Grover vs. Cryptography: Is It Safe?

Super Tangle

Abstract. In this report, we show that the application of Grover's search algorithm to key recovery attacks against cryptographic algorithms on IBMQ environments. Cryptographic algorithms, we are targeting, are `Caesar`, `Vigenère`, `Simplified-DES`, and `Simplified-AES`. To apply the Grover search algorithm efficiently, we optimize target cryptographic algorithms in quantum circuits. The implementation is evaluated on two kinds of environments (i.e. qasm-simulator and IBMQ cloud resources). We achieved meaningful results of Grover key search on the simulator. Furthermore, we also perform Grover key search on real quantum processor using IBM cloud service. Finally, we explore the impact of current quantum computers on cryptography.

Keywords: Caesar · Vigenère · DES · Grover Algorithm · IBMQ · Quantum-computer

1 Introduction

International companies, such as IBM, Google, and Microsoft, are advancing the development of large-scale quantum computers. Quantum computers have more computational power than classical computers in certain areas, such as deep learning, chemistry, and cryptography. If a large-scale quantum computer capable of operating quantum algorithms is developed, the safety of currently widely used cryptographic algorithms may be lowered or broken. It has been proven that Shor's algorithm can break the safety of RSA and Elliptic Curve Cryptography (ECC). How long RSA and ECC can be used depends on the development of quantum computers and optimization of Shor's algorithm. In preparation for the collapse of the security of public key cryptography, NIST (National Institute of Standards and Technology) is currently conducting a PQC (Post-Quantum Cryptography) competition.

We focus on another quantum algorithm for cryptanalysis (i.e. Grover's search algorithm). Grover's search algorithm can be used to lower the security level of symmetric key cryptography and hash function. By using Grover's search algorithm, n-bit security symmetric key ciphers can be reduced to $\frac{n}{2}$-bit security.

In this report, we present quantum key searches design for `Caesar`, `Vigenère`, `Simplified-DES`, and `Simplified-AES` using the Grover search algorithm. Test results in this report are based on the use of IBM Quantum Simulator and IBMQ Cloud Quantum Service. Finally, based on the test results, we confirmed the impact of quantum computers on the cryptography.

2 Backgrounds

2.1 Grover's search algorithm

Grover's search algorithm can accelerate the brute force attack. If n searches were required for the brute force attack, it is reduced to $\sqrt{n}$ searches by applying the Grover's search algorithm. Grover's search algorithm consists of an oracle that inverts the sign of the answer, and a diffusion operator that amplifies the amplitude of the inverted answer. When the answer of 2-qubit is 10, 2-qubit after the oracle and the diffusion operator are shown in Figure 1. Grover's search increases the amplitude of measuring the answer by repeating the oracle and the diffusion operator. However, in the case of 2-qubit, the answer can be found with 100% probability in one shot without the repetition.

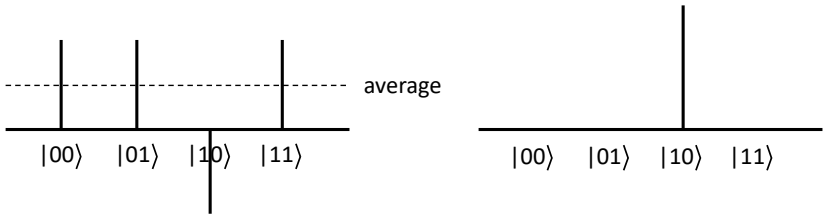


Fig. 1: After performing oracle (left) and diffusion operator (right).

2.2 Quantum Gates

Quantum gates used in quantum computers are reversible gates that can return to their original state during computation. Quantum computing using reversible quantum gates is different from classical computing because there is no logical operator, but there are quantum gates that can replace logical operations performed in classical computers. There are X, $CNOT$, $Toffoli$, and $Swap$ gates, which are shown in Figure 2.

- **X gate**: This quantum gate can replace the NOT operation, inverting the state of the input qubit.

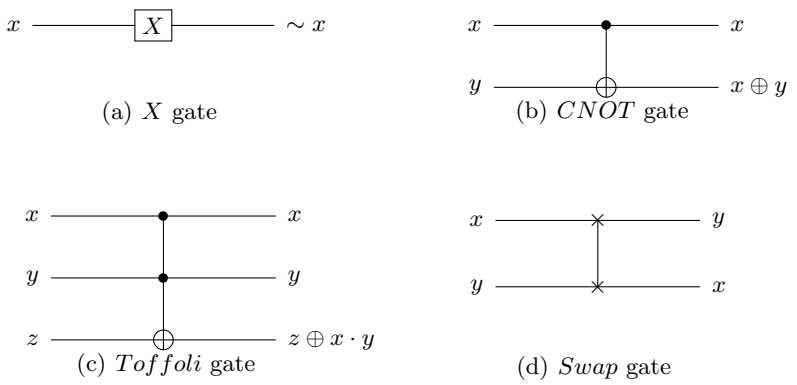
$$X(x) = \sim x.$$

- **CNOT gate**: This quantum gate can replace the XOR operation, inverting the state of y only if x is 1.

$$CNOT(x,\, y) = (x,\, x \oplus y).$$

- **Toffoli gate**: This quantum gate can replace the AND operation, inverting the state of z only if x and y is 1.

$$Toffoli(x,\, y,\, z) = (x,\, y,\, z \oplus x \cdot y).$$



(a) X gate (b) $CNOT$ gate

(c) $Toffoli$ gate (d) $Swap$ gate

Fig. 2: Quantum gates.

