effect than Grover algorithm. Our results have some enlightening significance for analyzing the post-quantum security of AES-like block ciphers.

Keywords: Modified HHL algorithm, Macaulay system, AES, Algebraic attack

1 Introduction

Rijndael, which is a family of ciphers with different key lengths and block sizes, was chosen to be Advanced Encryption Standard(AES) by NIST in 2001[1]. As an extension of AES, Big Encryption System(BES) was proposed in 2002. Through a bijective mapping, AES can be embeded into BES. The algebraic equations of BES are defined in the same finite field.

The algebraic attack can be traced back to 1949[2]. It aims to get the secret key by solving an equation system derived from the encryption scheme. Compared with other cryptanalysis methods, it only needs a few plaintext and ciphertext pairs corresponding to the key. However, most of the equations established are not linear. We need to transform these nonlinear equations into linear equations. The basic linearization techniques requires about $n^2/2$ equations, where n is the number of variables. In [3], Kipnis and Shamir announced an improved linearization technique, called relinearization technique, to break HFE(Hidden Field Equations) by solving the multiple-variable quadratic equation system. For sufficiently overdefined systems, they conjectured that the algorithm has a polynomial time complexity, and the number of equations required is about $n^2/10$. In [4], Courtois, Klimov, Patarin, and Shamir introduced an improved version of relinearization technique, called the eXtended Linearization(XL) algorithm. Courtois and Pieprzyk analyzed the overdefined system and

proposed the eXtended Sparse Linearization(XSL) attack on Rijindeal[5], however, there exists too many assumptions and premises in the attacking process, which makes its effectiveness have been questioned[6]. As for the equation system describing BES, both linearization and relinearization require too many equations to realize. The complexity of XL algorithm to solve the equation system is much larger than that of the brute force attack[6].

Chen and Gao proposed two quantum algorithms based on HHL quantum algorithm, one of which is called BoolSol for solving boolean equation systems[7] and another of which is called FSol for solving equation systems over finite fields[8]. The BoolSol algorithm is used to analyze the boolean equation system derived from AES-128 and the derived complexity is $O(2^{73.30}\kappa^2)$, where κ is the condition number of coefficient matrix in the linear equation system. In [9], Ding et al. improved the BoolSol algorithm by transforming the measurement of quantum states outputed by HHL algorithm into the coupon collector problem. The Macaulay system is changed into a boolean Macaulay system whose solving degree is only 1. Besides, Ding et al. gave a lower bound on the condition number of the Macaulay system and boolean Macaulay system.

In this paper, we firstly show that, in the equation system built by Chen and Gao[7], the shiftrow procedure in the final round is missed and the sparseness of their second group of equations is not accurate due to the lack of equations, which leads to the inaccurate result. we reviewed the two improved quantum algorithms, BoolSol and Fsol, proposed by Chen and Gao in a clear and concise way, and built two equation systems based on the literatures [10] and [11]. The first one is a boolean equation system based on the result in [10], and the second one is an equation system over $GF(2^8)$[11]. We obtained the number of equations, the number of variables and sparseness of two equation systems by analyzing the inner structure of the system and simulating the matrices of linear transformations in each round. For the boolean equation system, the complexity of attacking 10-round AES-128 using BoolSol is $O(2^{72.98}\kappa^2)$, and for the equation system over $GF(2^8)$, the complexity of attacking 10-round AES-128 using Fsol is $O(2^{89.96}\kappa^2)$. Furthermore, we analyzed the complexity of AES-128 against the improved BoolSol proposed in [9] by considering the boolean Macaulay system and the corresponding lower bound on the condition number. The result illustrates that the complexity of attacking AES-128 using HHL algorithm is larger than that using Grover algorithm, that is, the effect of HHL algorithm attacking AES-128 will not be better than that of Grover algorithm attacking AES-128. We further analyzed the complexity of the classical and quantum algorithms for solving the condition number of a given matrix. It is shown that it is difficult to determine the condition number when the matrix size is very large even we have the quantum computation capability.

Our paper is organized as follows. In Section 2, two types of equation systems based on the literatures [10] and [11] are introduced. We showed that, in the equation system built by Chen and Gao, the shiftrow procedure in the final round of AES is missed and consequently the sparseness of their second group of equations is not accurate. Section 3 introduces the modified HHL algorithms. The scale of equation system derived from BES and complexity of modified HHL algorithm are analyzed in Section 4. In Section 5, we get the complexities of attacking AES-128 by using three modified HHL algorithms. In Section 6, we analyze the classical and quantum algorithms to calculate the condition number and reviewed the conntributions of Ding et al.. We show that, for the matrix derived from the equation system,

it is difficult to determine it's condition number. Section 7 concludes this paper.

2 Two Equation Systems

In order to simplify the algebraic analysis of AES, Murphy, Sean and Robshaw[11] proposed BES, which changes the affine transformation from the finite field $GF(2)$ to $GF(2^8)$ using polynomial interpolation. Then by introducing a bijective mapping, all the operations of the algebraic equation system of BES can be defined over the finite field $GF(2^8)$. Ferguson, Schroeppel and Whiting showed how to present Rijndael using equations over $GF(2^8)$ in 2001[12]. In 2002, the equation system over $GF(2)$ was proposed[5]. In 2016, Dubois and Fliol introduced a new method for establishing equations of AES over $GF(2)$ by using the truth table and Mobius transform[10]. Compared to the equation system of AES, the system of BES over $GF(2^8)$ is sparser and more regular[11], which is suitable for the HHL quantum algorithm analysis[13]. To accurately evaluate the parameters of the equation system and the final complexity, we give a brief description of the BES as follows[11].

2.1 Overview of BES

Define a mapping ϕ from $GF(2^8)$ to a subset of $GF(2^8)^8$, i.e.,

$$\tilde{\mathbf{a}} = \phi(a) = \left(a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5}, a^{2^6}, a^{2^7}\right),$$

and consider $\phi^{-1} : Im(\phi) \to GF(2^8)$ as an extraction mapping which recovers the first component in the vector conjugate. Obviously, ϕ is a bijective mapping. Using this mapping, both the internal state and subkeys of AES can be mapped into BES, so we can define vector spaces of AES and BES as $\mathbf{A} = \mathbf{F}^{16}$ and $\mathbf{B} = \mathbf{F}^{128}(\mathbf{F} = GF(2^8))$ respectively, where 16 and 128 are the number of bytes in vectors and every byte can be represented as a number in $GF(2^8)$.

The operations in BES are as follows.

(i) **Inversion**. For any nonzero element $\mathbf{b} \in \mathbf{B}$, define $\mathbf{b} \to \mathbf{b}^{-1}$.

(ii) **ShiftRow**. This process can be considered as a permutation of the components in a column vector $\mathbf{a} \in \mathbf{A}$. It can be represented as a multiplication of the state vector $\mathbf{a} \in \mathbf{A}$ by a 16×16 matrix $\mathbf{R_A}$. Consider the property of the embedded mapping, it is straightforward to represent this operation in BES as a multiplication of the state vector $\mathbf{b} \in \mathbf{B}$ by a 128×128 matrix $\mathbf{R_B}$, and we can easily get $\mathbf{R_B}$ from $\mathbf{R_A}$.

(iii) **MixColumn**. In AES, this step is defined by a 4×4 matrix $\mathbf{C_A}$, we rewrite it as

$$\begin{bmatrix} \theta & (\theta+1) & 1 & 1 \\ 1 & \theta & (\theta+1) & 1 \\ 1 & 1 & \theta & (\theta+1) \\ (\theta+1) & 1 & 1 & \theta \end{bmatrix} \tag{1}$$

where $\theta = \mathbf{0x}2$. So when the representation of the internal state in AES is changed into a column vector, we need to transform $\mathbf{C_A}$ into $\mathbf{Mix_A} = Diag_4(\mathbf{C_A})$. Now, consider

the property of the embedded mapping again, we can get

$$\mathbf{C_B}^{(k)} = \begin{bmatrix} \theta^{2^k} & (\theta+1)^{2^k} & 1 & 1 \\ 1 & \theta^{2^k} & (\theta+1)^{2^k} & 1 \\ 1 & 1 & \theta^{2^k} & (\theta+1)^{2^k} \\ (\theta+1)^{2^k} & 1 & 1 & \theta^{2^k} \end{bmatrix}$$

where $k = 0, \cdots, 7$. Obviously, $\mathbf{C_B}^{(0)} = C_A$ and we get

$$\mathbf{Mix_B} = Diag(Diag_4(\mathbf{C_B}^{(0)}), \cdots, Diag_4(\mathbf{C_B}^{(7)})).$$

(iv) **AddRoundKey**. This step in BES is identical to that in AES, i.e., if we have the state vector and subkey of AES and BES which are denoted by

$$\mathbf{a} \in \mathbf{A}, (\mathbf{k_A})_i \in \mathbf{A}$$

$$\mathbf{b} \in \mathbf{B}, (\mathbf{k_B})_i \in \mathbf{B},$$

then, this step can be represented by

$$\mathbf{a} \to \mathbf{a} + (\mathbf{k_A})_i, \mathbf{b} \to \mathbf{b} + (\mathbf{k_B})_i.$$

(v) **KeySchedule**. In the key shcedule of AES, a 16-byte AES key $\mathbf{k_A}$ generates ten subkeys, each of which is in $\mathbf{A}$. So is BES. The key schedule procedure uses the same operations as the encryption procedure, so we can realize key schedule of BES using technologies introduced above. In this way, we have $\mathbf{k_B} = \phi(\mathbf{k_A})$, then $(\mathbf{k_B})_i = (\phi(\mathbf{k_A}))_i$ for every round subkey, so the embedded images of an AES subkey sequence form a BES subkey sequence.

2.1.1 Affine Transformation

In AES, this step is defined as a matrix $\mathbf{L_A}$. We define the mapping ψ as

$$\psi : GF(2^8) \to GF(2)^8.$$

Then the affine transformation can be represented as

$$f(a) = \psi^{-1}(\mathbf{L_A}(\psi(a))).$$

For the attacker, the maps ψ and ψ^{-1} make the analysis of AES complicated.

In order to bypass the two mappings and simplify the analysis of AES, the polynomial interpolation is introduced as follows.

$$f(a) = \sum_{k=0}^{7} \lambda_k a^{2^k} \; for \, a \in \mathbf{F}$$

where $(\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7) = (05, 09, f9, 25, f4, 01, b5, 8f)$[11]. Using this technique, the affine transformation of BES can be represented as an 8×8 matrix for every byte, that

is,

$$\mathbf{L_B} = \begin{bmatrix} (\lambda_0)^{2^0} & (\lambda_1)^{2^0} & (\lambda_2)^{2^0} & (\lambda_3)^{2^0} & (\lambda_4)^{2^0} & (\lambda_5)^{2^0} & (\lambda_6)^{2^0} & (\lambda_7)^{2^0} \\ (\lambda_7)^{2^1} & (\lambda_0)^{2^1} & (\lambda_1)^{2^1} & (\lambda_2)^{2^1} & (\lambda_3)^{2^1} & (\lambda_4)^{2^1} & (\lambda_5)^{2^1} & (\lambda_6)^{2^1} \\ (\lambda_6)^{2^2} & (\lambda_7)^{2^2} & (\lambda_0)^{2^2} & (\lambda_1)^{2^2} & (\lambda_2)^{2^2} & (\lambda_3)^{2^2} & (\lambda_4)^{2^2} & (\lambda_5)^{2^2} \\ (\lambda_5)^{2^3} & (\lambda_6)^{2^3} & (\lambda_7)^{2^3} & (\lambda_0)^{2^3} & (\lambda_1)^{2^3} & (\lambda_2)^{2^3} & (\lambda_3)^{2^3} & (\lambda_4)^{2^3} \\ (\lambda_4)^{2^4} & (\lambda_5)^{2^4} & (\lambda_6)^{2^4} & (\lambda_7)^{2^4} & (\lambda_0)^{2^4} & (\lambda_1)^{2^4} & (\lambda_2)^{2^4} & (\lambda_3)^{2^4} \\ (\lambda_3)^{2^5} & (\lambda_4)^{2^5} & (\lambda_5)^{2^5} & (\lambda_6)^{2^5} & (\lambda_7)^{2^5} & (\lambda_0)^{2^5} & (\lambda_1)^{2^5} & (\lambda_2)^{2^5} \\ (\lambda_2)^{2^6} & (\lambda_3)^{2^6} & (\lambda_4)^{2^6} & (\lambda_5)^{2^6} & (\lambda_6)^{2^6} & (\lambda_7)^{2^6} & (\lambda_0)^{2^6} & (\lambda_1)^{2^6} \\ (\lambda_1)^{2^7} & (\lambda_2)^{2^7} & (\lambda_3)^{2^7} & (\lambda_4)^{2^7} & (\lambda_5)^{2^7} & (\lambda_6)^{2^7} & (\lambda_7)^{2^7} & (\lambda_0)^{2^7} \end{bmatrix} \tag{2}$$

Then we can represent the affine transformation as a 128×128 matrix $\mathbf{Lin_B}$ in BES, where $\mathbf{Lin_B} = Diag_{16}(\mathbf{L_B})$.

