Grover on Quantum Cryptanalysis

Dr. Dhiman Saha[†], Gopal Ramesh Dahale

11840520, IIT Bhilai, gopald@iitbhilai.ac.in

Abstract. Grover's search algorithm promises to recover a block cipher's key in $O(\sqrt{N})$ calls to quantum oracle where N is the key search space. To apply Grover's search on a block cipher, one needs to implement it in a quantum circuit. This paper studies implementation of quantum circuits of hardware as well as software efficient block ciphers and Grover's attack on them. Specifically, we study SAES, SIMON 2n/mn, PRESENT, and AES-128 and optimize the count of qubits. Based on the submission requirements of NIST, we investigate the cost of quantum key search attacks with a depth constraint and offer ways for reducing oracle depth while requiring more qubits for AES-128.

We provide python implementation of Grover's Oracle for SAES as part of this work and quantum resource estimates. We test on IBMQ simulators and verify the results.

Keywords: Quantum Cryptanalysis · Grover's Algorithm · Quantum Simulation · AES · Qiskit · PRESENT · SIMON · SAES · Resource estimates

1 Introduction

Shor's algorithm was successful in factorizing a large number N which shows its high impact on asymmetric key cryptography. The impact of quantum computers on symmetric-key cryptography is still unclear. Grover's algorithm promise to find a number in a list size n in order of $2^{n/2}$ steps. To apply Grover's algorithm to a cipher for finding its key, the cipher has to be efficiently implemented as a quantum circuit. There are two types of symmetric encryption schemes: those that are suitable for hardware implementation and those that are suitable for software implementation. Ciphers like ChaCha[Ber08], SPECK[Bea+13] etc. are software efficient whereas ciphers like PRESENT[Bog+07], SIMON[Bea+13] etc. works effectively in hardware implementations.

In [Alm+18] and [Jan+21a], the authors implement a simplified version of AES as a quantum circuit. Similar to AES, SAES is based on the substitution-permutation design principle and is both software and hardware friendly[Sch+00]. To optimize a quantum circuit on a quantum computer, one must optimize the number of qubits, the count of quantum gates, and the depth of the circuit. Authors of [Alm+18] use 64 qubits with 8 ancilla qubits for implementing SAES whereas authors of [Jan+21a] have optimized on it and has used 32 qubits.

[AMM20] design quantum circuits for all variants of SIMON cipher which is a lightweight block cipher optimized for the hardware. They show that the count of qubits needed for Grover's Attack on SIMON 2n/mn is $O(mn + 2nr)$ where r is the count of chosen plaintext-ciphertext pairs. [Jan+21b] presents an optimal quantum circuit design for PRESENT and GIFT lightweight ciphers. They conclude that to efficiently implement a quantum circuit of a cipher, one must optimize the implementation on Sbox.

NIST[16] proposed requirements for the post-quantum cryptography process in which they upper-bounded the maximum depth of a quantum circuit denoted by MAXDEPTH as follows:

[†]Supervisor, IIT Bhilai, dhimans@iitbhilai.ac.in

- 2^{40} logical gates (the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year).
- 2^{64} logical gates (the approximate number of gates that current classical computing architectures can perform serially in a decade).
- 2^{96} logical gates (the approximate number of gates that atomic-scale qubits with the speed of light propagation times could perform in a millennium).

There exists no bound on width. [Jaq+20] study Grover's Attack on AES and LowMC by designing these block ciphers as quantum circuits under MAXDEPTH. They use the parallel version of Grover's Search Algorithm. They used the metrics of gate count and depth-width product for optimizing their design constrained to MAXDEPTH.

Other works include [KHJ18] who developed a framework to estimate the space-time complexity of quantum search algorithms in terms of qubits and depth. Using this framework they evaluate the security of AES-128 and SHA-2. They conclude that the cost of quantum circuits can be estimated in terms of Toffoli gates and the count of qubits.

[BHT06] described an algorithm for finding claws in pair of functions. This algorithm is an extension to the BHT algorithm [BHT97]. As pointed out in [Ber17], it suffers from a similar problem as BHT and does not take into account the hardware cost (QRAM).

This paper is organized in the following sections: Section 2 describes SAES in brief followed by section 3 which describes the work of [Alm+18]. Section 4 shows the optimizations made by [Jan+21a] on [Alm+18]. Section 5 briefly describes the SIMON cipher followed by section 6 which discusses the work of [AMM20]. Section 7 studies PRESENT cipher and section 8 shows the quantum circuit of PRESENT by [Jan+21b] and compare the work with [AMM20]. Work of [Jaq+20] is briefly discussed in section 9 and 10. Section 11 concludes the paper.

2 SAES

SAES is a nibble-oriented algorithm and has a block and key size of 16 bits. Figure 1 shows the block diagram of SAES. SAES uses two rounds. The main operations are: sub nibbles, shift rows, mix columns and add key. Similar to AES, SAES does not have a Mix Column in the last round.

$$\underbrace{b_0b_1b_2b_3}_{S_0}\underbrace{b_4b_5b_6b_7}_{S_1}\underbrace{b_8b_9b_{10}b_{11}}_{S_2}\underbrace{b_{12}b_{13}b_{14}b_{15}}_{S_3} = \begin{bmatrix} S_0 & S_2 \\ S_1 & S_3 \end{bmatrix} = State$$

As shown above, the initial state of SAES is captured using a 2 by 2 matrix, each element of which is a nibble.

2.1 Sub Nibbles

The sub nibbles operation takes a nibble as input and gives a nibble as output. The Sbox is computed as follows for an input x

1. Compute the multiplicative inverse x i.e. $y = x^{-1}$ in $GF(2^4)$.
2. The result of the Sbox is computed using the follows operation:

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{1}$$

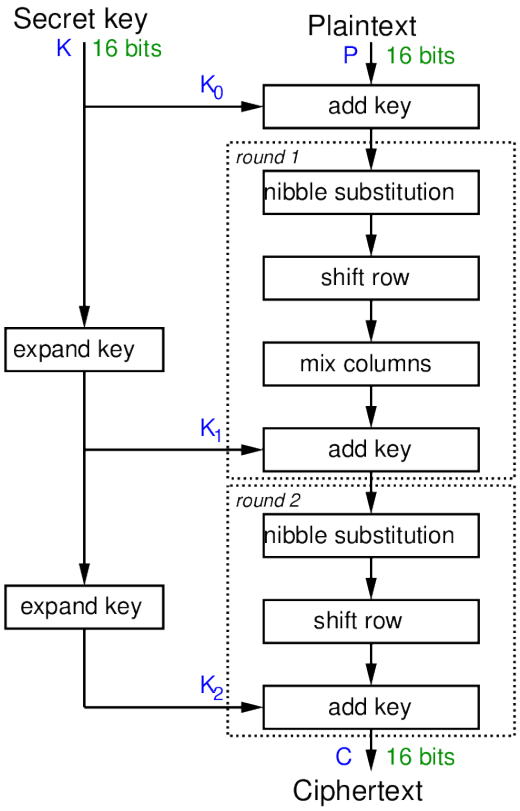


Figure 1: SAES encryption [Gor22]