– **Swap gate**: This quantum gate swaps the states of x and y.

$$Swap(x, y) = (y, x).$$

2.3 Caesar Cipher

The `Caesar` cipher is a simple substitution cipher that rotates the plaintext by the value of the key K. Figure 3 shows how the plaintext is encrypted with the `Caesar` cipher when the key is `3`. It can be seen that the plaintext is rotated to the right by 3 slots. `A` changes to `D` and `B` changes to `E`.

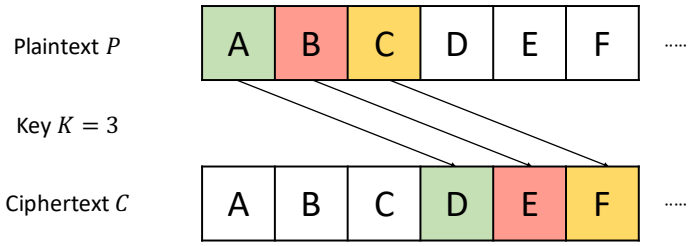


Fig. 3: `Caesar` cipher.

2.4 Vigenère Cipher

The `Vigenère` cipher is a multiple substitution cipher. General multi-substitution ciphers, such as Alberti and Trithemius, had a problem that if the algorithm was leaked, all encrypted data in the same way can be decrypted. However, the

`Vigenère` cipher uses different key chains for each encryption. If attackers know the algorithm, they cannot recover the plaintext without the valid key. In the `Vigenère` cipher, the substitution follows Table 4. If the plaintext `ABC` is encrypted with the key `DEF`, `A` is changes to `D` in the `D` row. `B` is changes to `F` in the `E` row, and `C` is changes to `H` in the `F` row, respectively.

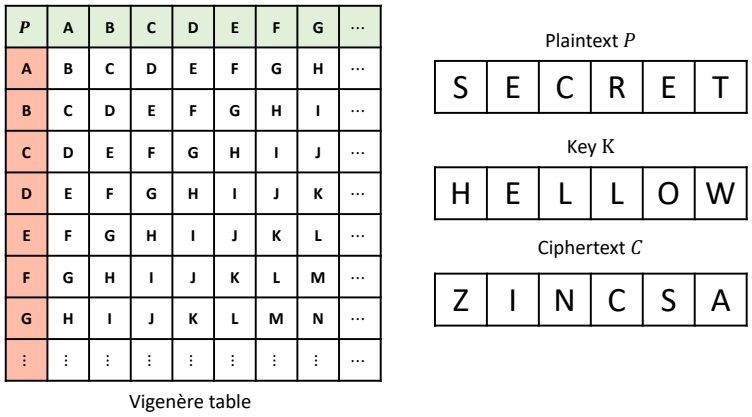


Fig. 4: `Vigenère` algorithm.

2.5 Simple DES

The Simplified Data Encryption Standard (i.e. `SimpleDES`) is a simple description of DES and it is an encryption algorithm used for the purpose of explaining DES easily, rather than a practically usable encryption algorithm. Between DES and `SimpleDES`, the biggest difference is the difference in the length of blocks and keys and the number of rounds. The overall algorithm is shown in the Figure 5.

2.6 Simple AES

The Simplified Advanced Encryption Standard (i.e. `SimpleAES`) uses much smaller parameters than AES, but has similar properties and structure to AES. The differences between AES and `SimpleAES` is the difference in the length of key(16-bit) the block (16-bit) and the number of rounds (2-round). The overall algorithm is shown in the Figure 6.

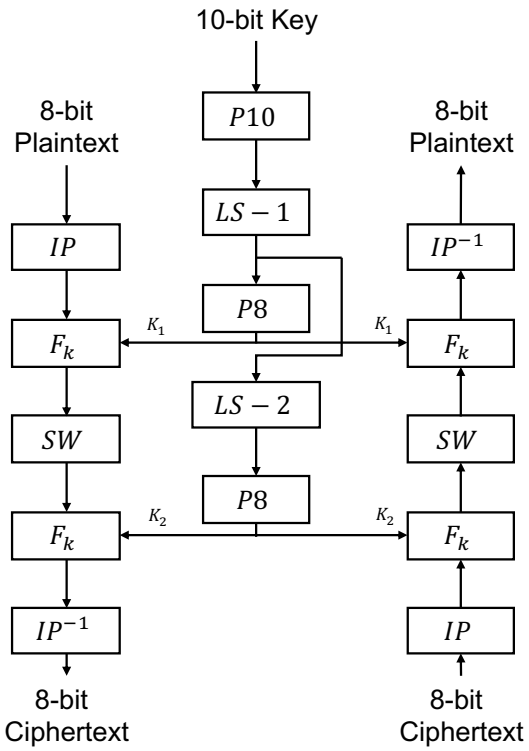


Fig. 5: `Simplified DES` algorithm.