2.1.2 Round Function of BES

By denoting the state and subkey of BES as $\mathbf{b} \in \mathbf{B}$ and $(\mathbf{k_B})_i \in \mathbf{B}$ respectively, we can represent the round function of BES as

$$\begin{aligned} Round_B(\mathbf{b}, (\mathbf{k_B})_i) &= \mathbf{Mix_B}(\mathbf{R_B}(\mathbf{Lin_B}(\mathbf{b}^{(-1)}))) + (\mathbf{k_B})_i \\ &= \mathbf{M_B} \cdot (\mathbf{b}^{(-1)}) + (\mathbf{k_B})_i \end{aligned} \tag{3}$$

where $\mathbf{M_B}$ is a 128×128 matrix for performing linear diffusion within the BES. Furthermore, if we denote the state and subkey of AES as $\mathbf{a} \in \mathbf{A}$ and $(\mathbf{k_A})_i \in \mathbf{A}$, then we have

$$Round(\mathbf{a}, (\mathbf{k_A})_i) = \phi^{-1}(Round_B(\phi(\mathbf{a}), \phi((\mathbf{k_A})_i))).$$

2.2 Multivariate Quadratic System

We denote the plaintext and the corresponding ciphertext by $\mathbf{p} \in \mathbf{B}$ and $\mathbf{c} \in \mathbf{B}$ respectively, and denote the state vectors before and after the i^{th} invocation of the inversion of S box by $\mathbf{w}_i \in \mathbf{B}$ and $\mathbf{x}_i \in \mathbf{B}$ respectively. From the structure of BES, we can describe BES as the following equation system

$$\begin{cases} \mathbf{w}_0 = \mathbf{p} + \mathbf{k}_0 & \\ \mathbf{x}_i = \mathbf{w}_i^{(-1)} & 0 \le i \le 9 \\ \mathbf{w}_i = \mathbf{M_B}\mathbf{x}_{i-1} + \mathbf{k}_i & 1 \le i \le 9 \\ \mathbf{c} = \mathbf{M_B}^*\mathbf{x}_9 + \mathbf{k}_{10} & \end{cases} \tag{4}$$

Because there is no MixColumn operation in the final round, we get $\mathbf{M_B}^* = \mathbf{R_B} \cdot \mathbf{Lin_B} = \mathbf{M_B} \cdot \mathbf{Mix_B}^{-1}$.

To make it clearer, we denote the $(8j+m)^{th}$ component of $\mathbf{x}_i, \mathbf{w}_i$ and $\mathbf{k}_i$ by $x_{i,(j,m)}, w_{i,(j,m)}$ and $k_{i,(j,m)}$ respectively. From the structure of AES and the property of the embedded mapping, we can easily see that $0 \le j \le 15$ and $0 \le m \le 7$, and rewrite the system as

$$\begin{cases} w_{0,(j,m)} = p_{j,m} + k_{0,(j,m)} & \\ x_{i,(j,m)} = w_{i,(j,m)}^{(-1)} & 0 \le i \le 9 \\ w_{i,(j,m)} = (\mathbf{M_B}\mathbf{x}_{i-1})_{j,m} + k_{i,(j,m)} & 1 \le i \le 9 \\ c_{j,m} = (\mathbf{M_B}^*\mathbf{x}_9)_{j,m} + k_{10,(j,m)} & \end{cases}$$

Furthermore, if we assume 0-inversion does not occur during the encryption and key schedule procedure, this assumption is true for 53% of encryptions and 85% of 128 bit keys[11]. In this case we can rewrite $x_{i,(j,m)} = w_{i,(j,m)}^{(-1)}$ as $x_{i,(j,m)}w_{i,(j,m)} = 1$. Now we rewrite the equation system as

$$\begin{cases} 0 = w_{0,(j,m)} + p_{j,m} + k_{0,(j,m)} & \\ 0 = x_{i,(j,m)}w_{i,(j,m)} + 1 & 0 \le i \le 9 \\ 0 = w_{i,(j,m)} + (\mathbf{M_B}\mathbf{x}_{i-1})_{j,m} + k_{i,(j,m)} & 1 \le i \le 9 \\ 0 = c_{j,m} + (\mathbf{M_B}^*\mathbf{x}_9)_{j,m} + k_{10,(j,m)} & \end{cases}$$

The $\mathbf{M_B}$ and $\mathbf{M_B}^*$ can be decomposed into column vectors as (α) and (β), then we get

$$\begin{cases} 0 = w_{0,(j,m)} + p_{j,m} + k_{0,(j,m)} & \\ 0 = x_{i,(j,m)}w_{i,(j,m)} + 1 & 0 \le i \le 9 \\ 0 = w_{i,(j,m)} + \sum_{j',m'} \alpha_{(j,m),(j',m')}x_{i-1,(j',m')} & 1 \le i \le 9 \\ 0 = c_{j,m} + k_{10,(j,m)} + \sum_{(j',m')} \beta_{(j,m),(j',m')}x_{9,(j',m')} & \end{cases}$$

Finally, using the property of embedded mapping, we can get more equations, and classify the system by writing linear and quadratic equations separately,i.e.,for $1 \le i \le 9$, we have

$$\begin{cases} 0 = w_{0,(j,m)} + p_{j,m} + k_{0,(j,m)} \\ 0 = w_{i,(j,m)} + \sum_{j',m'} \alpha_{(j,m),(j',m')}x_{i-1,(j',m')} \\ 0 = c_{j,m} + k_{10,(j,m)} + \sum_{(j',m')} \beta_{(j,m),(j',m')}x_{9,(j',m')} \\ \\ 0 = x_{i,(j,m)}w_{i,(j,m)} + 1 \\ 0 = x_{i,(j,m)}^2 + x_{i,(j,m+1)} \\ 0 = w_{i,(j,m)}^2 + w_{i,(j,m+1)} \end{cases} \tag{5}$$

The BES encryption procedure can therefore be described as an overdetermined multivariate quadratic systems and the problem of attacking AES can be transformed into the problem of solving such an equation system of BES.

Chen and Gao also established the equation system used to describe AES based on [11].

The equation system they established is as follows to describe AES-(N_k, N_r)[7]:

$$0 = w_{0,(j,m)} + p_{j,m} + x_{0,(j,m)} \tag{6}$$

$$0 = x_{i,(j,m)} + w_{i,(j,m)} + \sum_{j',m'} \alpha_{(j,m),(j',m')} x_{i-1,(j',m')} \quad for \quad i = 1, \cdots, N_r - 1 \tag{6a}$$

$$0 = c_{(j,m)} + w_{N_r,(j,m)} + y_{N_r-1}(5j \mod 16, m) \tag{6b}$$

$$0 = \mathcal{S}(x_{i,(j,0)}, \cdots, x_{i,(j,7)}, y_{i,(j,0)}, \cdots, y_{i,(j,7)}) \quad for \quad i = 1, \cdots, N_r - 1 \tag{6c}$$

$$0 = \mathcal{S}(w_{i,(\overline{j},0)}, \cdots, w_{i,(\overline{j},7)}, \overline{w}_{i,(\overline{j},0)}, \cdots, \overline{w}_{i,(\overline{j},7)}) \quad for\, \overline{j} = 4N_k - 4, \cdots, 4N_k - 1$$

$$0 = w_i(\overline{j}, m) + w_{i-1}(\overline{j}, m) + \overline{w}_{i-1}(\overline{j} + 13, m) + \chi(m_i) \quad for\, \overline{j} = 0, 1, 2$$

$$0 = w_i(3, m) + w_{i-1}(3, m) + \overline{w}_{i-1}(12, m) + \chi(m_i)$$

$$for\, N_k \leq 6$$

$$0 = w_i(\overline{j}, m) + w_{i-1}(\overline{j}, m) + w_i(\overline{j} - 4, m) \quad for\, \overline{j} = 4, 5, \cdots, 4N_k - 1$$

$$for\, N_k > 6$$

$$0 = \mathcal{S}(w_{i,(\overline{j},0)}, \cdots, w_{i,(\overline{j},7)}, \overline{w}_{i,(\overline{j},0)}, \cdots, \overline{w}_{i,(\overline{j},7)}) \quad for\, \overline{j} = 12, 13, 14, 15$$

$$0 = w_i(\overline{j}, m) + w_{i-1}(\overline{j}, m) + \overline{w}_i(\overline{j} - 4, m) \quad for\, \overline{j} = 16, 17, 18, 19$$

$$0 = w_i(\overline{j}, m) + w_{i-1}(\overline{j}, m) + w_i(\overline{j} - 4, m) \quad for\, \overline{j} = 4, \cdots, 15, 20, \cdots, 4N_k - 1$$

where j runs from 0 to $4N_k - 1$, and m runs from 0 to 7. $\overline{w}_{i,(j,m)}$, $x_{i,(j,m)}$ and $y_{i,(j,m)}$ are state variables. $w_{i,(j,m)}$ are key variables, and $\mathcal{S}$ is a set of 39 equations in $\mathbb{F}_2[x_0, \cdots, x_7, y_0, \cdots, y_7]$ represents the Rijndeal S-box. χ is the round constant[7].

We take AES-128 as an example to analyze the encryption part of this equation system, where $N_k = 4$ and $N_r = 10$. The first group of equations (6) denotes the initial addroundkey. The second group of equations (6a) denotes the shiftrow, mixcolumn and addroundkey from the first round to the ninth round. the third group of equations (6b) denotes the final addroundkey procedure and the fourth equation (6c) denotes all the subbytes in the encryption part.

We think that there are two flaws in the encryption part. Firstly, the shiftrow procedure in the final round is missing. Secondly, Chen et al. said that in the second group of equations (6a), exactly 640 of $\alpha(j, m, j', m')$ are 1 for given i[7], however, our analysis shows that the actual sparseness of this part is 732, the details can be found in Appendix.

We further established a boolean equation system according to [10]. Details of equation system and calculation process can be found in Appendix. We transformed and solved the boolean equation system by using Boolsol, and illustrated that the complexity of attacking 10-round AES-128 is $O(2^{72.98}\kappa^2)$. In addition, we analyzed the complexity of attacking AES-128 using the combination of boolean Macaulay system and improved BoolSol proposed by Ding et al., which is $O(poly(5152)\kappa(\tilde{\mathcal{B}}))$. However, for the equation system based on BES over $GF(2^8)$, we solved it by using FSol, and the complexity of attacking 10-round AES-128 is $O(2^{89.96}\kappa^2)$. We analyzed the reasons for the the difference in complexity caused by the two methods in Section 5. Our results will help to more accurately evaluate the ability of AES-128 to resist HHL quantum algebraic attacks.

3 Macaulay System and Modified HHL algorithm

A nonlinear equation system to describe a stream cipher or a block cipher can be transformed into a linear equation system. Compared with the classical algorithm, the speed of HHL quantum algorithm in solving linear equation system may be exponentially improved. This section reviews a general technique which is used to transform a nonlinear equation system into a Macaulay linear equation system, then we review the HHL quantum algorithm and its modified versions.

The original HHL algorithm requires a quantum state $|b\rangle$ as input, which greatly reduces the application scope. Chen and Gao gave a method for generating $|b\rangle$ from the vector $\overrightarrow{\mathbf{b}}$ efficiently[14]. Secondly, the original HHL algorithm outputs a quantum state as the result, however, if we want to get the solution in the classical state, we have to apply HHL algorithm for at least N times, where N is the length of solution vector[13]. The modified HHL algorithm solves the two problems. The following subsection comes from [13].

3.1 HHL algorithm

The input of HHL algorithm is a $N \times N$ sparse Hermitian matrix and a unit vector $\overrightarrow{\mathbf{b}}$. We aim to find $\overrightarrow{\mathbf{x}}$ satisfying $\mathbf{A}\overrightarrow{\mathbf{x}} = \overrightarrow{\mathbf{b}}$. The original HHL algorithm is as follows.

(i) Representing $\overrightarrow{b}$ as a quantum state $|b\rangle = \sum_{i=1}^{N} b_i|i\rangle$.

(ii) By using the Hamiltonian simulation, we apply $e^{i\mathbf{A}t}$ to $|b\rangle$ for a superposition of different times t. This ability to exponentiate $\mathbf{A}$ is translated, via the well known technique of phase estimation, into the ability to decompose $|b\rangle$ in the eigenbasis of $\mathbf{A}$ and to find the corresponding eigenvalues λ_j. Furthermore, if we denote $|b\rangle = \sum_{j=1}^{N} \beta_j|u_j\rangle$, where u_j is the eigenbasis of $\mathbf{A}$, then the state of system after this step is close to $\sum_{j=1}^{N} \beta_j|u_j\rangle|\lambda_j\rangle$.

(iii) Performing the linear map which transforms $|\lambda_j\rangle$ into $C\lambda_j^{-1}|\lambda_j\rangle$, where C is a normaling constant. As this operation is not unitary, it has some probability of failing.

(iv) If the last step succeeds, we uncompute the $|\lambda_j\rangle$ register, and the quantum state are left with a state proportional to $\sum_{j=1}^{N} \beta_j\lambda_j^{-1}|u_j\rangle = \mathbf{A}^{-1}|b\rangle = |x\rangle$.

Theorem 1 *[7]*

Given an s sparse matrix $\mathbf{A} \in \mathbb{C}^{M\times N}$ with the condition number κ, singlar values $\lambda_1, \cdots, \lambda_n$, and a unitary quantum state $|b\rangle \in \mathbb{C}^M$. Let $|v_j\rangle(|u_j\rangle)$ be the eigenvectors of $\mathbf{A}^\dagger\mathbf{A}(\mathbf{A}\mathbf{A}^\dagger)$ with respect to the nonzero eigenvalues λ_j^2 of $\mathbf{A}^\dagger\mathbf{A}(\mathbf{A}\mathbf{A}^\dagger)$. Then the singular value decomposition of $\mathbf{A}$ is $\mathbf{A} = \sum_{j=1}^{n} \lambda_j|u_j\rangle\langle v_j|$. For the linear equation system $\mathbf{A}|x\rangle = |b\rangle$, HHL algorithm will give an approximation to the solution state $|x\rangle$ for the following vector

$$\tilde{x} = \sum_{j=1}^{n} |v_j\rangle\langle v_j|b\rangle$$

in time $O(\log(N+M)s\kappa^2/\varepsilon)$ with error bounded by a given $\varepsilon \in (0,1)$.