The final Sbox is the given below:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$Sbox(x)$	9	4	A	B	D	1	8	5	6	2	0	3	C	E	F	7

2.2 Shift Rows

The Shift Rows operation is the same as AES.

$$\begin{bmatrix} S_0 & S_2 \\ S_1 & S_3 \end{bmatrix} \longrightarrow \begin{bmatrix} S_0 & S_2 \\ S_3 & S_1 \end{bmatrix}$$

2.3 Mix Columns

SAES work on the ring $\mathbb{F}_2[x]/(x^4 + x + 1)$ and Mix Columns treats 1 byte (1 column of the state matrix) as an element in $GF(2^4)$ with ring $\mathbb{F}_{2^4}[x]/(x^2 + 1)$. The transformation is as follows:

$$\begin{bmatrix} S'_0 \\ S'_1 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} S_0 \\ S_1 \end{bmatrix}$$

The elements of the matrix are in $\mathbb{F}_{2^4}[x]/(x^2 + 1)$.

2.4 Key Expansion

The master key (16 bit) can be thought as 2 bytes B_0B_1. SAES uses 3 keys in one full encryption. The first key is the master key. The first round key B_2B_3 and the second round key B_4B_5 are expanded from the master key using the following algorithm:

KEY EXPANSION FOR SAES(K)

$keys = [B_0, B_1, B_2, B_3, B_4, B_5]$
$keys[0] = K[0 \dots 8]$
$keys[1] = K[8 \dots 16]$
for $i = 2$ **to** 5
    if $i\%2 == 0$
        $keys[i] = keys[i-2] \oplus RCON(i/2) \oplus Sbox(RotNib(keys[i-1]))$
    else
        $keys[i] = keys[i-2] \oplus keys[i-1]$
return B_0B_1, B_2B_3, B_4B_5

The $RotNib$ function rotates the two nibbles of a byte. $RCON$ is the round constant function and is defined as

$$RCON(i) = (x^{i+2}||0000)$$

For example $RCON(1) = 10000000$, $RCON(2) = 00110000$.

3 Quantum SAES18

We now describe the design of the quantum circuit for SAES as proposed by [Alm+18] and call it QSAES18 as it was published in the year 2018. The proposed design is shown in Figure 2. PTXT denotes plaintext, SN denotes sub nibbles, MC denotes mix columns and RC denotes round constant. Initially, key is added to the first 16 qubits followed by the addition of plaintext.

3.1 Working of the circuit

We briefly describe the working of the quantum circuit. $|\psi_i\rangle$ denote the state at a slice in the circuit.

3.1.1 Initialisation

At state $|\psi_1\rangle$, the qubits are initialized with the master key and then XORed with the plaintext.

3.1.2 Round 1

The first four nibbles are passed through the Sbox for the sub-nibbles operation. For the shift row operation, they are mapped to other nibbles. Specifically

$$N_0 \longrightarrow N_{10}$$
$$N_1 \longrightarrow N_9$$
$$N_2 \longrightarrow N_8$$
$$N_3 \longrightarrow N_{11}$$

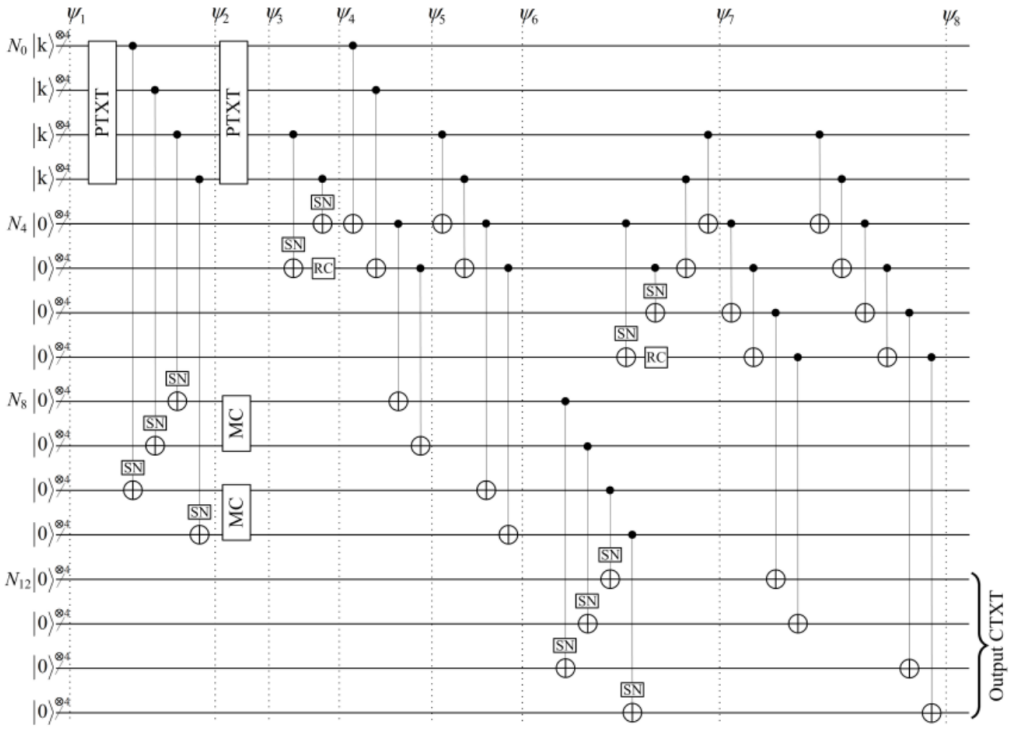


Figure 2: QSAES18 [Alm+18]

$$\begin{bmatrix} S_0 & S_2 \\ S_1 & S_3 \end{bmatrix} \longrightarrow \begin{bmatrix} S_0 & S_2 \\ S_3 & S_1 \end{bmatrix}$$

The state after shift rows is at $|\psi_2\rangle$. Note that the plaintext is again XORed with the first 4 nibbles at $|\psi_2\rangle$ to recover the master key.

After the shift rows operation, mix columns will operate on the the two columns $\begin{bmatrix} S_0 \\ S_3 \end{bmatrix}$ and $\begin{bmatrix} S_2 \\ S_1 \end{bmatrix}$. S_0, S_3 are on nibble N_{10}, N_{11} and S_2, S_1 are on nibble N_8, N_9. Therefore the two mix column gates are placed accordingly.

3.1.3 Round 2

Similar to round 1, the state after sub nibbles and shift rows is at $|\psi_7\rangle$. The key expansion part is continued by the top qubits and follows the same method as explained previously. The final ciphertext is captured at $N_{12}, N_{13}, N_{14}, N_{15}$ at state $|\psi_8\rangle$.