3 Implementation

3.1 Key search using Grover's algorithm

The Grover search algorithm can be applied to key search of symmetric key cryptography. The process of Grover key search is as follows, and the overall structure of the quantum circuit is shown in Figure 7.

- **Input setting**: The n-qubit key exists in superposition state due to Hadamard gates. Due to the superposition state, all possible key values exist simultaneously in qubits as a probability. This is one of main advantages of quantum computers.
- **Oracle**: The encryption process of the target encryption algorithm must be implemented as a quantum circuit in Oracle. Since the encryption quantum circuit occupies most of Oracle, how compact it is designed is important in terms of quantum resources. The encryption circuit encrypts plaintext qubits with the key in the superposition state. Therefore, ciphertexts for all key values are generated in plaintext qubits. Among the generated ciphertexts,

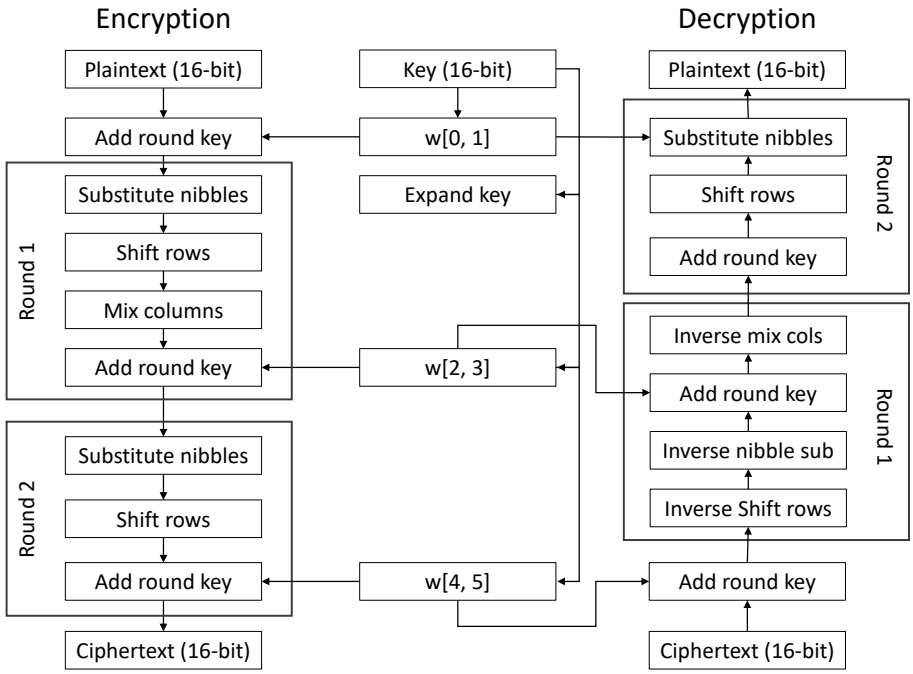


Fig. 6: `Simplified AES` algorithm.

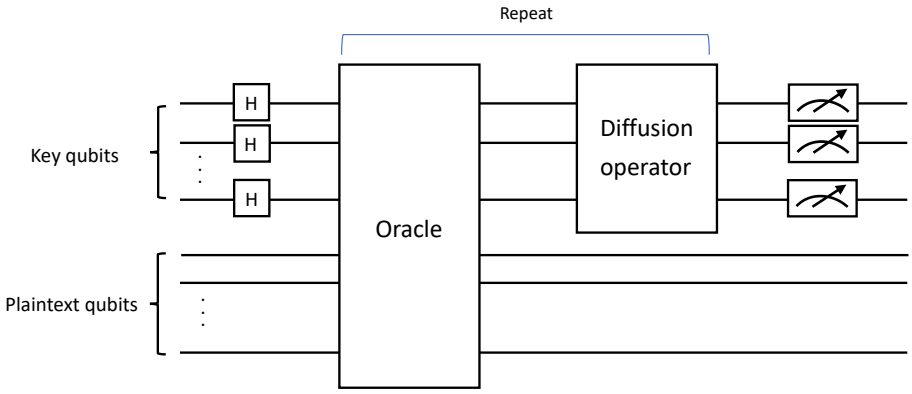


Fig. 7: Grover key search.