In the following equations, the log$q$ notation refers to the logarithm in base 2 of q.

3.2 Modified HHL algorithm

The main idea of Chen and Gao's modification[7] is to add equations $\overrightarrow{\mathbf{0}}\mathbf{x} = \overrightarrow{\mathbf{1}}$ and $\overrightarrow{\mathbf{0}}\mathbf{x} = \overrightarrow{\mathbf{0}}$ to the initial system to enlarge both the length of 1 in $\overrightarrow{\mathbf{b}}$ and total length of $\overrightarrow{\mathbf{b}}$ to power of 2,

then $|b\rangle$ can be prepared efficiently[14]. It is obvious that adding $\overrightarrow{\mathbf{0}}\mathbf{x} = \overrightarrow{\mathbf{0}}$ doesn't change the solution. When it comes to $\overrightarrow{\mathbf{0}}\mathbf{x} = \overrightarrow{\mathbf{1}}$, if the added equations are placed at the end of equation system, we have $\mathbf{A}^\dagger\mathbf{A} = (\mathbf{A}')^\dagger\mathbf{A}'$, where $\mathbf{A}'$ is the coefficient matrix of equation system with equations $\overrightarrow{\mathbf{0}}\mathbf{x} = \overrightarrow{\mathbf{1}}$. By the property of the HHL quantum algorithm and $\mathbf{A}^\dagger\mathbf{A} = (\mathbf{A}')^\dagger\mathbf{A}'$, the added equation $\overrightarrow{\mathbf{0}}\mathbf{x} = \overrightarrow{\mathbf{1}}$ will not affect the returned quantum state.

3.3 Macaulay Linear System

As the original HHL algorithm was proposed to solve linear equation system, we firstly give a breif introduction on the Macaulay linear system.

Lexicographic monomial order is used for $x_1 > x_2 > \cdots > x_n$, and let $\mathrm{m}_{\leq d}$ be the set of all monomials which are factors of $x_1^d x_2^d \cdots x_n^d$, then we sort

$$\mathrm{m}_{\leq d} = \{m_{d,0}, m_{d,1}, \cdots, m_{d,(d+1)^n-1}\}$$

in ascending lexicographic monomial ordering. Obviously we have

$$1 + C_n^1 \times d + C_n^2 \times d^2 + \cdots + C_n^n \times d^n = (d+1)^n.$$

Denote the aimed boolean system by $\mathcal{F} = \{f_1, \cdots, f_r\}$, where $d_i = deg(f_i)$ is the degree of each f_i and $t_i = \#f_i$ is the number of terms of each f_i. To establish Macaulay system, we choose a $D \in \mathbb{N}$ with $D \geq max_{i=1}^r d_i$. Let $\overline{d}$ be the minimal integer satisfying $\overline{d} \geq D - min_i d_i$ and $\overline{d} + 1 = 2^\delta$ for certain $\delta \in \mathbb{N}$. Set $\overline{D}$ to be the minimal integer satisfying $\overline{D} \geq D$ and $\overline{D}+1 = 2^\Delta$ for certain $\Delta \in \mathbb{N}$. The parameters $\overline{d}$ and $\overline{D}$ are introduce to make the complexity of BoolSol easier to be simplified in Section 4.3.

For $i = 1, 2, \cdots, r$ and each $m_{\overline{d},j} \in \mathrm{m}_{\leq \overline{d}}$ with $deg(m_{\overline{d},j}) \leq D - d_i$, $m_{\overline{d},j} f_i$ could be considered as a linear function of the monomials in $\mathrm{m}_{\leq \overline{D}}$. For $deg(m_{\overline{d},j}) > D - d_i$, we replace $m_{\overline{d},j} f_i$ by 0. Introduce the following notation:

$$m_{\overline{d},j,i} = \begin{cases} m_{\overline{d},j} & \text{if } deg(m_{\overline{d},j}) \leq D - d_i \\ 0 & \text{if } deg(m_{\overline{d},j}) > D - d_i \end{cases}$$

The zero rows are added so that the Macaulay matrix can be efficiently queried.

Then Macaulay system can be given as follows

$$\begin{array}{c} \\ m_{\overline{d},0,1} f_1 \\ \vdots \\ m_{\overline{d},(\overline{d}+1)^n-1,1} f_r \\ m_{\overline{d},0,2} f_2 \\ \vdots \\ m_{\overline{d},(\overline{d}+1)^n-1,r} f_r \end{array} \begin{array}{c} m_{\overline{D},1} < \cdots < m_{\overline{D},(\overline{D}+1)^n-1} \\ \begin{bmatrix} \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \end{bmatrix} \end{array} \begin{array}{c} m_{\overline{D},0} = 1 \\ \begin{bmatrix} m_{\overline{D},1} \\ m_{\overline{D},2} \\ \vdots \\ m_{\overline{D},(\overline{D}+1)^n-1} \end{bmatrix} \end{array} = \begin{bmatrix} -f_1(0) \\ \vdots \\ -f_r(0) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{7}$$

Denote it as

$$\mathcal{M}_{\mathcal{F},D} m_D = \mathbf{b}_{\mathcal{F},D}$$

By increasing all the orders of f_i in $\mathcal{F}$, we introduce more equations and transform $\mathcal{F}$ into a linear system. Next we discuss the sparseness of Macaulay system.

(i) As we can see, each row in $\mathcal{M}_{\mathcal{F},D}$ consists of the coefficients of the polynomial in $\mathcal{F}$ or that of the polynomials multiplied by a monomial. If a polynomial is multiplied by a monomial, then its sparseness remains unchange. So the upper bound of row sparseness of $\mathcal{M}_{\mathcal{F},D}$ is $max_i t_i$.

(ii) We arrange all monomials in m_D with the lexicographic order and view every polynomial as an vector. Then, multiplying a polynomial by a monomial in m_D is equivalent to shifting the polynomial vector to the right, and the steps of shifting depends on the mulitiplied monomial. Using this technique, we can estimate the upper bound of $\mathcal{M}_{\mathcal{F},D}$'s column sparseness. For a polynomial, the column sparseness achieves the maximum value when all the nonzero coefficients are placed in a same column. Therefore, the upper bound of the column sparseness of $\mathcal{M}_{\mathcal{F},D}$ is given as $T = \sum_{i=1}^{r} t_i$.

It is obvious that the scale of Macaulay matrix is $(r(\overline{d}+1)^n) \times ((\overline{D}+1)^n - 1)$.

The following theorem is used to determine the upper bound of the solving degree $\overline{D} = Sdeg(\mathcal{F})$ for the nonlinear system $\mathcal{F}$.

Theorem 2 *[7] Let $\mathbb{X} = \{x_1, x_2, \cdots, x_n\}$, and I be an ideal in $\mathbb{C}[\mathbb{X}]$ generated by $\mathcal{F} = \{f_1, \cdots, f_r\}$ of degrees $d_1, \cdots, d_r$ such that $d_1 \geq d_2 \geq \cdots \geq d_r$. Choose any graded monomial order. If $\mathcal{F}$ satisfies Lazard's condition[15], then $Sdeg(\mathcal{F}) \leq d_1 + \cdots + d_{n+1} - n + 1$ with $d_{n+1} = 1$ if $r = n$.*

Furthermore, if we denote $d = max_i d_i$, then $Sdeg(\mathcal{F}) \leq (n+1)(d-1)+2$ under Lazard's condition.

Chen and Gao showed that the solving degree is not large enough for monomial solving with the Macaulay and proposed complete solving degree($CSdeg(\mathcal{F})$). They also showed that if we consider the following equation system

$$\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2 \subset \mathbb{C}[\mathbb{X}]$$

where $\mathcal{F}_1$ is the reduced Grobner basis of $\mathcal{F}$ such that $\forall i(lm(f_i) = x_i^{d_i}, d_i \geq 1), \mathcal{F}_2 = f_1, f_2, \cdots, f_r$, we have $CSdeg(\mathcal{F}) \leq d - 2n + 2\sum_{i=1}^{n} d_i$.[7]

3.4 Boolean Macaulay System

Ding et al. proposed a boolean Macaulay System based on Macaulay system[9]. More specifically, if we reorder the monomials elaborately and then perform row operation to simplify the Macaulay matrix, a boolean Macaulay system can be obtained as a submatrix of the simplified result. The following is a concise description.

Firstly, each column in the Macaulay matrix is labeled by a monomial. The columns of Macaulay matrix can be partitioned into the left side part and the right side part. The labels on the right side are multilinear monomials and ordered in ascending order with respect to the integer represented by the exponent vector of the multilinear mononmial, and the labels in the left side are the nonmultilinear monomials and ordered under any monomial order. The resulted Macaulay matrix can be written as

$$\tilde{\mathcal{M}} = \begin{bmatrix} \mathbf{L_1} & \mathbf{R_1} \\ \mathbf{L_2} & \mathbf{R_2} \end{bmatrix}$$

where $\mathbf{L_1}$ and $\mathbf{R_1}$ denote the rows generated by $\mathcal{F}_1$, $\mathbf{L_2}$ and $\mathbf{R_2}$ denote the rows generated by $\mathcal{F}_2$.

Finally, Ding et al. proved that, using row operation, Macaulay matrix can be simplified to

$$\tilde{\mathcal{M}}' = \begin{bmatrix} \mathbf{0} & \tilde{\mathcal{B}} \\ \mathbf{I_2} & \mathbf{B_2} \end{bmatrix}$$

where $\tilde{\mathcal{B}}$ is the boolean Macaulay matrix.

The column of $\tilde{\mathcal{B}}$ is labeled by multilinear monomials and the number of multilinear monomials is 2^n. The number of rows of $\mathcal{M}$ is $r(\overline{d}+1)^n$ and the matrix I_2 is of dimension $(\overline{D}+1)^n-1-2^n$, so the number of rows of $\tilde{\mathcal{B}}$ is $r(\overline{d}+1)^n-(\overline{D}+1)^n+1+2^n = r2^{\sigma n}-2^{\Delta n}+1+2^n$, which is usually greater that the number of columns.

The boolean Macaulay system has an amazing property that its solving degree(max degree) is 1, while the Macaulay system proposed by Chen et al[7]'s solving degree is $3n$, where n is the number of variables.

3.5 A quantum algorithm to find boolean solutions

Chen and Gao proposed a modified HHL algorithm to solve boolean equation systems, and the returned result is a classical solution of the equation system[7]. We call the algorithm BoolSol.

Let $\mathbb{X} = \{x_1, x_2, \cdots, x_n\}$, the BoolSol algorithm takes $\mathcal{F} = \{f_1(\mathbb{X}), \cdots, f_r(\mathbb{X})\}$ with $T = \sum_{i=1}^{r} \#f_i, d_i = deg(f_i)$, and $\varepsilon \in (0,1)$ as input, and the output is a boolean solution $\mathbf{a} \in \mathbb{V}_{\mathbb{C}}(\mathcal{F}, \mathbb{H}_{\mathbb{X}})$ or $\emptyset$ with success probability at least $1-\varepsilon$, where $\mathbb{V}_{\mathbb{C}}(\mathcal{F}, \mathbb{H}_{\mathbb{X}})$ denotes the common zeros of polynomials in $\{\mathcal{F}, \mathbb{H}_{\mathbb{X}}\}$ and $\mathbb{H}_{\mathbb{X}} = \{x_1^2 - x_1, \cdots, x_n^2 - x_n\}$.

(i) Test whether $\overrightarrow{\mathbf{0}}$ and $\overrightarrow{\mathbf{1}}$ are the solutions of the equation system.

(ii) According to the property of boolean variables, we replace x_i^m with x_i for all i and $m \in \mathbb{N}$. The new variable set is written as $\mathbb{Y}$ and the new equation system is denoted by $\mathcal{F}_1$.

(iii) As described in Theorem 2, let $\mathcal{F}_2 = \mathcal{F}_1 \cup \mathbb{H}_{\mathbb{Y}}$ satisfy Lazard's condition, then we will get the upper bound of solving degree, i.e., $D = Sdeg(\mathcal{F}_2)$.

(iv) Denote the Macaulay system by $\mathcal{M}_{\mathcal{F}_2,D}\mathbf{m}_D = \mathbf{b}_{\mathcal{F}_2,D}$, then we apply modified HHL algorithm to obtain a quantum state $|\tilde{m}\rangle$ with the error bound $\sqrt{\varepsilon_1/n}$. When it comes to classical state, the error bound becomes ε_1/n. Later in this algorithm, we can prove that the loop from step 3 to step 9 will run at most n times, so the upper bound of error of step 3 to step 9 is ε_1.

(v) Measure the quantum state $|\tilde{m}\rangle$, then we will get a state $|e_k\rangle$. Because the $\mathcal{F}_2$ is restricted in the Boolean form, from the state $|e_k\rangle$, we can get a classcial state easily by using a unitary operator.

(vi) The quantum state collapses to $|e_k\rangle$ implies that the k_{th} element of m_D is 1. In other words, if we denote the k_{th} element of m_D as $m_k = \prod_{i=1}^{u_k} y_{n_i}$, then for $\mathcal{F}_1 \subset \mathbb{C}[\mathbb{Y}]$ we have $y_{n_i} = 1, i = 1, \cdots, u_k$. So we can get the values of at least one variable in every loop, i.e. the loop from step 3 to step 9 will run at most n times.

(vii) The found solution can be back substituted into $\mathcal{F}_1$, which is simplified, then the variable set is refreshed as $\mathbb{Y} = \mathbb{Y} \backslash \{y_{n_i} | i = 1, \cdots, u_k\}$.