We now describe the implementation of each operation

3.2 Sub Nibbles

As per equation 1, we need to find the multiplicative inverse of the input x. The authors use the Fermat inversion algorithm which is the square and multiply method to find multiplicative inverse in $GF(2^4)$. The theorem states that the multiplicative inverse of an element $x \in GF(2^4)$ can be calculated as follows:

$$x^{-1} = x^{2^4-2} = x^{16-2} = x^{14} = x^2 \times (x^2)^2 \times ((x^2)^2)^2$$

The above equation has 2 multiplication operations and 3 squaring operations. The authors use a quantum multiplier in $GF(2^4)$ from [Che+08]. The quantum circuit for the multiplier is shown in Figure 3.

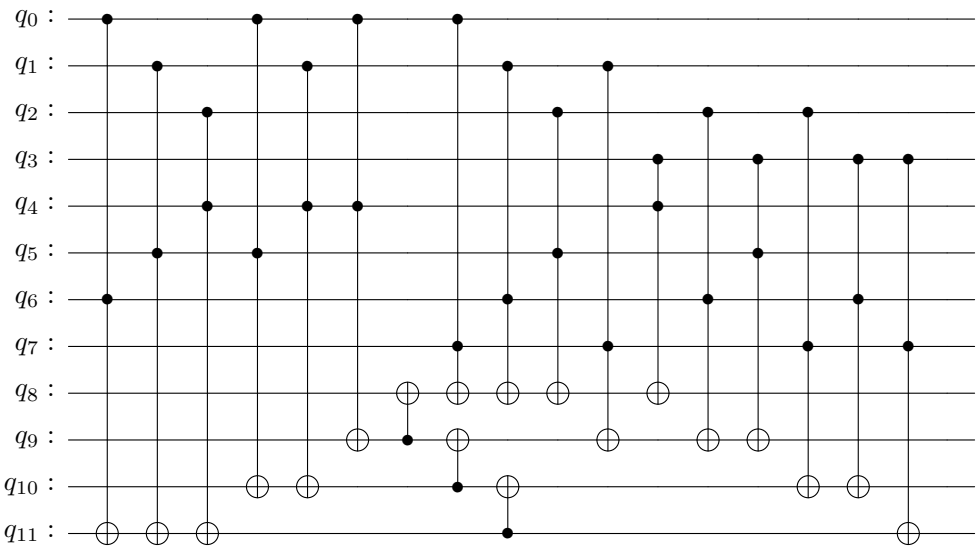


Figure 3: Multiplier

The first 8 qubits are input to the multiplier circuit. $q_0 - q_3$ is the first vector and $q_4 - q_7$ is the second vector. The output is taken at the last 4 qubits $q_8 - q_{11}$.

The squarer circuit can be expressed by using the following equation for some input x.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{2}$$

For the squarer circuit, the authors use the CNOT synthesis algorithm to decompose the matrix of equation 2. The squarer circuit is shown in Figure 4.

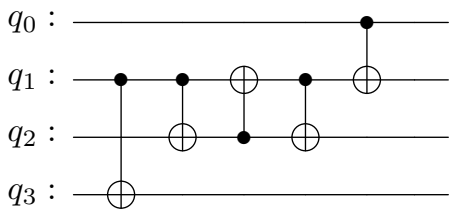


Figure 4: Squarer

For the affine transformation (equation 1), the affine matrix is decomposed to remove the need for ancilla qubits. Figure 5 shows the circuit for affine transformation.

The complete Sbox circuit is shown in Figure 6. The input is taken from the first 4 qubits and the output is taken at the last 4 qubits. 8 ancilla qubits are used.

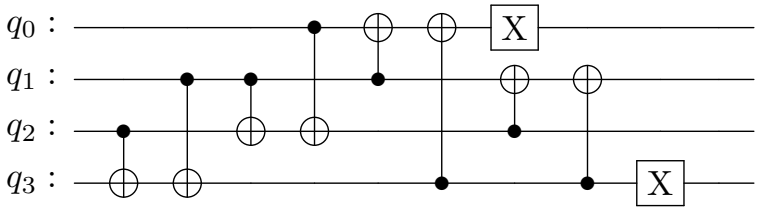


Figure 5: Affine transformation

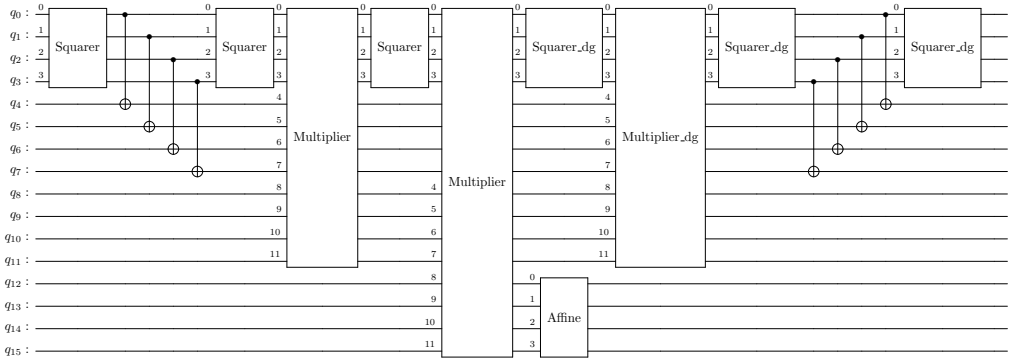


Figure 6: Sbox

ff

3.3 Mix columns

CNOT synthesis algorithm is used to remove the need for ancilla qubits by decomposing the mix column matrix. Figure 7 shows the circuit for mix column.

o

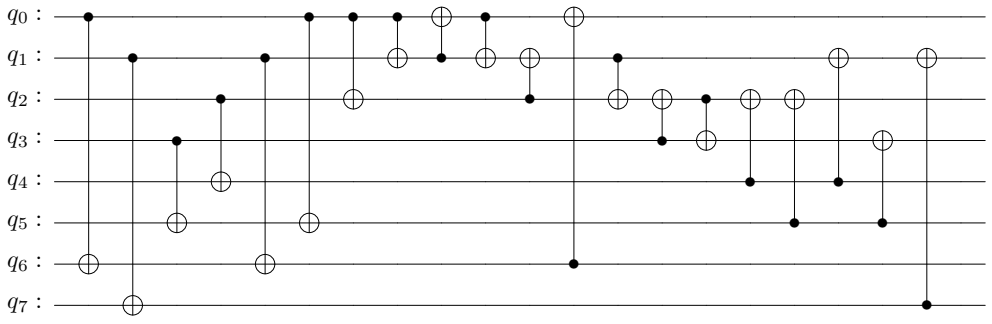


Figure 7: Mix column

The circuit takes 1 byte (1 column of the state matrix) and outputs the corresponding matrix multiplication in $GF(2^4)$.