X gates and Controlled-Z gates are used to select known ciphertexts. For example, to find the ciphertext $(c_0, c_1, c_2, c_3) = \texttt{0011}$, X gates are performed on c_0 and c_1 qubits and all qubits are set to `1`. Then, a multi controlled-Z gate that operates when c_0, c_1, c_2, and c_3 are all `1` is performed. The key value that interacted in the past and generated ciphertext 0011 is also

inverted due to entanglement. That is, the inverted key value is the solution. Since Grover's search has to be repeated, it performs the reverse operation on the previously performed X gates and the encryption circuit to prepare for the next iteration.

– **Diffusion operator**: The diffusion operator is a module that amplifies the amplitude of the solution found in Oracle, and the following equation is performed.

$$\text{Average amplitude} - (\text{Each amplitude} - \text{Average amplitude}) \tag{1}$$

Through the following equation, the amplitude of the inverted solution increases, and the amplitude of non-solution candidates decreases. Since the amplitude does not sufficiently increase at once, Grover's search repeats Oracle and the Diffusion operator to increase the amplitude of the solution sufficiently before the observation (i.e. measurement). The optimal number of iterations is determined by the number of qubits of the key. If the key is n-qubit, iteration is performed by $\sqrt{2^n}$ times. If more iterations than the optimal number are repeated, it over-fits and the amplitude decreases.

3.2 `Caesar` Cipher in Oracle

The `Caesar` encryption can be expressed as the following Equation 2. In order to implement Equation 2 as a quantum circuit, an quantum adder is required, and we chose the ripple-carry adder [1]. In addition of a and b, the ripple-carry adder stores the result in b, which is the input. It has an advantage that it requires fewer qubits.

$$C_i \leftarrow Enc(P_i, K) = (P_i + K) \bmod 16 \tag{2}$$

In the implemented Add, plaintext P and key K are input, and the result of adding K to plaintext P becomes ciphertext.

$$Add(P_i, K) = \begin{cases} P_i = (P_i + K) \bmod 16, \\ K = K \end{cases} \tag{3}$$

The `Caesar` encryption can be implemented simply with a quantum adder. We optimize by avoiding the top-level carry-qubit computation of the ripple-carry adder by utilizing the characteristic of modular addition to erase the top-level carry. The `Caesar` key search Oracle we designed is shown in Figure 8, including the input setting. Figure 8 is the Oracle design for recovering key $K=$ `0x2` when plaintext $P =$ `0xf2` and ciphertext $C =$ `0x14`.

3.3 `Vigenère` Cipher in Oracle

Unlike the `Caesar` cipher, which encrypts with only one key, the `Vigenère` cipher encrypts plaintext elements with various keys. Similar to the `Caesar` cipher, it

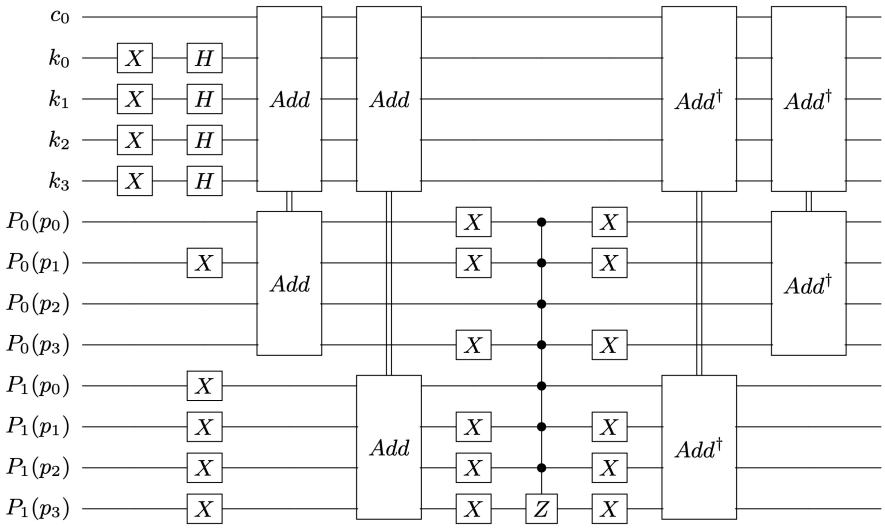


Fig. 8: `Caesar` input setting + key search Oracle.

uses modular addition for encryption as follows. Therefore, we use the same *Add* module, but perform separate key searches for each plaintext element.

$$C_i \leftarrow Enc(P_i, K_{i \bmod j}) = (P_i + K_{i \bmod j}) \bmod 16 \tag{4}$$

$$Add(P_i, K_{i \bmod j}) = \begin{cases} P_i = (P_i + K_{i \bmod j}) \bmod 16, \\ K_{i \bmod j} = K_{i \bmod j} \end{cases} \tag{5}$$

Since each plaintext element is encrypted with a different key, the Grover key search for `Vigenère` is performed as long as the key length. Due to the limitation of qubits, the Grover key search is performed in the case where the plaintext length is 8-qubit, and therefore the key length is 8-qubit. The `Vigenère` key search Oracle we designed is shown in Figure 9, including the input settings. Figure 9 is the Oracle design for recovering key K= `0x8c` when plaintext P = `0x42` and ciphertext C = `0xce`. In Figure 9, plaintext elements are encrypted with different key elements, and a key search is performed for each.