(viii) Because there is an error probability for the returned result of modefied HHL algorithm, we have to give a test for the solution we got. It is enough to test whether $\overrightarrow{\mathbf{1}}$ is the solution of simplified system, because if the solution of the simplified system is $\overrightarrow{\mathbf{1}}$, then

$\overrightarrow{\mathbf{1}}$ is also the solution of the initial system. However, the test in step 1 has indicated that $\overrightarrow{\mathbf{1}}$ is not the solution of initial system. A contradiction appears, then we turn to step 11.

(ix) Test if $\mathcal{F}_1 \neq \emptyset$. Here $\mathcal{F}_1 \neq \emptyset$ means that the system have not been solved completely, then test whether $\mathcal{F}(\overrightarrow{\mathbf{0}}) = \overrightarrow{\mathbf{0}}$, i.e., test whether $\overrightarrow{\mathbf{0}}$ is the solution of simplified system. If neither of the test holds, then we have to go to step 3 to further solve the simplified system.

(x) Return $(a_1, \cdots, a_n)$ where if $y_i \in \mathbb{Y}$, then $a_i = 0$, else $a_i = 1$.

(xi) Let ℓ be the number of loops from step 3 to step 9. If $\lceil \log_{\varepsilon_1} \varepsilon \rceil < \ell$, output $\emptyset$, else let $\ell = \ell + 1$ and turn to step 2.

Remark 1 *We have to test if* $\lceil \log_{\varepsilon_1} \varepsilon \rceil < \ell$ *because* ε_1/n *is the upper error bound of the loop from step 3 to step 9 and* ε *is the upper error bound of entire algorithm, so if* $\mathcal{F}$ *does have boolean solutions, this algorithm return* $\emptyset$ *with probability less than* $1-(1-\varepsilon_1/n)^n < \varepsilon_1$*, which means that the probability that the algorithm runs to step 11 is smaller than* ε_1*. What's more, the loop from step2 to step11 will run at most* $\lceil \log_{\varepsilon_1} \varepsilon \rceil$ *times because* $\varepsilon_1^{\lceil \log_{\varepsilon_1} \varepsilon \rceil} < \varepsilon$*.*

Chen and Gao have proved that $CSdeg(\mathcal{F}_1 \cup \mathbb{C}_{\mathbb{X}}) \leq d - 2n + 2\sum_{i=1}^{n} d_i \leq 3n$. Furthermore, they used two techniques in [7], one of which is used to control the sparseness of equation system, called s-sparse split set, and another of which is used to transform equation system over finite field into that over $\mathbb{C}$.

Using the s-sparse split set technique, we can transform a polynomial with the sparseness t into a polynomial system, in which each polynomial has the sparseness s. This technique can result in a sparse polynomial system which can be suitable for the HHL quantum algorithm.

(i) **s-sparse split set**. For a polynomial $f = \sum_{i=1}^{t} m_i$ with the sparseness t, given a positive integer s, we can define $S_t = \lceil \frac{t-s}{s-2} \rceil$ and introduce new variables $\mathbb{U}_f = \{u_1, u_2, \cdots, u_{S_t}\}$. Then we have

$$S(f,s) = \begin{cases} \{f\} & if\ t \leq s \\ \{\tilde{f}_1, \cdots, \tilde{f}_{S_t+1}\} & otherwise \end{cases}$$

where

$$\begin{cases} \tilde{f}_1 = \sum_{k=1}^{s-1} m_k + u_1 \\ \tilde{f}_j = \sum_{k=(j-1)(s-2)+2}^{j(s-2)+1} m_k + u_{j-1} + u_j \quad j = 2, \cdots, S_t \\ \tilde{f}_{S_t+1} = \sum_{k=S_t(s-2)+2}^{t} m_k + u_{S_t} \end{cases}$$

$S(f,s)$ is called the split set of f. We can see that the upper bound of sparseness of new polynomial generated by f is s. If we denote the number of elements in $\{f\}$ by $\#f$, then we have

$$\#\mathbb{U}(f,s) = S_t = \lceil \frac{t-s}{s-2} \rceil,$$

$$\#S(f,s) = \lceil \frac{t-s}{s-2} \rceil + 1.$$

Apply this technique to all polynomials in $\mathcal{F}$, then

$$S(\mathcal{F},s) = \bigcup_{f \in \mathcal{F}} S(f,s), \text{and } \mathbb{U}(\mathcal{F},s) = \bigcup_{f \in \mathcal{F}} \mathbb{U}(f,s).$$

Furthermore

$$\#S(\mathcal{F},s)=\#\mathcal{F}+\sum_i\lceil\frac{t_i-s}{s-2}\rceil,$$

$$\#(\mathbb{X}\cup\mathbb{U}(\mathcal{F},s))=n+\sum_i\lceil\frac{t_i-s}{s-2}\rceil.$$

(ii) **Transformation**. If all elements in $\mathcal{F}=\{f_1,\cdots,f_r\}$ are boolean polynomials with $\#f_i=t_i$, let

$$F_i=\prod_{k=f_i(0)}^{\lfloor t_i/2\rfloor}(f_i-2k)$$

$$C(\mathcal{F})=\{F_1,\cdots,F_r\}\cup\mathbb{H}_{\mathbb{X}}$$

we can see that there is a bijective mapping between $\mathbb{V}_{\mathbb{F}_2}(\mathcal{F})$ and $\mathbb{V}_{\mathbb{C}}(C(\mathcal{F}))$, so this technique can transform the system over finite fields into that over $\mathbb{C}$.

3.6 Solving MQ system over finite field

It is well known that, by introducing new variables, a polynomial system over finite fields can be represented as an MQ systems, so we only discuss how to solve the MQ system over finite fields by using the modified HHL algorithm. A transformation procedure is given in [8]. For completeness, we give a brief description as follows.

Firstly,we transform the MQ system into a new Boolean system $B(\mathcal{F})$ by rewriting every variable in MQ system as its binary representation. Because every variable is defined over $GF(p)$ and the system is an MQ system,we have $\#B(\mathcal{F})=n\log p$ and $T_{B(\mathcal{F})}=O(T_{\mathcal{F}}(\log p)^2)$ where $T_{\mathcal{F}}$ denotes the sparseness of MQ system.

Then, we use the s-sparse split set technique to control the sparseness of $B(\mathcal{F})$ and transform the new variables in $\mathbb{U}$ into their binary representations. Denote the resulted system by $S_{bit}(B(\mathcal{F}))$, we have $\#S_{bit}(B(\mathcal{F}))=O(n\log p+T_{\mathcal{F}}(\log p)^3)$ and $T_{S_{bit}(B(\mathcal{F}))}=O(T_{\mathcal{F}}(\log p)^3)$.

Finally, since $S_{bit}(B(\mathcal{F}))$ is defined over $GF(p)$, we need to transform $S_{bit}(B(\mathcal{F}))$ into a system over $\mathbb{C}$ in order to apply BoolSol. Denote the final system by $P(\mathcal{F})$, then we have $\#P(\mathcal{F})=O(n\log p+T_{\mathcal{F}}(\log p)^3)$ and $T_{P(\mathcal{F})}=O(T_{\mathcal{F}}(\log p)^5)$.

The quantum algorithm to solve polynomial systems over F_p is given as follows[8].

Input. $\mathcal{F}=\{f_1,\cdots,f_r\}$ and error bound $\varepsilon\in(0,1)$.

Output. A solution $\mathbf{a}\in V_{F_p}(\mathcal{F})$ or $\emptyset$.

(i) Using the procedure described above, we transform the MQ system over $GF(p)$ into boolean systems over $\mathbb{C}$, that is,

$$\mathcal{F}_1=P(Q(\mathcal{F}))\subset C[X_{bit},V_{bit},U_{bit}].$$

(ii) Because the solution of $\mathcal{F}_1$ does exist,we find the solution $\overrightarrow{\mathbf{b}}$ of $\mathcal{F}_1$ by using the quantum algorithm BoolSol, that is,

$$\overrightarrow{\mathbf{b}}=BoolSol(\mathcal{F}_1,\varepsilon).$$

(iii) By expressing the $\overrightarrow{\mathbf{b}}$ as follows,

$$\overrightarrow{\mathbf{b}}=(\hat{X}_{bit},\hat{V}_{bit},\hat{U}_{bit})$$

where

$$\hat{X}_{bit} = (\hat{x}_{10}, \cdots, \hat{x}_{1\lfloor \log p \rfloor}, \hat{x}_{20}, \cdots, \hat{x}_{n\lfloor \log_p \rfloor})$$

then the solution of the initial MQ system is

$$(\sum_{k=0}^{\lfloor \log p \rfloor} \hat{x}_{1k} 2^k, \cdots, \sum_{k=0}^{\lfloor \log p \rfloor} \hat{x}_{nk} 2^k) \, mod \, p$$

In this algorithm, the MQ system over $GF(p)$ is firstly transformed into the boolean system over $\mathbb{C}$, then the BoolSol is used to find a solution for the boolean system. Finally, the boolean solution is transformed into a solution over $GF(p)$. In the rest of this paper, we call this algorithm FSol.

3.7 Improved BoolSol by Ding et al.

In section 3.5 and 3.6 we reviewed two variants of HHL algorithm to solve Macaulay system. In this section, we introduce a new variant of HHL algorithm proposed by Ding et al. which is used to solve boolean Macaulay system.

Ding et al. thinks that the measurement of quantum states outputed by HHL algorithm can be regared as the coupon collector problem. More precisely, if $S \subset \mathbb{X}$ is a set containing all the variables whose solutions are all equal to 1, and S_d be the set containing all nonempty subsets of S of size at most d. When applying the HHL algorithm to the boolean Macaulay system, the output quantum state can be represented as $|x\rangle = \frac{1}{\sqrt{|S_d|}} \sum_{R \in S_d} |R\rangle$.

Ding et al. proved that, after measuring $O((|S|/d) \log(|S|/\varepsilon))$ copies of quantum superposition state, the set S can be computed with probability at least $1 - \varepsilon$[9]. And if $\frac{|S|}{k} \leq d \leq n$, where k is a positive integer, only $O(\log |S|)$ copies is needed.

However, in actual application, we have no information about $|S|$, so the upper bound $O(\log n)$ will be used because $n \geq |S|$. Their algorithm is as follows[9]:

(i) Apply HHL algorithm to the boolean Macaulay system $\tilde{\mathcal{B}}\overrightarrow{\mathbf{x}} = \overrightarrow{\mathbf{b}}$ of total degree n(the number of variables) and get the solution in quantum state

$$|x\rangle = \frac{1}{\sqrt{|S_d|}} \sum_{R \in S_d} |R\rangle$$

(ii) Perform measurement on the quantum state and get outcome $|R\rangle$, then let all the variables in the set R be 1.

(iii) Repeat step1 and step2 $O(\log n)$ times, and then set all left remaining variables to 0.

(iv) Return the solution.

In consequence, Ding et al. proved that the complexity of their variant of HHL algorithm is $O(poly(n)\kappa(\tilde{\mathcal{B}}))$, where $\kappa(\tilde{\mathcal{B}})$ denotes the condition number of the matrix $\tilde{\mathcal{B}}$.

4 Complexity analysis

We will analyze the parameters such as the number of equations and the number of variables of BES equation system. Furthurmore, we will use the same technique to analyze the system including key schedule. The result will serve for the complexity analysis of solving this equation systems using modified HHL algorithm.

4.1 Scale of equation system

To estimate the attack complexity, we have to consider the number of equations and variables for the n-th round of AES. The analysis will be decomposed into two parts: the encryption part and the key schedule part.

(i) **The encryption part**. In equation (4), all of the variables appeared in the encryption part are $\mathbf{w}_i, \mathbf{x}_i$ and $\mathbf{k}_i$, and the numbers of each of which are shown in the following Table 1.

Table 1. Number of Variables

variable	number
$\mathbf{w}_i$	$128N_r$
$\mathbf{x}_i$	$128N_r$
$\mathbf{k}_i$	$128(N_r+1)$

The equation (5) is obtained by expanding equation (4) and is added some equations by considering the conjugate property. Since the added equations do not introduce new variables, the number of variables of two equations are equal.
Similarly, the numbers of different types of equations are shown in Table 2.

Table 2. Number of Equations

equation	number
$0 = w_{0,(j,m)} + p_{j,m} + k_{0,(j,m)}$	128
$0 = w_{i,(j,m)} + \sum_{j',m'} \alpha_{(j,m),(j',m')} x_{i-1,(j',m')}$	$128(N_r-1)$
$0 = c_{j,m} + k_{10,(j,m)} + \sum_{(j',m')} \beta_{(j,m),(j',m')} x_{9,(j',m')}$	128
$0 = x_{i,(j,m)} w_{i,(j,m)} + 1$	$128N_r$
$0 = x_{i,(j,m)}^2 + x_{i,(j,m+1)}$	$128N_r$
$0 = w_{i,(j,m)}^2 + w_{i,(j,m+1)}$	$128N_r$

So, we have $128(3N_r+1)$ variables and $128(4N_r+1)$ equations in the encryption part totally.

(ii) **The key schedule part**.
Because the subword procedure is divided into inversion and transformation, and in every round of key schedule, only a quarter of the round keys takes part in the subword procedure, so there are 32 variables needed to be introduced to generate equations in every round which will be denoted as $\mathbf{k}'$. These variables will simplify the equations and reduce the sparseness. We list the equations of key generation in AES-128 as follows:

$$\begin{cases} \mathbf{k}_{i,(j+13,m)} \mathbf{k}'_{i,(j,m)} = \mathbf{1} \quad j = 0,1,2 \\ \mathbf{k}_{i,(12,m)} \mathbf{k}'_{i,(3,m)} = \mathbf{1} \\ \mathbf{k}_{i+1,(j,m)} = Lin_B \mathbf{k}'_{i+1,(j,m)} + \mathbf{k}_{i,(j,m)} \quad j = 0,1,2,3 \\ \mathbf{k}_{i+1,(j,m)} = \mathbf{k}_{i+1,(j-4,m)} + \mathbf{k}_{i,(j,m)} \quad j = 4,\cdots,15 \end{cases}$$

For simplicity, we omitted Rcon (in Fig.1) in the third equation, because constant term will not affect the total sparseness and complexity. So there are $32N_r$ new variables will be introuced to generate $128N_r + 32N_r$ equations.