3.4 Round Constants

3.5 Complexity Analysis

The work was focused on reducing the number of qubits and since the number of gates used is inversely proportional to the number of qubits, the circuit depth is bound to increase. Still, the quantum circuit uses a polynomial number of gates. The cost for implementing SAES is shown in Table 1:

Table 1: Cost for QSAES18 [Alm+18]

	Qubits	X	CX	CCX	Ancilla
Key expansion	32	10	568	192	8
Encryption	32	16	512	384	-
Total	64	26	1080	576	8

3.6 Grover's Attack

To perform Grover's attack, a reversible circuit of the block cipher is needed. The circuit shown in Figure 2 is only for encryption. To create a reversible circuit, all the operations need to be implemented in reverse order. This will double the cost of Table 1. The boolean function for Grover's oracle can be defined as follows:

$$f(k) = \begin{cases} 1 & SAES(k,p) = c \\ 0 & else \end{cases}$$

k is the key which encrypts p to c using $SAES$ encryption oracle. Figure 8 shows the one iteration of Grover's attack on SAES. At $|\psi_1\rangle$, we have all the qubits in a superposition state which describes all possible keys. The encryption is completed at $|\psi_2\rangle$ and if the ciphertext obtained from the oracle matches with the actual ciphertext, then the target qubit (oracle qubit in the figure) is flipped (this logic is implemented by placing the actual ciphertext in after the encryption oracle and flipping the oracle qubit using unbounded Toffoli gate [Qis22a]). At $|\psi_3\rangle$ we complete the reversible circuit of SAES and at $|\psi_4\rangle$ we obtain the state after Grover's Diffusion Operator.

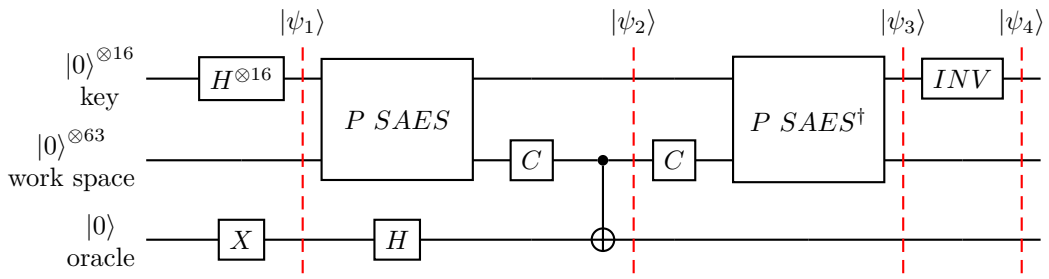


Figure 8: Grover's Attack on SAES

To calculate the number of iterations required for a successful Grover's Attack, we use the following equation:

$$t = \frac{\pi}{4}\sqrt{\frac{2^k}{s}} \tag{3}$$

k is the size of the key and therefore 2^k is the total key space. s is the total number of solutions possible. More than one key can encrypt the same plaintext to the same ciphertext. The authors chose $r = 2$, where r is the number of plaintext-ciphertext pairs. This guarantees to find the unique key. Plugging the values we get:

$$t = \frac{\pi}{4}\sqrt{\frac{2^{16}}{2}} = 142$$

They propose a modified version of Grover's Attack to find the unique key which is shown in Figure 9.

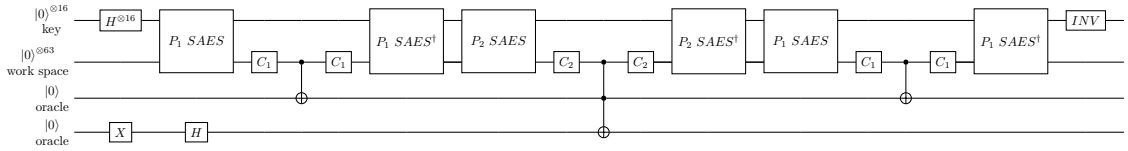


Figure 9: Grover's Attack to find unique key with r = 2

The corresponding boolean function is described as follows:

$$f(k) = \begin{cases} 1 & (SAES(k, p_1) = c_1) \wedge (SAES(k, p_2) = c_2) \\ 0 & else \end{cases}$$

Using the circuit shown in Figure 9, the authors were able to successfully run Grover's Attack on SAES. For the plaintext 0110 1111 0110 1011 and ciphertext 0000 0111 0011 1000, the actual key is 1010 0111 0011 1011. We verified this classically using a python program [Dah22].

4 Quantum SAES21

We now describe the design of the quantum circuit for SAES as proposed by [Jan+21a] and call it QSAES21 as it was published in the year 2021. This version is optimized compared to [Alm+18] as it only uses 32 qubits and no ancilla qubits. The proposed design is shown in Figure 10.

The top 16 qubits are used for storing the master key and for the process of key expansion. The bottom 16 qubits are used for storing plaintext, round operations, and outputting ciphertext.

4.1 Sub Nibbles

Unlike [Alm+18] which uses 16 qubits (4 input, 4 output, and 8 ancillae) for Sbox computation, [Jan+21a] uses only 4 qubits using LIGHTER-R tool [Das+19]. The output of the quantum circuit is a permutation of the bits of the actual Sbox output and therefore SWAP gates are used which are not measured in the overall quantum resources used in QSAES21. Figure 11 shows the quantum circuit for Sbox.

We also need to implement the inverse Sbox operation which will be required during reversible SAES. Figure 12 shows the quantum circuit of inverse Sbox.

4.2 Mix Columns

Similar to [Alm+18], the authors use 8 qubits to perform mix columns operation on 1 column of the state matrix. Figure 13 shows the quantum circuit for mix column operation on 1 byte (1 column) of the state matrix. Compared to [Alm+18], the authors use less