3.4 Quantum circuit implementation of `Simplified-DES`

In the `S-DES` encryption process in Figure 5, Keyschedule is executed for 10-bit key K to generate round keys K_1 and K_2. Then, generated round keys K_1 and K_2 are XORed to the plaintext P_0 and P_1.

We compute the round key K_1 on K without allocating qubits that generate K_1 and K_2, separately. K becomes K_1 and XORs the plaintext. After that, a reverse operation is performed to return K_1 to K again. Finally, the round key

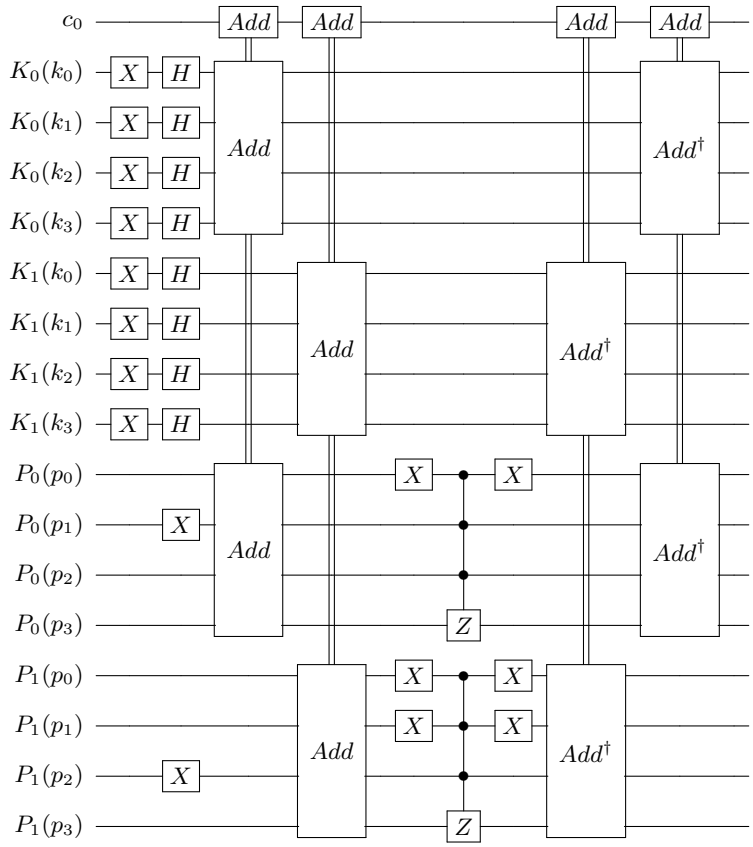


Fig. 9: Input setting and oracle design for `Vigenère` Cipher.

K_2 is computed on K and used as a round key. In this way, we do not allocate additional qubits by changing the 10-qubit key K. The proposed `S-DES` quantum circuit design is shown in Algorithm 1. [2] was referenced when implementing `S-DES` as a quantum circuit.

Algorithm 1 Quantum circuit for `Simplified-DES`.

Input: 8-qubit plaintext $P(p_7, ..., p_0)$, 10-qubit key $K(k_9, ..., k_0)$.
Output: 8-qubit ciphertext $C(c_7, ..., c_0)$.

Round 1 :
  Keyschedule(K_1) :
    $K \leftarrow P10(K)$
    $K \leftarrow LS\text{-}1(K)$
    $K \leftarrow P8(K)$ // Reverse target
  Encryption(K_1) :
    $P \leftarrow IP(P)$
    $P \leftarrow AddRoundkey(K, P)$
    $P \leftarrow F_k(K, P)$
    $P \leftarrow SW(P)$
Round 2 :
  Keyschedule(K_2) :
    $K \leftarrow P8_{reverse}(K)$ // Reverse
    $K \leftarrow LS\text{-}2(K)$
    $K \leftarrow P8(K)$
  Encryption(K_1) :
    $P \leftarrow F_k(K, P)$
    $P \leftarrow IP_{inverse}(P)$
return $C(c_7, ..., c_0) = P(p_7, ..., p_0)$

3.5 Quantum circuit implementation of `Simplified-AES`

We optimized `S-AES` as a quantum circuit to apply Grover key search on S-AES. Similarly, in [3], authors optimized `S-AES` for Grover key search. By implementing SBox and Mixcolumns, which require a lot of quantum resources, in `S-AES`, the quantum resources required for Grover search were significantly reduced compared to the results of [3]. Table 1 shows the comparison results. This section describes how `S-AES` is optimized as a quantum circuit.

Table 1: Quantum resources for `S-AES` implementation.