So, the equation system built for the key schedule contains $128(5N_r+1)+32N_r$ equations and $128(3N_r+1)+32N_r$ variables. When $n=10$, the MQ system have 6848 equations and 4288 variables.

In order to estimate the complexity of attacking MQ system, we need to give the upper bound of total sparseness of the equation system. The sparseness of the whole equation system is illustrated in the following Table 3.

Table 3. Sparseness of Equations

equation	sparseness
$0 = w_{0,(j,m)} + p_{j,m} + k_{0,(j,m)}$	128×3
$0 = w_{i,(j,m)} + \sum_{j',m'} \alpha_{(j,m),(j',m')} x_{i-1,(j',m')}$	$128(N_r-1) \times 13$
$0 = c_{j,m} + k_{10,(j,m)} + \sum_{(j',m')} \beta_{(j,m),(j',m')} x_{9,(j',m')}$	128×14
$0 = x_{i,(j,m)} w_{i,(j,m)} + 1$	$128N_r \times 2$
$0 = x^2_{i,(j,m)} + x_{i,(j,m+1)}$	$128N_r \times 2$
$0 = w^2_{i,(j,m)} + w_{i,(j,m+1)}$	$128N_r \times 2$

The values of $128(N_r-1)\times 13$ and 128×14 in the second and third rows are obtained from a simulation result, which shows that the sparseness of M_B in subsection 2.2.2 and M_B^* in subsection 2.3 is 12. The sparseness of other equations can be obviously obtained from what they look like. Therefore, the sparseness of the equation system in the encryption part is $128n \times 19 + 128 \times 4$.

We have discussed the number of equations and variables derived from the key schedule, which contains $128N_r$ linear equations and $32N_r$ quadratic equation. If we take the key schedule into account, the upper bound of sparseness is

$$32N_r \times 14 + 128N_r \times 2.$$

So the upper bound of total sparseness is

$$\begin{aligned} &128N_r \times 19 + 128 \times 4 + 32N_r \times 14 + 128N_r \times 2 \\ =\ &128N_r \times 21 + 32N_r \times 4 + 128 \times 4. \end{aligned}$$

4.2 Scale of Macaulay system

As we discussed above, the scale of Macaulay system is $(r(\overline{d}+1)^n) \times ((\overline{D}+1)^n - 1)$.

If we take the key schedule part into account, in which the number of variables is $128(3n+1)+64(n+1)$, and the number of equations is $128(6n+1)$, then we can get the upper bound of complete solving degree $\overline{D} \leq 3 \times [128(3n+1)+64(n+1)]$. The scale of Macaulay system for AES achieves $(r(\overline{D}+1)^n) \times ((\overline{D}+1)^n - 1)$, approximately $2^{65421} \times 2^{65408}$ when $n=10$. The linear system is too large for classical computer to be dealed with, so quantum algorithm is necessary.

The ideas of the following subsection 4.3 and 4.4 come from [7] and [8]

4.3 Complexity of Solving System by BoolSol

We know that the scale of $\mathcal{M}_{\mathcal{F}_2,D}$ is $(r(\overline{d}+1)^n) \times ((\overline{D}+1)^n - 1)$ and it is $(2n+T)$-sparseness, where $2n$ comes from $\mathbb{H}_{\mathbb{Y}}$, so according to the complexity of modified HHL algorithm, the complexity of step 4 in BoolSol algorithm is given as

$$c\log(N+M)T_{\mathcal{F}_2}\kappa^2/\varepsilon = c\log((r(\overline{d}+1)^n) + ((\overline{D}+1)^n - 1)(2n+T)\kappa^2\sqrt{n/\varepsilon_1}.$$

In Section 3.4, we have also analyzed that the loop of step 3 to step 9 will run at most n times and the loop of step 2 to step 11 will run at most $\lceil \log_{\varepsilon_1} \varepsilon \rceil$ times, if we let $\varepsilon_1 = 1/2$, then the complexity of first loop in BoolSol algorithm is given as

$$\begin{aligned}&\sum_{j=0}^{n-1}(c\log((r(\overline{d}+1)^n)+((\overline{D}+1)^n-1)(2(n-j)+T)\kappa^2\sqrt{n/\varepsilon_1})\\&=c\log((r(\overline{d}+1)^n)+((\overline{D}+1)^n-1)(n(n+1)+nT)\kappa^2\sqrt{n/\varepsilon_1}.\end{aligned}$$

The complexity of second loop in BoolSol algorithm is given as

$$\begin{aligned}&c\log((r(\overline{d}+1)^n)+((\overline{D}+1)^n-1)(n(n+1)+nT)\kappa^2\sqrt{n/\varepsilon_1}\lceil\log_{\varepsilon_1}\varepsilon\rceil\\&=c\log((r(\overline{d}+1)^n)+((\overline{D}+1)^n-1)n^{1.5}(n+1+T)\kappa^2\sqrt{2}\lceil\log 1/\varepsilon\rceil.\end{aligned}$$

Notice that $D \leq 3n$ and $\overline{D}+1 \leq 2D+1$.Further more, this inequality can be simplified as follows

$$\begin{aligned}&\log((r(\overline{d}+1)^n)+((\overline{D}+1)^n-1))\\&\leq \log((r+1)(\overline{D}+1)^n)\\&\leq \log(r+1)+n\log(\overline{D}+1)\\&\leq \log(r+1)+n\log(2D+1)\\&\leq \log(r+1)+n\log(6n+1)\end{aligned}$$

Then the complexity is given as follows,

$$\begin{aligned}&c\log((r(\overline{d}+1)^n)+((\overline{D}+1)^n-1))n^{1.5}(n+1+T)\kappa^2\sqrt{2}\lceil\log 1/\varepsilon\rceil\\&\leq \sqrt{2}c(\log r+n\log(6n+1))n^{1.5}(n+1+T)\kappa^2\lceil\log 1/\varepsilon\rceil\\&=O(n^{2.5}(n+T)\kappa^2\log 1/\varepsilon).\end{aligned}$$

Furthermore, we will show how complexity changes if we consider the s-sparseness split set and transformation technique in Section3.4.

(i) s-sparseness split set. If the initial equation system with total sparseness T contains n variables, after applying s-sparseness split set, where $s = 3$ in [7], the upper bound of its number of variables is $n + T$ and the upper bound of its total sparseness is $2T$.

(ii) Transformation. After applying transformation technique, the number of variables remains unchange and the upper bound of total sparseness changes from $2T$ to $4T$, because the sparseness of $f(f-2)$ can be bound by 6 according to [7], where f is an equation with sparseness 3.

So after applying these two techniques, the complexity is

$$O((n+T)^{2.5}(n+4T)\kappa^2\log 1/\varepsilon). \tag{8}$$

which is more accurate compared to the complexity

$$O((n^{3.5}+T^{3.5})\kappa^2\log 1/\varepsilon) \tag{9}$$

in [7]. We will use this equation (8) instead of the equation (9) in [7] to estimate the complexity of attacking AES-128.

4.4 Complexity of Solving System by FSol

Let $D' = n + \sum_{i=1}^{n} max_j \lfloor \log(deg_{x_i}(f_j)) \rfloor$, $d = max\{2, \log(deg_{x_i}(f_j)), i = 1, \cdots, n, j = 1, \cdots, n\}$, and let $Q(\mathcal{F})$ be the multi-variable quadratic equation system transformed from the original equation system $\mathcal{F}$, Chen and Gao in [8] have shown that

$$\#Q(\mathcal{F}) \leq (T_{\mathcal{F}} + 1)(\sum_{i=1}^{n} \lfloor \log d_i \rfloor) + (n-2)T_{\mathcal{F}}.$$

From the expression of D', we can get $D' - n = \sum_{i=1}^{n} \lfloor \log d_i \rfloor$, so we have

$$\#Q(\mathcal{F}) \leq (T_{\mathcal{F}} + 1)(D' - n) + (n-2)T_{\mathcal{F}},$$

and

$$T_{Q_{\mathcal{F}}} \leq (2T_{\mathcal{F}} + 2)(D' - n) + (2n-3)T_{\mathcal{F}}.$$

In Section 3.5, we have shown that

$$\#P(\mathcal{F}) = O(n \log p + T_{\mathcal{F}} (\log p)^3),$$

and

$$T_{P(\mathcal{F})} = O(T_{\mathcal{F}} (\log p)^5).$$

Then, by replacing $\mathcal{F}$ with $Q(\mathcal{F})$, we have,

$$\begin{aligned}
\#P(Q(\mathcal{F})) &= O((\#Q(\mathcal{F}))\log p + T_{Q_{\mathcal{F}}}(\log p)^3) \\
&= O(((T_{\mathcal{F}} + 1)(D' - n) + (n-2)T_{\mathcal{F}})\log p + ((2T_{\mathcal{F}} + 2)(D' - n) + (2n-3)T_{\mathcal{F}})(\log p)^3) \\
&= O(T_{\mathcal{F}} D' (\log p)^3),
\end{aligned}$$

and

$$\begin{aligned}
T_{P(Q(\mathcal{F}))} &= O(T_{\mathcal{F}} (\log p)^5) \\
&= O(((2T_{\mathcal{F}} + 2)(D' - n) + (2n-3)T_{\mathcal{F}})(\log p)^5) \\
&= O(T_{\mathcal{F}} D' (\log p)^5).
\end{aligned}$$

We can rewrite D' as $D' = n + \sum_{i=1}^{n} max_j \lfloor \log(deg_{x_i}(f_j)) \rfloor = O(n \log d)$, then we have

$$\#P(Q(\mathcal{F})) = O(T_{\mathcal{F}} D' (\log p)^3) = O(n T_{\mathcal{F}} \log d (\log p)^3),$$

and

$$T_{P(Q(\mathcal{F}))} = O(T_{\mathcal{F}} D' (\log p)^5) = O(n T_{\mathcal{F}} \log d (\log p)^5).$$

Since $P(Q(\mathcal{F}))$ can be viewed as the Boolean expression of $\mathcal{F}$, we can solve $P(Q(\mathcal{F}))$ with the algorithm BoolSol, so the compelxity to solve $\mathcal{F}$ is as follows

$$\begin{aligned}
C_{FSol} &= O(\#P(Q(\mathcal{F}))^{2.5}(\#P(Q(\mathcal{F})) + T_{P(Q(\mathcal{F}))})\kappa^2 \log 1/\varepsilon) \\
&= O((T_{\mathcal{F}} D' (\log p)^3)^{2.5}(T_{\mathcal{F}} D' (\log p)^3 + T_{\mathcal{F}} D' (\log p)^5)\kappa^2 \log 1/\varepsilon) \\
&= O(T_{\mathcal{F}}^{3.5} D'^{3.5} \log p^{12.5} \kappa^2 \log 1/\varepsilon),
\end{aligned}$$

or

$$C_{FSol} = O(n^{3.5} T_{\mathcal{F}}^{3.5} \log d^{3.5} \log p^{12.5} \kappa^2 \log 1/\varepsilon).$$

Furthermore, if the initial system $\mathcal{F}$ is an MQ system, then $\mathcal{F}_1 = P(\mathcal{F})$ and

$$\#\mathcal{F}_1 = O(T_{\mathcal{F}}(\log p)^3)$$

$$T_{\mathcal{F}_1} = O(T_{\mathcal{F}}(\log p)^5)$$

The complexity to solve $\mathcal{F}_1$ with FSol is given as follows

$$\begin{aligned} C_{FSol} &= O(\#\mathcal{F}_1^{2.5}(\#\mathcal{F}_1 + T_{\mathcal{F}_1})\kappa^2 \log 1/\varepsilon) \\ &= O((n\log p + T_{\mathcal{F}}(\log p)^3)^{3.5}(n\log p + T_{\mathcal{F}}(\log p)^3 + T_{\mathcal{F}}(\log p)^5)\kappa^2 \log 1/\varepsilon) \\ &= O(T_{\mathcal{F}}^{3.5}\log p^{12.5}\kappa^2 \log 1/\varepsilon). \end{aligned} \tag{10}$$

4.4.1 Solving polynomial equations over F_q

We have discussed how to solve a polynomial equation system over F_p, where p is a prime. In this subsection, we will extend our conclusion to that over F_q, where $q = p^m$. Let $F_q = F_p(\theta)$, where θ is a root of $\xi(x) = 0$, and $\xi(x)$ is a monic irreducible polynomial with $deg(\xi) = m$.

For any $g \in F_q[X] = F_p[\theta, X]$, we let $x_i = \sum_{j=0}^{m-1} x_{ij}\theta^j, c = \sum_{j=0}^{m-1} c_j\theta^j$, and g can be rewritten as $g = \sum_{j=0}^{m-1} g_j\theta^j$, where $g_j \in F_p[X_\theta], X_\theta = \{x_{ij} | i = 1, \cdots, n, j = 0, \cdots, m-1\}$. If we define

$$G(g) = \{g_0, g_1, \cdots, g_{m-1}\} \subset F_p[X_\theta]$$

and

$$G(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} G(f) \subset F_p[X_\theta],$$

then we have a bijective mapping as follows

$$\prod_q : \mathbb{V}_{F_p}(G(\mathcal{F})) \to \mathbb{V}_{F_q}(\mathcal{F}),$$

where

$$\prod_q (x_{ij}) = (\sum_{j=0}^{m-1} x_{1j}\theta^j, \cdots, \sum_{j=0}^{m-1} x_{nj}\theta^j).$$

Now the conslusion on $\mathbb{V}_{F_p}(\mathcal{F})$ is also ture on $\mathbb{V}_{F_q}(\mathcal{F})$. The algorithm FSol can solve the polynomial equation system over F_q too. The following focuses on the complexity of FSol dealing with the system over F_q.