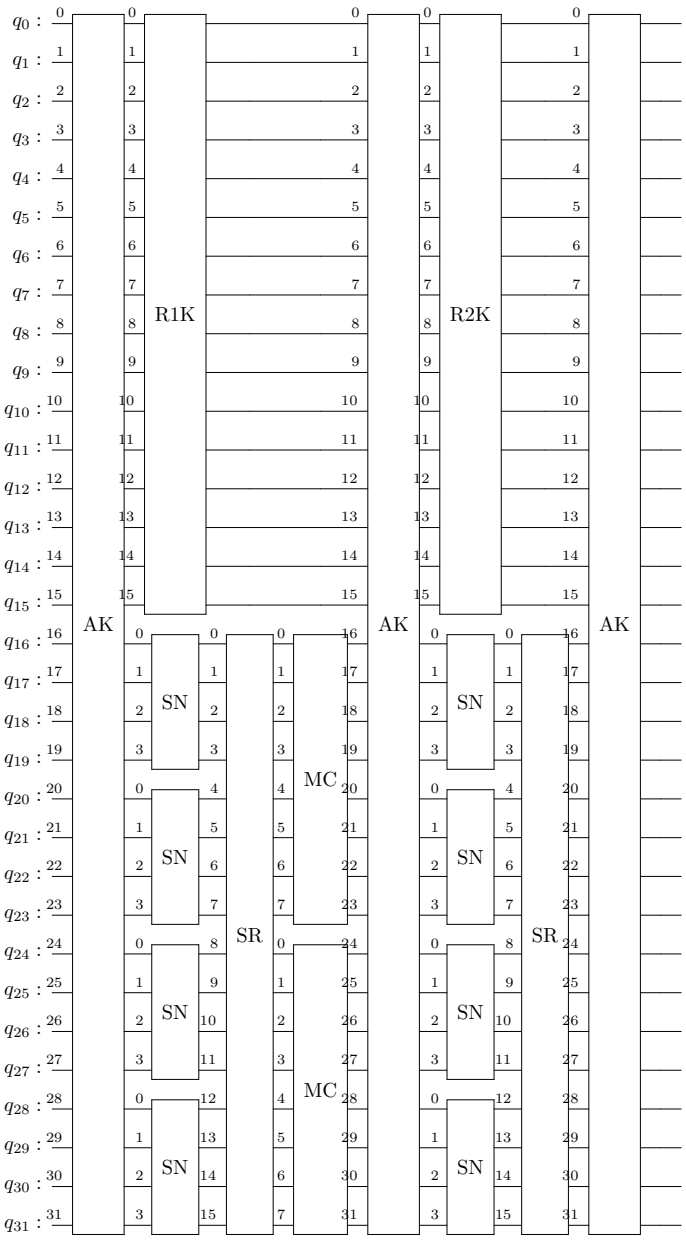


Figure 10: QSAES21

number of qubits but require SWAP gates to get the correct result. Since SWAP gates are not counted in the quantum resources, the quantum circuit for the mix column by [Jan+21a] is more optimal compared to [Alm+18].

4.3 Shift Rows

The authors implemented shift rows operation using swap gates only whereas [Alm+18] used extra qubits for that with additional CNOT gates. The circuit for shift rows is shown in Figure 14a.

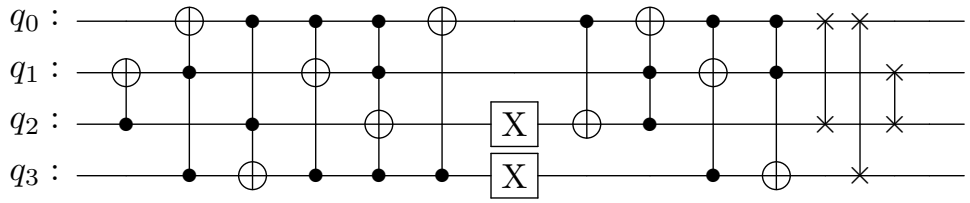


Figure 11: Sbox

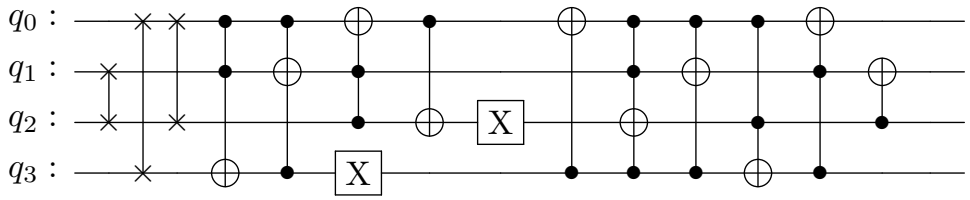


Figure 12: Inverse Sbox

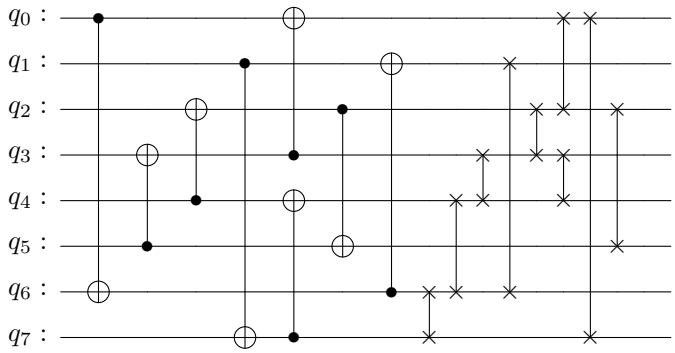


Figure 13: Mix column

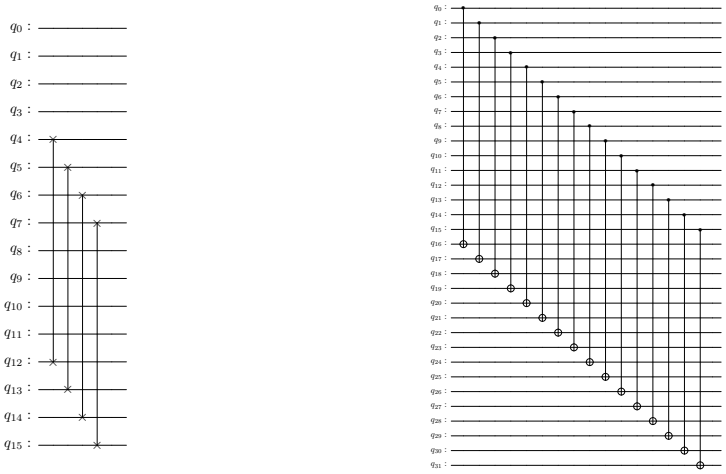
4.4 Add Round Key

Add round key can be implemented simply using 16 CNOT gates as shown in Figure 14b

4.5 Key Expansion

Round keys are generated on the fly. Figure 15 shows the circuit for generating the first round key from the master key.

Referring to the key expansion algorithm 2.4, for the first round key, we perform swap and substitution of B_1 (SWAP gate and Sbox gate) and then xor it with B_0 (CNOT gates). Then xor is performed with the round constant (single not gate as the first round key is 10000000). Now the first 8 qubits hold B_2. To get B_3, we need to xor B_1 and B_2. Since the operations on the bottom 8 qubits result in the loss of B_1, we need to perform inverse swap and substitution on the lower 8 qubits to get back B_1. Then we XORed B_2 with B_1



(a) Shift rows

(b) Add round key

Figure 14: Shift rows and Add round key

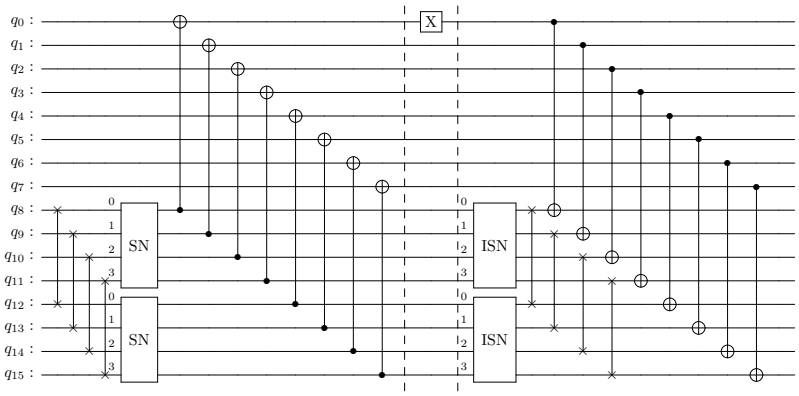


Figure 15: Round key 1

to get B_3.

Therefore the top 8 qubits hold B_2 and the bottom 8 qubits hold B_3.

Similarly Figure 16 shows the circuit for generating the second round key with round constant as 01100000.