	Qubits	CNOT	CCNOT	CCCNOT	X	Depth
Key Expansion	40	568	192	-	10	-
Encryption	32	512	384	-	16	-
Total	72	1,080	576	-	26	-
Key Expansion (Ours)	**16**	**56**	**48**	**8**	**19**	**-**
Encryption (Ours)	**16**	**88**	**48**	**8**	**16**	**-**
Total (Ours)	**32**	**144**	**96**	**16**	**35**	**47**

AddRoundkey In AddRoundkey, the 16-bit round key is XORed on the 16-bit plaintext P. Since it is a simple structure using only XOR operation, AddRoundkey is designed only with CNOT gates. The quantum circuit for AddRoundkey is explained in Algorithm 2.

Algorithm 2 Quantum implementation of AddRoundkey.

Input: 16-qubit plaintext $P(p_{15}, ..., p_0)$, 16-qubit key $K(k_{15}, ..., k_0)$.
Output: 16-qubit plaintext $P(p_{15}, ..., p_0)$.
1: **for** $i = 0$ to 16 **do**
2: $p_i \leftarrow$ CNOT (k_i, p_i)
3: **end for**
4: **return** $P(p_{15}, ..., p_0)$

Substitution For the Substitution, using the SBox table method is inefficient in quantum computers. Because of the superposition state, all input values of the SBox exist as probabilities. We need to prepare an outcome for all cases. When implementing Substitution, the S-Box table method is not usually chosen as it is inefficient in all aspects of qubits, quantum gates, and depth.

In [3], authors implemented the substitution operation as a quantum circuit (i.e. not SBox table). However they used additional qubits and many quantum gates. On the other hand, we chose an implementation method based on the S-Box table of `S-AES`. Using the LIGHTER-R tool [4] to implement a 4-bit reversible S-Box, the S-Box was optimized and implemented as a quantum circuit without additional qubits as shown in Figure 10.

The LIGHTER-R is an extension of LIGHTER for classical computers, targeting quantum computers. The LIGHTER-R utilizes the Meet In The Middle (MITM) approach to design the compact algebraic normal form for classical computers. On the other hand, the LIGHTER-R can design algebraic normal form for 4-bit SBox table. Details of the LIGHTER-R tool are described in [4].

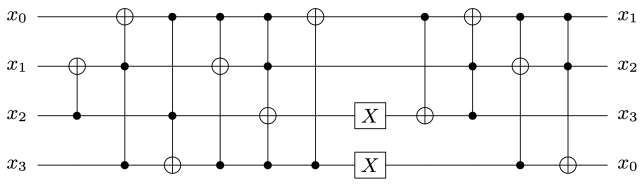


Fig. 10: Quantum implementation of SBox (`94ABD1856203CEF7`) for `S-AES`.

MixColumns In [3], 34 $CNOT$ gates were used for the MixColumns. We implemented MixColumns based on the final result computation. Through this,

we used only 16 $CNOT$ gates. Equation 6 is the result after performing a MixColumn of 8-bit. To generate Equation 6 values, we performed $CNOT$ gates with each other on 8-qubit (b_0,..., b_7), which can be expressed as 3. The 16-qubit plaintext is divided into 8-qubit to perform MixColumns. The proposed MixColumns implementation correctly generates all the resulting values. However, the resulting values are stored in different qubit locations. Therefore, after performing Algorithm 3, values of qubits must be changed by using $Swap$ gates.

$$\begin{array}{llll} p_0 \oplus p_6, & p_1 \oplus p_4 \oplus p_7, & p_2 \oplus p_4 \oplus p_5, & p_3 \oplus p_5 \\ p_2 \oplus p_4, & p_0 \oplus p_3 \oplus p_5, & p_0 \oplus p_1 \oplus p_6, & p_1 \oplus p_7 \end{array} \tag{6}$$

Algorithm 3 Quantum circuit for MixColumn.

Input: 8-qubit plaintext $P(p_0, ..., p_7)$.
Output: 8-qubit plaintext $P(p_0, ..., p_7)$.
1: $p_6 \leftarrow$ CNOT (p_0, p_6)
2: $p_3 \leftarrow$ CNOT (p_5, p_3)
3: $p_2 \leftarrow$ CNOT (p_4, p_2)
4: $p_7 \leftarrow$ CNOT (p_1, p_7)
5: $p_4 \leftarrow$ CNOT (p_7, p_4)
6: $p_5 \leftarrow$ CNOT (p_2, p_5)
7: $p_0 \leftarrow$ CNOT (p_3, p_0)
8: $p_1 \leftarrow$ CNOT (p_6, p_1)
9: **return** $P(p_0, p_1, ..., p_7)$

Keyschedule Figure 11 shows proposed Keyschedule process. We do not allocate qubits separately to store round keys in `S-AES`. Utilizing the on-the-fly method, the input key K is updated before AddRoundkey and used as a round key. To do this, we used the reverse operation. It temporarily creates a value for the round key in W_1 and W_3. After the XOR operation to create W_2 and W_4, the reverse operation is performed to return to the original value. By using this method, additional qubits for the round key are not allocated, and the SBox used in the Keyschedule is already optimized. Finally, since the round constant is known in advance, the XOR operation is performed using X gates, which is more efficient than CNOT gates on W, depending on where the bit of the constant is `1`.