It is obvious that $\#G(\mathcal{F}) = m\#\mathcal{F} = mr, \#X_\theta = m\#X = mn$. For any element in an MQ system, it can be expressed as one of the following form,

$$c, cx, cxy,$$

Since the sparseness of the constant and the linear terms over F_q are bounded by m and m^2, all that we have to do is to analyze the terms with the form of cxy. We have $T_{G(cxy)} \leq m^3$, furthermore, $T_{G_{\mathcal{F}}} \leq m^3 T_{\mathcal{F}}$. Therefore,

$$\begin{aligned} \#Q(\mathcal{F}) &\leq r + (T_{\mathcal{F}} + 1)(\sum_{i=1}^{n} \lfloor \log d_i \rfloor) + (n-2)T_{\mathcal{F}} \\ &= r + (T_{\mathcal{F}} + 1)(D' - n) + (n-2)T_{\mathcal{F}} \\ &= O(T_{\mathcal{F}} D'), \end{aligned}$$

and

$$\begin{aligned} T_{Q_{\mathcal{F}}} &\leq (2T_{\mathcal{F}}+2)\sum_{i=1}^{n}\lfloor \log d_i \rfloor + (2n-3)T_{\mathcal{F}} \\ &= (2T_{\mathcal{F}}+2)(D'-n) + (2n-3)T_{\mathcal{F}} \\ &= O(T_{\mathcal{F}}D'). \end{aligned}$$

We have

$$\#G(Q(\mathcal{F})) = O(mT_{\mathcal{F}}D'),$$

and

$$T_{G(Q(\mathcal{F}))} = O(m^3 T_{\mathcal{F}} D').$$

With the complexity in equation (10), we get

$$\begin{aligned} &O((m^3 T_{\mathcal{F}} D')^{3.5} \log p^{12.5} \kappa^2 \log 1/\varepsilon) \\ &= O(m^{10.5} T_{\mathcal{F}}^{3.5} D'^{3.5} \log p^{12.5} \kappa^2 \log 1/\varepsilon) \end{aligned}$$

If the initial system is an MQ system, the complexity is

$$O(m^{10.5} T_{\mathcal{F}}^{3.5} \log p^{12.5} \kappa^2 \log 1/\varepsilon)$$

Furthermore, if $q = 2^m$, the complexity can be rewritten as

$$O(m^{10.5} T_{\mathcal{F}}^{3.5} \kappa^2 \log 1/\varepsilon)$$

This complexity will be used in the next section for the complexity analysis of attacking AES by using the modified HHL algorithm.

5 Complexity of attacking AES using modified HHL algorithms

5.1 Complexity of attacking AES-128 using BoolSol

In Section 4.3, we have shown that the complexity of solving the boolean equation system with sparseness T using BoolSol is

$$O((n+T)^{2.5}(n+4T)\kappa^2 \log 1/\varepsilon).$$

From the Appendix we can see that for AES-128 with N_r round, the number of equations is $128 \times 4N_r$ and the number of variables is $128 \times 4N_r$. Equations can be divided into three parts according to their sparseness.

(i) The keyExpansion part. There are $128N_r$ equations with sparseness $16528N_r$.
(ii) The subBytes and shiftRow part.There are $128N_r$ equations with sparseness $16208N_r$.
(iii) The mixColumn part. There are $128(N_r - 1)$ equations with sparseness $732N_r$.
(iv) The AddRoundKey part. There are $128(N_r + 1)$ equations with sparseness no more than $3 \times 128(N_r + 1)$.

So the total sparseness is $T = 16528N_r + 16208N_r + 732(N_r - 1) + 384N_r$ for different round N_r. Obviously, the sparseness is much larger than the number of the variables n. Taking $\varepsilon = 1\%$, we illustrate the complexity of attacking AES-128 using BoolSol in Table 4.

From this table, we can see that the complexity of attacking 10-round AES-128 is $O(2^{72.98}\kappa^2)$, which is different from $O(2^{73.30}\kappa^2)$ in [7]. We think that the following reasons lead to different complexity. The first one is the equation system in [7] is different from ours, and we made some improvements during the equation establishment process. The second one is that we calculated the complexity according to the equation (8) which is more accurate than that in [7].

Table 4. Complexity of Attacking AES-128 using BoolSol

Round	Number of Eqs.	Number of Vars.	$T_{\mathcal{F}}$	Complexity
1	512	512	33504	$O(2^{61.32}\kappa^2)$
2	1024	1024	67356	$O(2^{64.84}\kappa^2)$
3	1536	1536	101208	$O(2^{66.89}\kappa^2)$
4	2048	2048	135060	$O(2^{68.35}\kappa^2)$
5	2560	2560	168912	$O(2^{69.48}\kappa^2)$
6	3072	3072	202764	$O(2^{70.40}\kappa^2)$
7	3584	3584	236616	$O(2^{71.18}\kappa^2)$
8	4096	4096	270468	$O(2^{71.86}\kappa^2)$
9	4608	4608	304320	$O(2^{72.45}\kappa^2)$
10	5120	5152	338172	$O(2^{72.98}\kappa^2)$

5.2 Complexity of attacking AES-128 using FSol

In Section 4.4, we have shown that the complexity of FSol is

$$O(m^{10.5}T_{\mathcal{F}}^{3.5}\kappa^2\log(1/\varepsilon))$$

by using the original HHL algorithm. Under the condition that the initial system is an MQ, $m = 8$, $\varepsilon = 1\%$, and $T_{\mathcal{F}}$ is total sparseness of equation system derived from BES, we illustrate the complexity of attacking AES-128 as follows,

Table 5. Complexity of Attacking AES-128 using FSol

Round	Number of Eqs.	Number of Vars.	$T_{\mathcal{F}}$	Complexity
1	800	544	3328	$O(2^{79.09}\kappa^2)$
2	1472	960	6144	$O(2^{82.19}\kappa^2)$
3	2144	1376	8960	$O(2^{84.09}\kappa^2)$
4	2816	1792	11766	$O(2^{85.47}\kappa^2)$
5	3488	2208	14592	$O(2^{86.55}\kappa^2)$
6	4160	2624	17408	$O(2^{87.44}\kappa^2)$
7	4832	3040	20224	$O(2^{88.20}\kappa^2)$
8	5504	3456	23040	$O(2^{88.86}\kappa^2)$
9	6176	3872	25856	$O(2^{89.44}\kappa^2)$
10	6848	4288	28672	$O(2^{89.96}\kappa^2)$

The equation system in this paper is defined over $GF(2^8)$. We sovled it by using FSol, and considered the variation of the number of variables, sparseness and the number of equations during the transformation process. From Table 4 and Table 5, we found that, for a full-round AES-128,the attacking compleixty based on HHL algorithm is always higher than that based

on Grover algorithm no matter how small the parameter κ is. That is, AES-128 has a good performance in resisting the quantum algebraic attack using HHL algorithm.

We found that complexity of attacking AES-128 using FSol is larger than that of BoolSol, here are two reasons.

(i) FSol includes more steps than BoolSol. When an equation system over finite field is solved by FSol, it is required to firstly transformed into an equation system over $\mathbb{C}$ with controlled sparseness, then solved by BoolSol. We think that only when the parameters of equation system over finite field, such as number of variables and sparseness, is much better than that of boolean equation system, can FSol perform better than BoolSol.
(ii) Two equation systems are different. The boolean equation system is derived from [10] and the equation system over finite field is derived from [11]. Although both are built mainly according to algebraic structure of AES, the BES mapped one byte of AES to an array of 8 bytes, which increases the number of variables and sparseness.

5.3 Complexity of attacking AES-128 using the improved BoolSol

In Section 3.7, we showed that the complexity of the variant of BoolSol is

$$O(poly(n)\kappa(\tilde{\mathcal{B}}))[9]$$

where $\kappa(\tilde{\mathcal{B}})$ is the condition number of Macaulay matrix. As we have analyzed the number of variables in our equation system, so we illustrate the complexity of attacking AES-128 using the variant of BoolSol proposed in [9] in Table 6.

Table 6. Complexity of Attacking AES-128 using the variant of BoolSol

Round	Number of Vars.	Complexity
1	512	$O(poly(512)\kappa(\tilde{\mathcal{B}}))$
2	1024	$O(poly(1024)\kappa(\tilde{\mathcal{B}}))$
3	1536	$O(poly(1536)\kappa(\tilde{\mathcal{B}}))$
4	2048	$O(poly(2048)\kappa(\tilde{\mathcal{B}}))$
5	2560	$O(poly(2560)\kappa(\tilde{\mathcal{B}}))$
6	3072	$O(poly(3072)\kappa(\tilde{\mathcal{B}}))$
7	3584	$O(poly(3584)\kappa(\tilde{\mathcal{B}}))$
8	4096	$O(poly(4096)\kappa(\tilde{\mathcal{B}}))$
9	4608	$O(poly(4608)\kappa(\tilde{\mathcal{B}}))$
10	5152	$O(poly(5152)\kappa(\tilde{\mathcal{B}}))$

From the table we can see that the complexity of attacking AES-128 using the variant of BoolSol is mainly dependent on the condition number $\kappa(\tilde{\mathcal{B}})$. And the complexity seems smaller than that derived from the BooSol. There are mainly two reasons.

(i) The scale of boolean Macaulay matrix is smaller than that of Macaulay matrix while their sparseness are both s[9].
(ii) In the variant of BoolSol, the original HHL algorithm only need to be repeated for $\log n$ times, while in BoolSol it need to be repeated at least n times, where n is the number of all varables introduced in the equations.

Ding et al. considered the truncated quantum linear system condition number as $\kappa_{\overrightarrow{\mathfrak{b}}}(\mathcal{M}) = ||\mathcal{M}||\frac{||\mathcal{M}^{\dagger}\overrightarrow{\mathfrak{b}}||}{||\overrightarrow{\mathfrak{b}}||}$, which is viewed as the lower bound of $\kappa(\mathcal{M})$[9]. We will give an estimate on the complexity by means of the lower bound. In the algebraic attack, the equation system built from the AES has a unique solution vector. So, if we denote the solving degree as $\overline{D}$, and the hamming weight of the solution vector as h, then $\kappa(\mathcal{M}) \geq \kappa_{\overrightarrow{\mathfrak{b}}}(\mathcal{M}) \geq \sqrt{((\overline{D}+1)^h - 1)}$.

Furthermore, when it comes to the boolean equation system, the lower bound will be much smaller. As we analyzed in Section 3.4, the solving degree of boolean Macaulay system is 1 while the solving degree of Macaulay system is $3n$. Then we can get $\kappa_{\overrightarrow{\mathfrak{b}}}(\mathcal{B}) \geq \sqrt{(2^h - 1)}$.

Finally, we analyze the complexity of solving boolean Macaulay system using HHL algorithm again. If we substitute the lower bound of $\kappa_{\overrightarrow{\mathfrak{b}}}(\mathcal{B})$ into Table 6, we can get an estimation on the lower bound of complexity, i.e., $O(poly(n)\sqrt{(2^h - 1)})$. For 10-round AES-128, the number of variables in the solution vector is 5152. Only when h is less than or equal to 128, the lower bound of the complexity may be better than Grover algorithm. We consider an extreme case, i.e., the all bits of the key are set to 0. Due to the function of S-box, the 0 in the first round will be transformed into 63 in hexadecimal. The diffusion and confusion of the block ciphers will make the subsequent bit be 1 with a probability of about 0.5. To a certain extent, the new variables introduced in the equations reflected the feature. On the other hand, Note that the real key of AES-128 is usually chosen randomly, hence the Hamming weight of the key is about 64. It implies that the hamming weight of the remaining 5024 bits in the solution vector needs to be less than 64 in order to achieve a better attack effect than Grover's algorithm. Let the probability that each introduced variable is 1 be p. If we take $p = 0.1$, the probability that the number of 1s in 5024 bits is less than 64 is about $2^{-482.50}$ according to the binomial distribution. As p approaches 0.5, the probability becomes smaller. It implies that the probability of $h \leq 128$ can be neglected. Therefore, the attack effect of variants of HHL algorithm based on boolean Macaulay matrix is difficult to be better than Grover algorithm.

6 Classical and quantum algorithms to determine the condition number κ

This section looks through the existing methods for determining the parameter κ, which is a key factor to determine the exact value of the attacking complexity. We show that, for a large matrix, it is difficult to find the value of the condition number of the matrix by using current either classical algorithms or quantum algorithms in existence.

6.1 Classical algorithms to estimate κ

Let the largest and the smallest eigenvalue of Macaulay matrix $\mathcal{M}$ be λ_1 and λ_n respectively, then $\kappa = \lambda_1/\lambda_n$. In order to calculate λ_1 and λ_n, we can solve $(\mathcal{M} - \lambda\mathbf{E})x = 0$, i.e. $\det(\mathcal{M} - \lambda\mathbf{E}) = 0$, but the order of the Macaulay matrix is huge, which results in this corresponding polynomial is too complex to be solved, so this method is infeasible.