4.6 Evaluation

In all the experiments performed, the quantum circuit was transpiled with an optimization level 2. According to Qiskit docs, transpilation is the process of rewriting a given input circuit to match the topology of a specific quantum device, and/or to optimize the circuit for execution on present-day noisy quantum systems [Qis22b]. The optimization level defines how much optimization to perform on the circuits. Higher levels generate more optimized circuits, at the expense of longer transpilation time. * 0: no optimization * 1: light optimization * 2: heavy optimization * 3: even heavier optimization. [Qis22b]

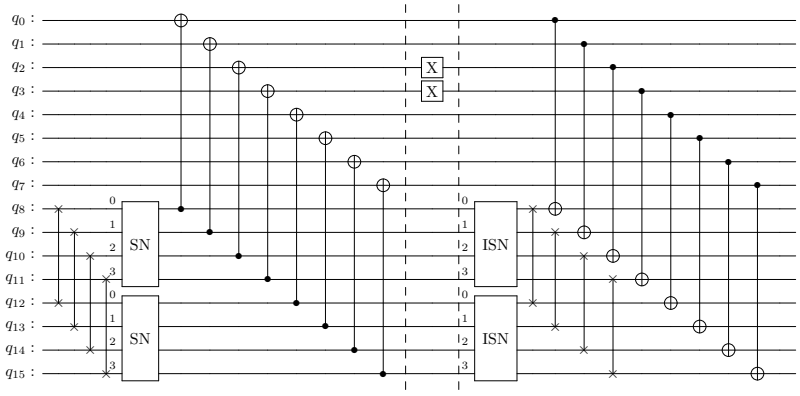


Figure 16: Round key 2

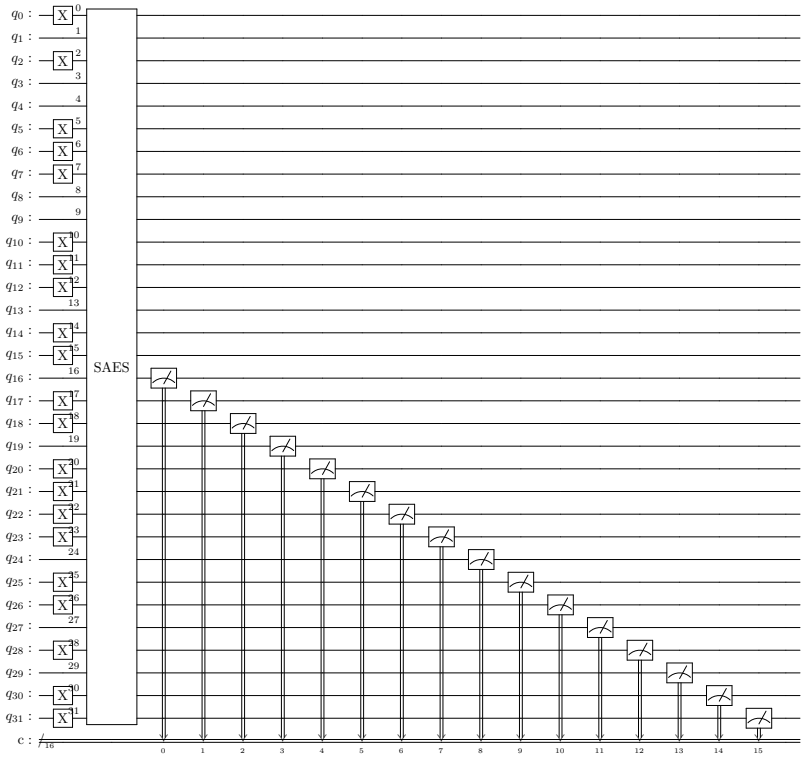


Figure 17: Verification of QSAES21

4.6.1 Experiment 1

In this experiment, we checked whether the quantum circuit for QSAES21 gives the same output as its classical counterpart. Figure 17 shows the circuit in which we initialize the qubits with a key 1010 0111 0011 1011 and plaintext 0110 1111 0110 1011 and obtained the ciphertext 0000 0111 0011 1000 which is the same as given by our implementation of the classical algorithm [Dah22]. The simulation was performed on IBMQ QASM Simulator [Qua21] and the results can be found at [Dah22]. Figure 18 shows the result obtained.

We can see that the output is $0x1ce0$ in hexadecimal, which in binary is 0001 1100 1110 0000 which is the reverse of the ciphertext (reverse because Qiskit's [ANI+21] measurement

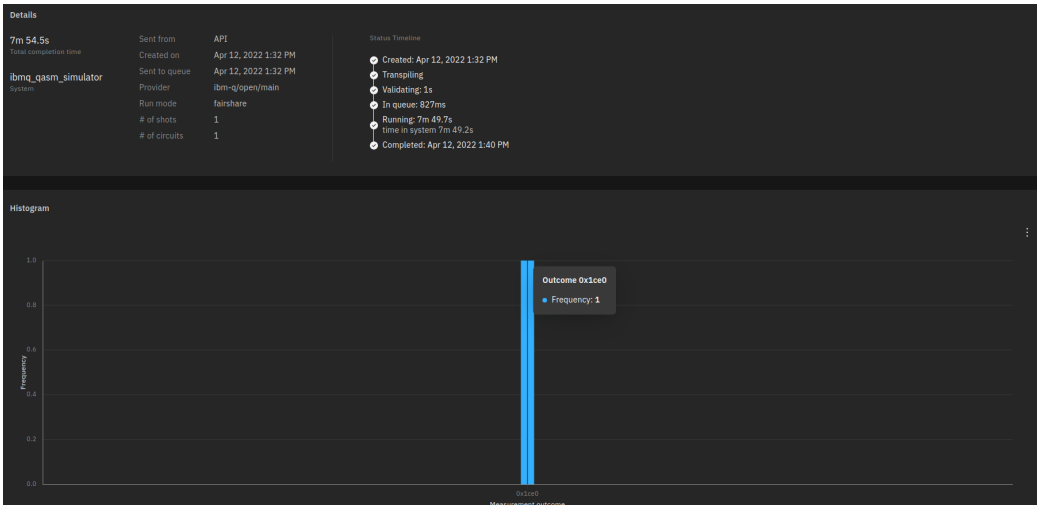


Figure 18: Output of encryption of plaintext 0110 1111 0110 1011 with key 1010 0111 0011 1011

shows the bottom qubit first).

4.6.2 Experiment 2

Figure 19 shows the quantum circuit for this experiment performed using IMBQ Statevector simulator [Qua21]. We put the key in a superposition of all states and fix the plaintext. The state vector simulator gives all possible states in the output. This will result in all possible key, ciphertext pairs. Ideally, we should run for 2^{32}, but the simulator gave a timeout error. We were able to run it for 4000 shots. Results can be found at [Dah22]. Figure 20 shows the result obtained.