4 Implementation results

For a data set consisting of n, the optimal number of Grover iterations is approximately $\sqrt{N}$. Therefore, the number of iterations of `Caesar` and `Vigenère`, which are search spaces for 4-qubit key, is 4, which is $\sqrt{2^4}$. However, the strictly optimal number of Grover iterations is $\lfloor \frac{\pi}{4}\sqrt{N} \rfloor$. Therefore, the optimal number of iterations for `Caesar` and `Vigenère` is `3`. If `4` iterations are performed, the probability of observing the solution is lower due to overfitting.

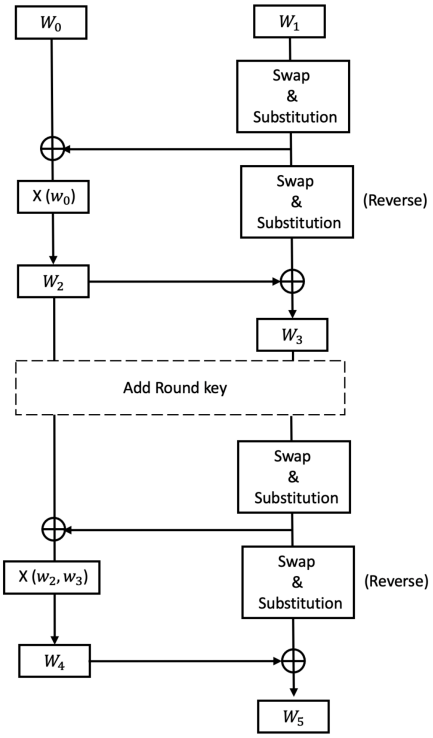


Fig. 11: S-AES keyschedule.

In the case of S-DES, the optimal number of iterations is 32 when calculated with $\sqrt{2^{10}}$ because the key search space is 10-qubit, and 25 with $\lfloor \frac{\pi}{4}\sqrt{2^{10}} \rfloor$. The results according to these two iteration numbers are shown in Figure 15 and Figure 16. In case 25, the solution key is recovered with a very high probability of 99.9%.

4.1 Quantum key search on Caesar

Figure 12 shows the results of a quantum key search for Caesar performed in the environment below.

- **Required qubits**: 13
- **Plaintext-Ciphertext**: 0x2F - 0x52
- **Target key value**: 0x3
- **Grover iterations**: 3
- **Key recovery probability** : 95.6%

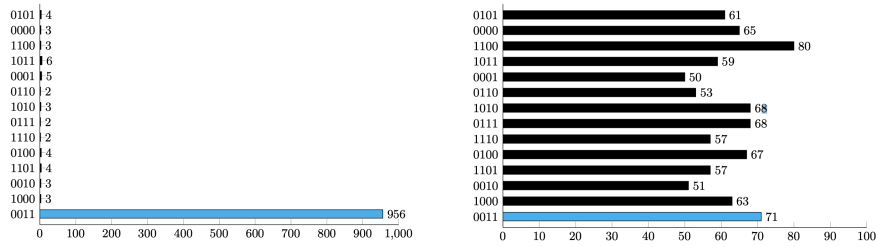


Fig. 12: (a) `Caesar` simulation (left). (b) `Caesar` Cloud (right).

4.2 Quantum key search on `Vigenère`

Figure 13 shows the results of a quantum key search for `Vigenère` performed in the environment below.

- **Required qubits**: 17
- **Plaintext-Ciphertext**: `0x42 - 0xCE`
- **Target key value**: `0x8C`
- **Grover iterations**: `3`
- **Key recovery probability** : `94.9%`

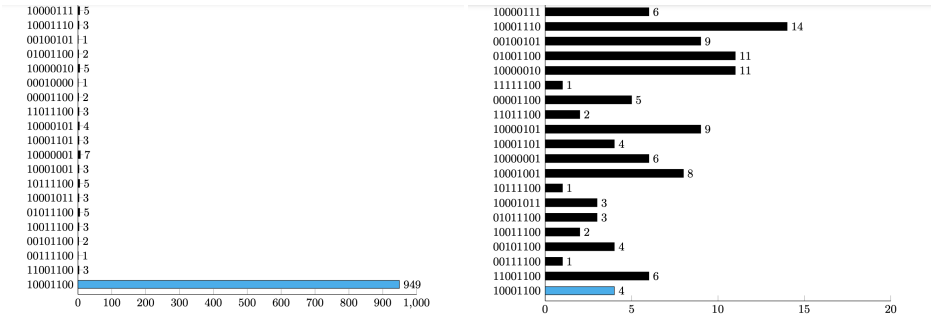


Fig. 13: (a) `Vigenère` simulation (left). (b) `Vigenère` Cloud (right).