Instead, we consider the iteration method. If we just want the largest and smallest eigenvalues, Lanczos method is most suitable. Sanjeev and Elad have proven that for an $\ell \times \ell$ matrix with the total sparseness T, Lanczos method' complexity is $\tilde{O}(min\{N, \frac{\ell^{1.5}}{\varepsilon\alpha}\} \cdot \sqrt{\frac{\ell}{\varepsilon}})$[16], where ε is error bound and α is the initial guess value of the smallest eigenvalue. Obviously, for the Macaulay matrix with $\ell \geq 2^{9208}$, the Lanczos method's complexity is still too much.

6.2 Quantum algorithms to calculate κ

It is known that for a unitary matrix $\mathcal{U}$ whose eigenvector can be represented as quantum state $|\phi\rangle$, Quantum Phase Estimation(QPE) algorithm can find a λ such that $\mathcal{U}|\phi\rangle = e^{2\pi i\lambda}|\phi\rangle$ with high probability. As a subroutine of HHL algorithm, QPE was used for decomposing vector in vector space constructed by eigenvalues of $\mathcal{U}$. Suppose we have a quantum state $|b\rangle = \sum b_i|\phi\rangle$ with $b_i \neq 0$ for any i, then applying QPE to it, we can get

$$\mathcal{U}|b\rangle = \mathcal{U}(\sum b_i|\phi\rangle) = \sum e^{2\pi i\lambda_i} b_i|\phi_i\rangle$$

where $|\phi_i\rangle$ is eigenvector of $\mathcal{U}$ and λ_i is its corresponding eigenvalue[17].

When we want to search a disired quantum state, Grover algorithm comes into mind, however there are two problems.

(i) Grover algorithm searches for the disired result which is hidden in quantum state, but the largest eigenvalue λ_1 and the smallest eigenvalue λ_n are hidden in amplitude.

(ii) Although Grover algorithm can exponentially improve the complexity, however, the resulting complexity is still too much because the scale of Macaulay matrix is very huge.

In chemistry, Quantum Variational Eigenvalue(QVE) algorithm was used to estimate the ground energy of atoms and it can also be used to estimate the smallest eigenvalue of matrix. By adjusting quantum state $|\psi\rangle$, the value of following equation tends to the ground energy of Hamiltonian $\mathcal{H}$(state of atom can be represented by Hamiltonian)

$$\langle\psi|\mathcal{H}|\psi\rangle = \frac{\langle\psi|\mathcal{H}|\psi\rangle}{\langle\psi|\psi\rangle},$$

so if we use a sequence of parameters $\{\overrightarrow{t}\}$ to control the preparation of quantum state $|\psi\rangle$, then apply Quantum Expection Estimation algorithm to it to get its average energy. After that, an optimization method, the Nelder-Mead Search(NMS) algorithm, can be used to optimize parameters $\{\overrightarrow{t}\}$ to get a better quantum state. Repeat these steps, the result will approach ground energy gradually[18].

When it comes to our problem, Hamiltonian is equivalent to the coefficient matrix and groud energy can be seen as the smallest eigenvalue, so QVE algorithm can be used to find the smallest eigenvalue theoretically, we then sketch complexity of QVE algorithm.

The complexity of simulating Hamiltonian $e^{i\mathcal{H}t}$ is $O(\log(N)s^2t)$, where N and s are scale and spaeseness of matrix $\mathcal{H}$ respectively. In every round, we need to repeat $O(|h_{max}|Mp^{-2})$ times in order to achieve the precision p , where $|h_{max}|$ and M are the biggest coefficient and number of terms of decomposition of $\mathcal{H}$. The complexity of Nelder-Mead Search(NMS) algorithm is $O(\log(N)(N^2+N\log(N)s^2))$[19]. So we can get the complexity of QVE algorithm, i.e.,

$$O(|h|_{max}M\log^2(N)s^2tp^{-2})(N^2 + N\log(N)s^2),$$

in which $t = 1$ in actual implemention.

We can see that, for a matrix with a very large size, the condition number is difficult to find out whether using the classical algorithm or the quantum algorithm. Unless an algorithm with an exponential improvement is proposed, it is hard to determine the exact value of the condition number of the corresponding Macaulay matrix derived from the two equation systems.

7 Conclusion

This paper discussed the quantum security of AES-128 against HHL algorithm. We perfected the equation system built in [7]. Furthermore, we built two types of equation system, a boolean equation system based on [10] and an equation system over $GF(2^8)$ based on [11], and investigated the number of variables, sparseness and the transformation process of two types of equation systems. And then, we analyzed the complexity of solving two types of equation systems by using BoolSol, FSol and improved BoolSol respectively.

Our result shows that complexity of attacking AES-128 using HHL algorithm is larger than that of Grover algorithm. The complexity of quantum algebraic attack on a cryptographic algorithm depends on the equation system of the cryptographic algorithm and the corresponding quantum solution algorithm.

Acknowledgement

The work was supported in part by the Key Research and Development Program of Shaanxi (No. 2021ZDLGY06-04) and the Guangxi Key Laboratory of Cryptography and Information Security (No.GCIS201802).

References

1. N.-F. Standard. Announcing the advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197(1-51):3–3, 2001.
2. C. E. Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.
3. A. Kipnis and A. Shamir. Cryptanalysis of the hfe public key cryptosystem by relinearization. In *Annual International Cryptology Conference*, pages 19–30. Springer, 1999.
4. N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 392–407. Springer, 2000.
5. N. T. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 267–287. Springer, 2002.
6. H. Nover. Algebraic cryptanalysis of aes: an overview. *University of Wisconsin, USA*, pages 1–16, 2005.
7. Y.-A. Chen and X.-S. Gao. Quantum algorithm for boolean equation solving and quantum algebraic attack on cryptosystems. *Journal of Systems Science and Complexity*, pages 1–40, 2021.
8. Y.-A. Chen, X.-S. Gao, and C.-M. Yuan. Quantum algorithms for optimization and polynomial systems solving over finite fields. *arXiv preprint arXiv:1802.03856*, 2018.
9. J. Ding, V. Gheorghiu, A. Gilyén, S. Hallgren, and J. Li. Limitations of the macaulay matrix approach for using the hhl algorithm to solve multivariate polynomial systems. *arXiv preprint arXiv:2111.00405*, 2021.
10. M. Dubois and E. Filiol. Hacking of the aes with boolean functions. *Representations*, 15:1, 2017.
11. S. Murphy and M. J. Robshaw. Essential algebraic structure within the aes. In *Annual International Cryptology Conference*, pages 1–16. Springer, 2002.
12. N. Ferguson, R. Schroeppel, and D. Whiting. A simple algebraic representation of rijndael. In *International Workshop on Selected Areas in Cryptography*, pages 103–111. Springer, 2001.
13. A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.
14. Y. Cao, A. Daskin, S. Frankel, and S. Kais. Quantum circuit design for solving linear systems of equations. *Molecular Physics*, 110(15-16):1675–1680, 2012.

15. D. Lazard. Gröbner bases, gaussian elimination and resolution of systems of algebraic equations. In *European Conference on Computer Algebra*, pages 146–156. Springer, 1983.
16. S. Arora, E. Hazan, and S. Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 339–348. IEEE, 2005.
17. U. Dorner, R. Demkowicz-Dobrzanski, B. Smith, J. Lundeen, W. Wasilewski, K. Banaszek, and I. Walmsley. Optimal quantum phase estimation. *Physical review letters*, 102(4):040403, 2009.
18. A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. Obrien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5(1):1–7, 2014.
19. S. Singer and S. Singer. Complexity analysis of nelder-mead search iterations. In *Proceedings of the 1. Conference on Applied Mathematics and Computation*, pages 185–196, 1999.

Appendix A

Our boolean equation system is derived from [10] by using the truth table and Mobius transformation. We will take an example to illustrate this as follows.

Suppose that we have the following truth table:

A_2	A_1	A_0	f
0	0	0	x_0
0	0	1	x_1
0	1	0	x_{10}
0	1	1	x_{11}
1	0	0	x_{100}
1	0	1	x_{101}
1	1	0	x_{110}
1	1	1	x_{111}

Then we can get an expression used to describe this truth table as follows:

$$\begin{aligned} f = {} & x_0(1+A_0)(1+A_1)(1+A_2) \\ & + x_1A_0(1+A_1)(1+A_2) \\ & + x_{10}(1+A_0)A_1(1+A_2) \\ & + x_{11}A_0A_1(1+A_2) \\ & + x_{100}(1+A_0)(1+A_1)A_2 \\ & + x_{101}A_0(1+A_1)A_2 \\ & + x_{110}(1+A_0)A_1A_2 \\ & + x_{111}A_0A_1A_2 \end{aligned}$$

which can be simplified as

$$\begin{aligned} f &= x_0 \\ &+ (x_0 + x_1)A_0 \\ &+ (x_0 + x_{10})A_1 \\ &+ (x_0 + x_{100})A_2 \\ &+ (x_0 + x_1 + x_{10} + x_{11})A_0A_1 \\ &+ (x_0 + x_1 + x_{10} + x_{101})A_0A_2 \\ &+ (x_0 + x_{10} + x_{100} + x_{110})A_1A_2 \\ &+ (x_0 + x_1 + x_{10} + x_{11} + x_{100} + x_{101} + x_{110} + x_{111})A_0A_1A_2 \end{aligned}$$

To make it clearer, we denote $f = \sum_{k=0}^{7} y_k a_k$, then the relationship of x_k and y_k can be expressed as a matrix:

$$\overrightarrow{\mathbf{y}} = \overrightarrow{\mathbf{x}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\overrightarrow{\mathbf{y}} = \left[y_0, y_1, y_{10}, y_{11}, y_{100}, y_{101}, y_{110}, y_{111}\right]$ and $\overrightarrow{\mathbf{x}} = \left[x_0, x_1, x_{10}, x_{11}, x_{100}, x_{101}, x_{110}, x_{111}\right]$. Denote this matrix as $\mathbf{T_3}$ and it can be blocked as

$$\left[\begin{array}{cc|cc|cc|cc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \begin{array}{rc} 0 \rightarrow & \{\} \\ 1 \rightarrow & \{A_0\} \\ 10 \rightarrow & \{A_1\} \\ 11 \rightarrow & \{A_0, A_1\} \\ 100 \rightarrow & \{A_2\} \\ 101 \rightarrow & \{A_0, A_2\} \\ 110 \rightarrow & \{A_1, A_2\} \\ 111 \rightarrow & \{A_0, A_1, A_2\} \end{array}$$

If we denote the corresponding variable subset of the $i-th$ row and $j-th$ column as S_i and S_j, respectively, we have

$$T(i,j) = \begin{cases} 1 & if\ S_i \subseteq S_j \\ 0 & otherwise \end{cases}$$

Through this example we can see that, when we want to transform a truth table to its corresponding ANF, we have two choices. 1. Build matrix $\mathbf{T}$ and do a matrix multiplication. 2. According to the property of block matrix $\mathbf{T_3}$ we can see that recursion can be used to deal with the truth table to get the corresponding ANF.

A.1 Build Boolean Equation System

The boolean equation system can be divided into the encryption part and key schedule part. In this section, B and b will be used to denote byte and bit, respectively, and the output will be denoted as B' and b'.

A.1.1 Boolean Equation System of Encryption part

Several procedures are included in the encryption part, which are subBytes, shiftRow and mixColumn.

(i) SubBytes[10]. Input and output of subBytes are both of length 8, so we can easily get a truth table of every output bit and then get its ANF using Mobius transformation.

(ii) ShiftRow[10]. Accroding to shiftRow operation defined in AES we can directly get the following equations.

$$
\begin{gathered}
x_0 = x_0\ x_1 = x_1\ x_2 = x_2\ x_3 = x_3 \\
x_4 = x_4\ x_5 = x_5\ x_6 = x_6\ x_7 = x_7 \\
x_8 = x_{40}\ x_9 = x_{41}\ x_{10} = x_{42} \\
x_{11} = x_{43}\ x_{12} = x_{44}\ x_{13} = x_{45}\ x_{14} = x_{46}\ x_{15} = x_{47} \\
x_{16} = x_{80}\ x_{17} = x_{81}\ x_{18} = x_{82} \\
x_{19} = x_{83}\ x_{20} = x_{84}\ x_{21} = x_{85}\ x_{22} = x_{86}\ x_{23} = x_{87} \\
x_{24} = x_{120}\ x_{25} = x_{121}\ x_{26} = x_{122}\ x_{27} = x_{123} \\
x_{28} = x_{124}\ x_{29} = x_{125}\ x_{30} = x_{126}\ x_{31} = x_{127} \\
x_{32} = x_{32}\ x_{33} = x_{33}\ x_{34} = x_{34}\ x_{35} = x_{35} \\
x_{36} = x_{36}\ x_{37} = x_{37}\ x_{38} = x_{38}\ x_{39} = x_{39} \\
x_{40} = x_{72}\ x_{41} = x_{73}\ x_{42} = x_{74}\ x_{43} = x_{75} \\
x_{44} = x_{76}\ x_{45} = x_{77}\ x_{46} = x_{78}\ x_{47} = x_{79} \\
x_{48} = x_{112}\ x_{49} = x_{113}\ x_{50} = x_{114}\ x_{51} = x_{115} \\
x_{52} = x_{116}\ x_{53} = x_{117}\ x_{54} = x_{118}\ x_{55} = x_{119} \\
x_{56} = x_{24}\ x_{57} = x_{25}\ x_{58} = x_{26}\ x_{59} = x_{27} \\
x_{60} = x_{28}\ x_{61} = x_{29}\ x_{62} = x_{30}\ x_{63} = x_{31} \\
x_{64} = x_{64}\ x_{65} = x_{65}\ x_{66} = x_{66}\ x_{67} = x_{67} \\
x_{68} = x_{68}\ x_{69} = x_{69}\ x_{70} = x_{70}\ x_{71} = x_{71} \\
x_{72} = x_{104}\ x_{73} = x_{105}\ x_{74} = x_{106}\ x_{75} = x_{107} \\
x_{76} = x_{108}\ x_{77} = x_{109}\ x_{78} = x_{110}\ x_{79} = x_{111} \\
x_{80} = x_{16}\ x_{81} = x_{17}\ x_{82} = x_{18}\ x_{83} = x_{19} \\
x_{84} = x_{20}\ x_{85} = x_{21}\ x_{86} = x_{22}\ x_{87} = x_{23} \\
x_{88} = x_{56}\ x_{89} = x_{57}\ x_{90} = x_{58}\ x_{91} = x_{59} \\
x_{92} = x_{60}\ x_{93} = x_{61}\ x_{94} = x_{62}\ x_{95} = x_{63} \\
x_{96} = x_{96}\ x_{97} = x_{97}\ x_{98} = x_{98}\ x_{99} = x_{99} \\
x_{100} = x_{100}\ x_{101} = x_{101}\ x_{102} = x_{102}\ x_{103} = x_{103} \\
x_{104} = x_8\ x_{105} = x_9\ x_{106} = x_{10}\ x_{107} = x_{11} \\
x_{108} = x_{12}\ x_{109} = x_{13}\ x_{110} = x_{14}\ x_{111} = x_{15} \\
x_{112} = x_{48}\ x_{113} = x_{49}\ x_{114} = x_{50}\ x_{115} = x_{51} \\
x_{116} = x_{52}\ x_{117} = x_{53}\ x_{118} = x_{54}\ x_{119} = x_{55} \\
x_{120} = x_{88}\ x_{121} = x_{89}\ x_{122} = x_{90}\ x_{123} = x_{91} \\
x_{124} = x_{92}\ x_{125} = x_{93}\ x_{126} = x_{94}\ x_{127} = x_{95}
\end{gathered}
$$