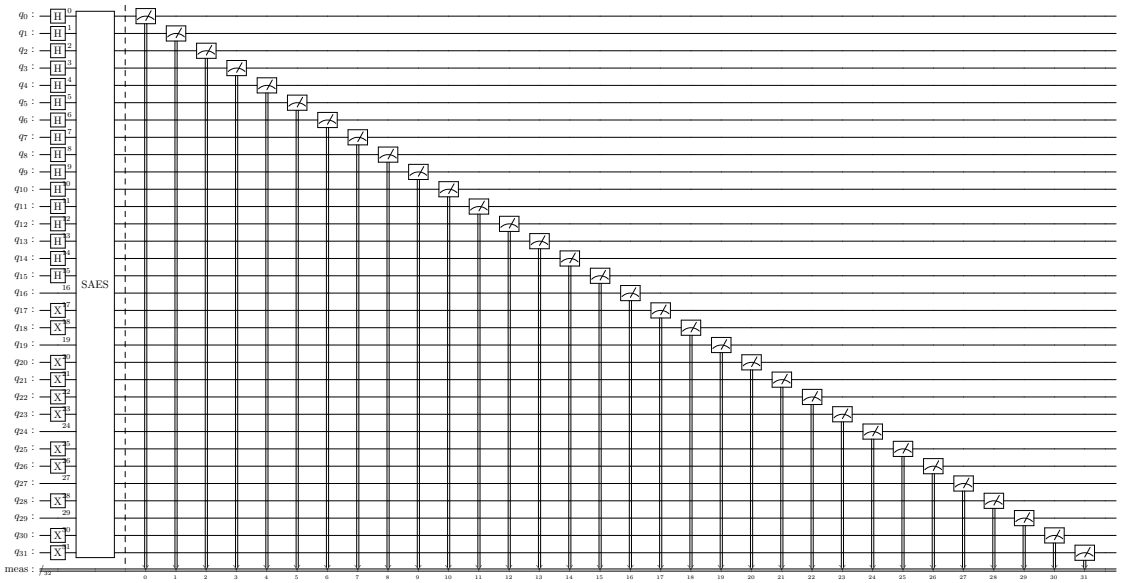


Figure 19: State vector simulation

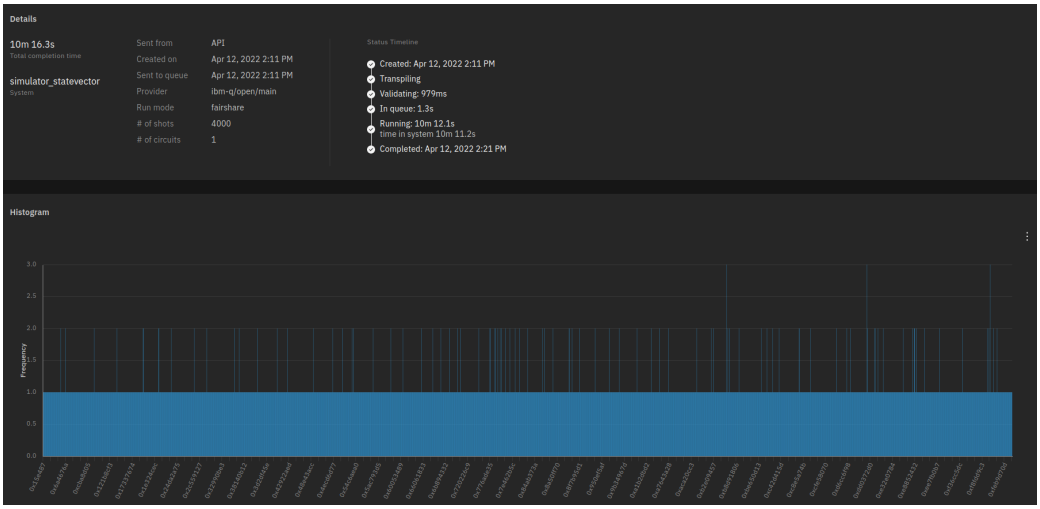


Figure 20: Keys in Superposition

We can see that in the output maximum states have frequency 1. Some states have frequencies 2 and even 3. This is possible has the superposition can collapse into any 2^{32} states.

4.6.3 Grover's Attack

For performing Grover's Attack, we need to create the oracle circuit. Figure 21a shows the oracle. The plaintext-ciphertext pair used is (0110111101101011, 0000011100111000). The oracle is the same as used by [Alm+18]. It consists of a SAES encryption circuit followed by the unbounded Toffoli gate [Qis22a]. Then the reversible circuit is used to undo the operation of SAES. Using this circuit, Grover's Attack circuit is designed as shown in Figure 22.

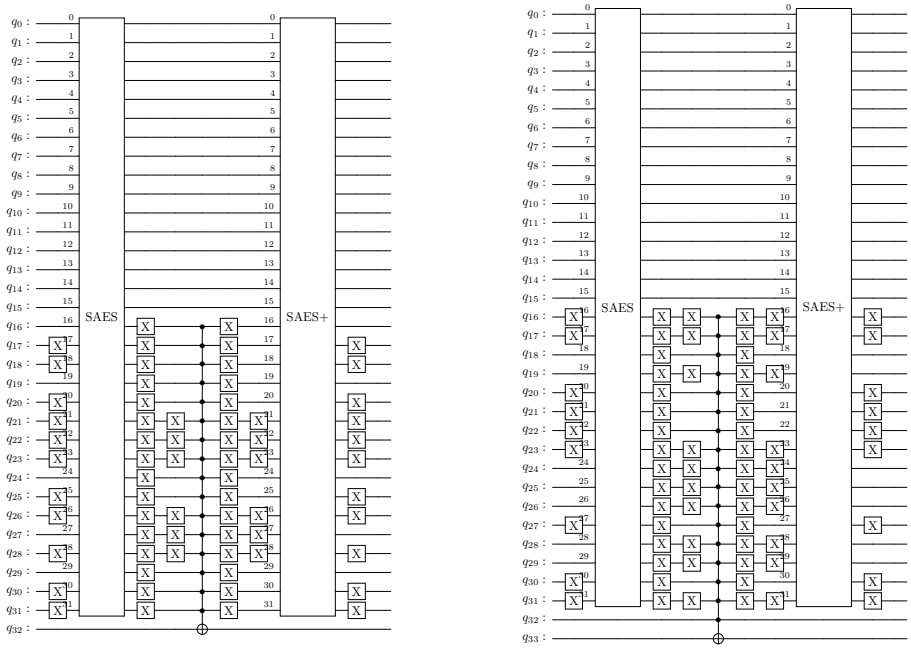
Since this will not find the unique key, another oracle is designed in which a different plaintext-ciphertext pair is used (0110111101101011,0000011100111000) whose circuit is shown in Figure 21b. The modified Grover's Attack circuit is shown in Figure 23 which is similar to Figure 9.

Table 2 compares the work of [Alm+18] and [Jan+21a]. We can see that the number of gates is reduced significantly.