4.3 Quantum key search on `Caesar` and `Vigenère` : Overfitted

Figure 14 shows that the probability of observing a solution is reduced due to overfitting with `4` iterations.

- **Grover iterations**: `4`
- **Key recovery probability for Caesar** : `58.4%`
- **Key recovery probability for Vigenère** : `32.4%`

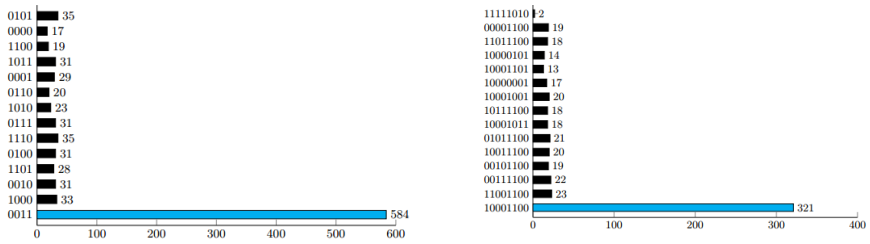


Fig. 14: (a) `Caesar` simulation (left). (b) `Vigenère` simulation (right).

4.4 Quantum key search on `S-DES`

Figure 15 shows the results of a quantum key search for `S-DES` performed in the environment below.

- **Required qubits**: 18
- **Plaintext-Ciphertext**: `0x10 - 0x33`
- **Target key value**: `0b1100010011`
- **Grover iterations**: `32`
- **Key recovery probability** : `83.6%`

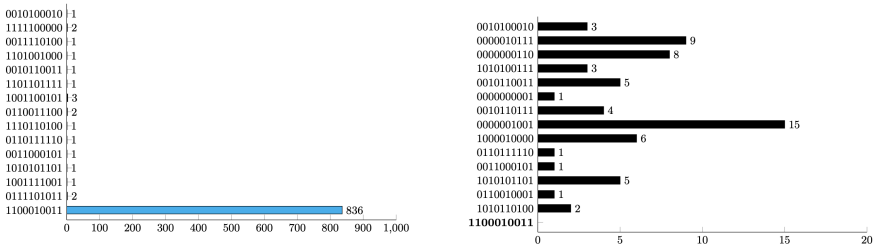


Fig. 15: (a) `SimpleDES` Simulation (left). (b) `SimpleDES` Cloud (right).

Figure 16 shows the results of a quantum key search for `S-DES` performed in the environment below.

- **Required qubits**: 18
- **Plaintext-Ciphertext**: `0x10 - 0x33`
- **Target key value**: `0b1100010011`
- **Grover iterations**: `25`
- **Key recovery probability** : `99.9%`

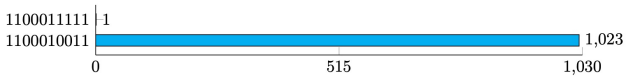


Fig. 16: `SimpleDES` Simulation (iteration: 25).

We performed quantum key search for both the simulator and the real quantum processor using cloud service. Comparing results for two different environments, the simulator recovers the target key, but the real quantum processor does not recover the target key. According to mentors' advice, it was concluded that quantum processors and cloud services still need development. There are algorithms for the noise control, and this is our future work. However, as it can be seen from the simulator results, the theory that quantum key search can quickly recover the secret key with high probability is established.

5 Conclusion

Quantum key search using Grover's algorithm is theoretically established. This is evidenced by the results of the simulator. However, it is still difficult to control the noises in real quantum processors. Quantum attacks using real quantum processors provided by cloud services were not successful. So fortunately, the cryptographic algorithms we use today are safe from attacks by quantum computers at the current development level. However, the cryptography community must prepare for the rapidly developing quantum computer. NIST (National Institute of Standards and Technology) is already in the process of standardizing the PQC (Post-Quantum Cryptography) algorithm. We will also focus on developing quantum computers and study their impacts on cryptography.

References

1. S. Cuccaro, T. Draper, S. Kutin, and D. Moulton, "A new quantum ripple-carry addition circuit," 11 2004.
2. D. Denisenko and M. Nikitenkova, "Application of Grover's quantum algorithm for SDES key searching," *Journal of Experimental and Theoretical Physics*, vol. 128, pp. 25–44, 01 2019.
3. M. Almazrooie, R. Abdullah, A. Samsudin, and K. Mutter, "Quantum Grover attack on the Simplified-AES," pp. 204–211, 02 2018.
4. V. A. Dasu, A. Baksi, S. Sarkar, and A. Chattopadhyay, "LIGHTER-R: Optimized reversible circuit implementation for sboxes," in *2019 32nd IEEE International System-on-Chip Conference (SOCC)*, pp. 260–265, 2019.