(iii) MixColumn. MixColumn in AES is defined as follows:

$$
\begin{bmatrix} B'_i \\ B'_{i+1} \\ B'_{i+2} \\ B'_{i+3} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} B_i \\ B_{i+1} \\ B_{i+2} \\ B_{i+3} \end{bmatrix}
$$

If we take B'_{15} as an example, we get

$$B'_{15} = 03 \cdot B_{12} + B_{13} + B_{14} + 02 \cdot B_{15}$$

Write $B_{12}, B_{13}, B_{14}, B_{15}$ as polynomial as follows

$$\begin{aligned}
B_{12} &= b_{96}x^7 + b_{97}x^6 + b_{98}x^5 + b_{99}x^4 + b_{100}x^3 + b_{101}x^2 + b_{102}x + b_{103} \\
B_{13} &= b_{104}x^7 + b_{105}x^6 + b_{106}x^5 + b_{107}x^4 + b_{108}x^3 + b_{109}x^2 + b_{110}x + b_{111} \\
B_{14} &= b_{112}x^7 + b_{113}x^6 + b_{114}x^5 + b_{115}x^4 + b_{116}x^3 + b_{117}x^2 + b_{118}x + b_{119} \\
B_{15} &= b_{120}x^7 + b_{121}x^6 + b_{122}x^5 + b_{123}x^4 + b_{124}x^3 + b_{125}x^2 + b_{126}x + b_{127}
\end{aligned}$$

The prime polynomial of AES is $x^8 + x^4 + x^3 + x + 1$, then we can calculate

$$\begin{aligned}
03 \cdot B_{12} &= (x+1)(b_{96}x^7 + b_{97}x^6 + b_{98}x^5 + b_{99}x^4 + b_{100}x^3 + b_{101}x^2 + b_{102}x + b_{103}) \\
&= b_{96}x^8 + (b_{96} + b_{97})x^7 + (b_{97} + b_{98})x^6 + (b_{98} + b_{99})x^5 + (b_{99} + b_{100})x^4 \\
&\quad + (b_{100} + b_{101})x^3 + (b_{101} + b_{102})x^2 + (b_{102} + b_{103})x + b_{103} \\
&= (b_{96} + b_{97})x^7 + (b_{97} + b_{98})x^6 + (b_{98} + b_{99})x^5 + (b_{99} + b_{100} + b_{96})x^4 \\
&\quad + (b_{100} + b_{101} + b_{96})x^3 + (b_{101} + b_{102})x^2 + (b_{102} + b_{103} + b_{96})x + (b_{103} \\
&\quad + b_{96})
\end{aligned}$$

and

$$\begin{aligned}
02 \cdot B_{15} &= x(b_{120}x^7 + b_{121}x^6 + b_{122}x^5 + b_{123}x^4 + b_{124}x^3 + b_{125}x^2 + b_{126}x + b_{127}) \\
&= b_{120}x^8 + b_{121}x^7 + b_{122}x^6 + b_{123}x^5 + b_{124}x^4 + b_{125}x^3 + b_{126}x^2 + b_{127}x \\
&= b_{121}x^7 + b_{122}x^6 + b_{123}x^5 + (b_{124} + b_{120})x^4 + (b_{125} + b_{120})x^3 + b_{126}x^2 \\
&\quad + (b_{127} + b_{120})x + b_{120}
\end{aligned}$$

If we denote $B'_{15} = b'_{120}x^7 + b'_{121}x^6 + b'_{122}x^5 + b'_{123}x^4 + b'_{124}x^3 + b'_{125}x^2 + b'_{126}x + b'_{127}$, then the corresponding coefficients are equal, and the following equation can be obtained:

$$\begin{aligned}
b'_{120} &= b_{96} + b_{97} + b_{104} + b_{112} + b_{121} \\
b'_{121} &= b_{97} + b_{98} + b_{105} + b_{113} + b_{122} \\
b'_{122} &= b_{98} + b_{99} + b_{106} + b_{114} + b_{123} \\
b'_{123} &= b_{99} + b_{100} + b_{107} + b_{115} + b_{124} + x_{96} + x_{120} \\
b'_{124} &= b_{100} + b_{101} + b_{108} + b_{116} + b_{125} + x_{96} + x_{120} \\
b'_{125} &= b_{101} + b_{102} + b_{109} + b_{117} + b_{126} \\
b'_{126} &= b_{102} + b_{103} + b_{110} + b_{118} + b_{127} + x_{96} + x_{120} \\
b'_{120} &= b_{103} + b_{111} + b_{119} + x_{96} + x_{120}
\end{aligned}$$

The ANF of other bytes can be obtained similiarly.

Then we describe how to get the ANF used to describe a full round, where the full round means the rounds include mixColumn, this technology can also be applied to the final round.

Denote the input and output of a round as $B = (b_1, \cdots, b_{128})$ and $B' = (b'_1, \cdots, b'_{128})$, respectively, we have

$$B' = MixColumn(ShiftRow(SubBytes(B)))$$

If we take b'_0 as an example, the ANF of mixColumn of b'_0 is

$$b'_0 = b_9 + b_{16} + b_8 + b_1 + b_{24}$$

the ANF of shiftRow of $b_9, b_{16}, b_8, b_1, b_{24}$ are

$$b_9 = b_{41}\ b_{16} = b_{80}\ b_8 = b_{40}\ b_1 = b_1\ b_{24} = b_{120}$$

then the ANF of b'_0 becomes

$$b'_0 = b_1 + b_{40} + b_{41} + b_{80} + b_{120}$$

we can finally substitute the ANF of subBytes of $b_1, b_{40}, b_{41}, b_{80}, b_{120}$ to get the ANF of b'_0 which is used to describe a full round.

A.1.2 Boolean Encryption System of Key Schedule part

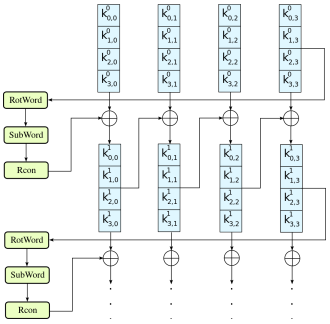

Fig. A.1. Key Schedule of AES-128

For AES-128, if we denote the initial key and the output of $i-th$ round of key schedule as

$$\mathbf{K} = \begin{bmatrix} k_{00} & k_{10} & k_{20} & k_{30} \\ k_{01} & k_{11} & k_{21} & k_{31} \\ k_{02} & k_{12} & k_{22} & k_{32} \\ k_{03} & k_{13} & k_{23} & k_{33} \end{bmatrix}$$

and

$$\mathbf{K^i} = \begin{bmatrix} k^i_{00} & k^i_{10} & k^i_{20} & k^i_{30} \\ k^i_{01} & k^i_{11} & k^i_{21} & k^i_{31} \\ k^i_{02} & k^i_{12} & k^i_{22} & k^i_{32} \\ k^i_{03} & k^i_{13} & k^i_{23} & k^i_{33} \end{bmatrix}$$

respectively, denote the first element in round constant of $i-th$ round as $rcon_i[1]$, we can get following equations which is used to describe the first column

$$\begin{aligned} k^1_{00} &= k_{00} + S(k_{31}) + rcon_i[1] \\ k^1_{01} &= k_{01} + S(k_{32}) + rcon_i[2] \\ k^1_{02} &= k_{02} + S(k_{33}) + rcon_i[3] \\ k^1_{03} &= k_{03} + S(k_{30}) + rcon_i[4] \end{aligned}$$

furthermore, the equation of second column can be represented as

$$\begin{aligned} k_{10}^1 &= k_{10} + k_{00}^1 \\ k_{11}^1 &= k_{11} + k_{01}^1 \\ k_{12}^1 &= k_{12} + k_{02}^1 \\ k_{13}^1 &= k_{13} + k_{03}^1 \end{aligned}$$

using same technique, we can get all 128 equations which are used to describe one round of key schedule.

A.2 Analyze Boolean Equation System

	Plaintext	
AddRoundKey		
	Introduce 128 *new variables*	*Generate* 128 *equations*
SubBytes ShiftRow		
	Introduce 128 *new variables*	*Generate* 128 *equations*
MixColumn		
	Introduce 128 *new variables*	*Generate* 128 *equations*
AddRoundKey		
	Introduce 128 *new variables*	*Generate* 128 *equations*
SubBytes ShiftRow		
	Introduce 128 *new variables*	*Generate* 128 *equations*
MixColumn		
	Introduce 128 *new variables*	*Generate* 128 *equations*
AddRoundKey		
	Introduce 128 *new variables*	*Generate* 128 *equations*
SubBytes ShiftRow		
	Introduce 128 *new variables*	*Generate* 128 *equations*
MixColumn		
	Introduce 128 *new variables*	*Generate* 128 *equations*
AddRoundKey		
	Introduce 128 *new variables*	*Generate* 128 *equations*
SubBytes ShiftRow		
	Introduce 128 *new variables*	*Generate* 128 *equations*
MixColumn		
	Introduce 128 *new variables*	*Generate* 128 *equations*
AddRoundKey		
	Introduce 128 *new variables*	*Generate* 128 *equations*
SubBytes ShiftRow		
	Introduce 128 *new variables*	*Generate* 128 *equations*
MixColumn		
	Introduce 128 *new variables*	*Generate* 128 *equations*
AddRoundKey		
	Introduce 128 *new variables*	*Generate* 128 *equations*
SubBytes ShiftRow		
	Introduce 128 *new variables*	*Generate* 128 *equations*
MixColumn		
	Introduce 128 *new variables*	*Generate* 128 *equations*
AddRoundKey		
	Introduce 128 *new variables*	*Generate* 128 *equations*
SubBytes ShiftRow		
	Introduce 128 *new variables*	*Generate* 128 *equations*
MixColumn		
	Introduce 128 *new variables*	*Generate* 128 *equations*
AddRoundKey		
	Introduce 128 *new variables*	*Generate* 128 *equations*
SubBytes ShiftRow		
	Introduce 128 *new variables*	*Generate* 128 *equations*
MixColumn		
	Introduce 128 *new variables*	*Generate* 128 *equations*
AddRoundKey		
	Introduce 128 *new variables*	*Generate* 128 *equations*
SubBytes ShiftRow		
	Introduce 128 *new variables*	*Generate* 128 *equations*
MixColumn		
	Introduce 128 *new variables*	*Generate* 128 *equations*
AddRoundKey		
	Introduce 128 *new variables*	*Generate* 128 *equations*
SubBytes ShiftRow		
	Introduce 128 *new variables*	*Generate* 128 *equations*
AddRoundKey		
	Ciphertext	*Generate* 128 *equations*

We can see that, $128 \times 29 = 3712$ variables are introduced to generate $128 \times 30 = 3840$ equations in the encryption part, in the key schedule part, $128 \times 11 = 1408$ variables are introduced to generate $128 \times 10 = 1280$ equations. So there are totally 5120 variables and 5120 equations.

The equations can be divided into four parts according to their sparseness.

(i) The key schedule part. There are 1280 equations with sparseness $16528 \times 10 = 165280$.
(ii) The MixColumn part. There are $128 \times 9 = 1152$ equations with sparseness $732 \times 9 = 6588$.
(iii) The SubBytes and ShiftRow part. There are $128 \times 10 = 1280$ equations with sparseness $16208 \times 10 = 162080$.
(iv) The AddRoundKey part. There are $128 = 1408$ equations with sparseness no more than $3 \times 128 \times 11 = 4224$, because the plaintext and ciphertext are known and do not increase sparseness.

So the total sparseness of this equation system is 338172. The complexity of solving this equation system using BoolSol is

$$O((n+T)^{2.5}(n+4T)\kappa^2 \log 1/\varepsilon) = O(2^{72.98}\kappa^2)$$

where we set $\varepsilon = 1/100$.