Table 2: Comparison of cost for QSAES18[Alm+18] and QSAES21[Jan+21a]

	Qubits	X	CX	CCX	Ancilla
Key expansion	32	10	568	192	8
Encryption	32	16	512	384	-
Total	64	26	1080	576	8
Key expansion	16	19	56	48	-
Encryption	16	16	88	48	-
Total	32	35	144	96	-
My Code	32	35	144	96	-

The cost was heavily reduced due to optimizations in Sbox, key expansion, and mix columns circuit. The cost of a Grover's Attack can be calculated as follows. Since the



(a) Oracle 1

(b) Oracle 2

Figure 21: Oracles for Grover's Attack

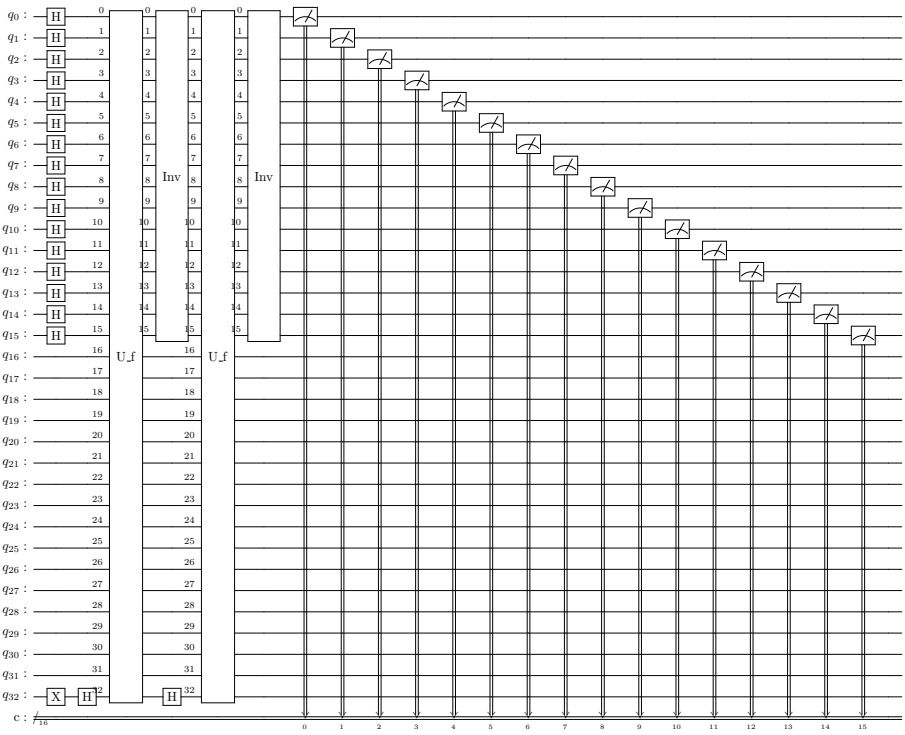


Figure 22: Grover's Attack (2 iterations)

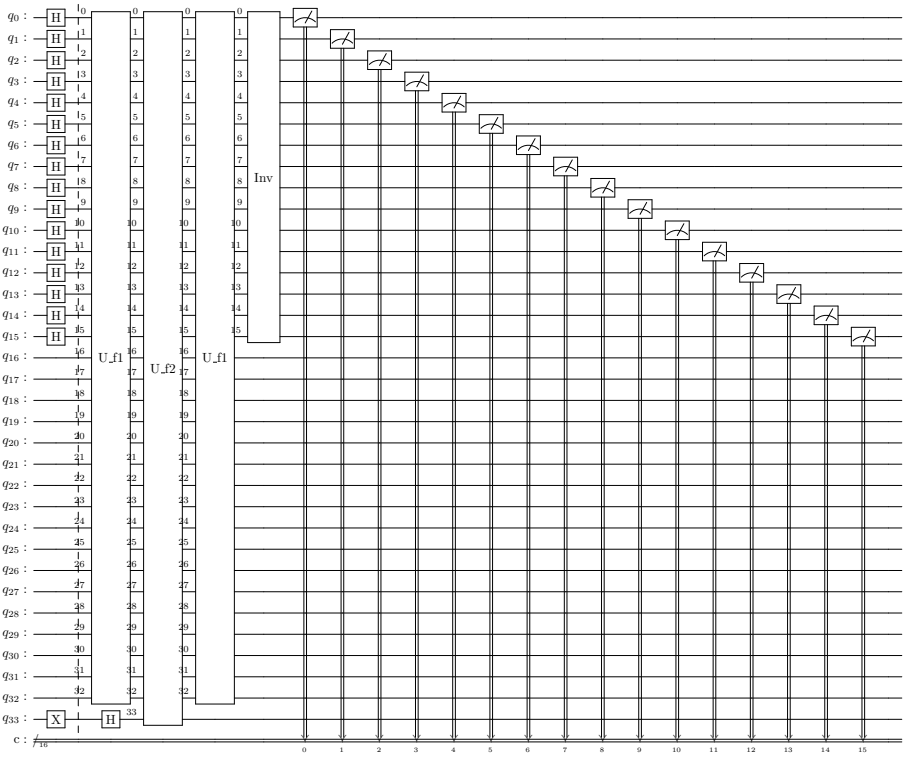


Figure 23: Grover's Attack for unique key with r = 2 (1 iteration)

encryption is performed twice (two plaintext-ciphertext pairs) and also the reversible circuit is added, the total cost of Grover's Attack to find a unique key is

$$2 \times 2 \times \frac{\pi}{4}\sqrt{2^{16}}$$

5 SIMON

[AMM20] discusses Grover's Attack on lightweight block cipher SIMON 2n/mn using $O(mn + 2nr)$ qubits where r is the count of chosen plaintext-ciphertext pairs. We briefly describe the SIMON cipher and then its quantum circuit. Figure 24 shows one round of the SIMON cipher. SIMON has 10 different block sizes shown in Table 3. In SIMON 2n/mn, 2n refers to the block size and mn refers to the key size.

The equation below describes one round of the SIMON cipher.

$$F(x, y) = (y \oplus (S^1(x) \wedge S^8(x)) \oplus S^2(x) \oplus k, x) \tag{4}$$

$S^j(x)$ denotes left circular shift by j bits. PT_1, PT_2 are also referred as L_i, R_i and CT_1, CT_2 are also referred as L_{i+1}, R_{i+1}. L_i, R_i are n bit strings as input to the i^{th} round and k is the round key.

The key expansion has 3 different methods which depend on the key size. For the first m rounds, the round keys are initialized from the master key. For the remaining T-m rounds, the round keys are generated using the function given below:

$$k_{m+i} = \begin{cases} c_i \oplus k_i \oplus S^{-3}(k_{i+1}) \oplus S^{-4}(k_{i+1}) & m = 2 \\ c_i \oplus k_i \oplus S^{-3}(k_{i+2}) \oplus S^{-4}(k_{i+2}) & m = 3 \\ c_i \oplus k_i \oplus S^{-1}(k_{i+1}) \oplus S^{-3}(k_{i+3}) \oplus S^{-4}(k_{i+3}) & m = 4 \end{cases} \quad (5)$$

c_i denotes the round constants.

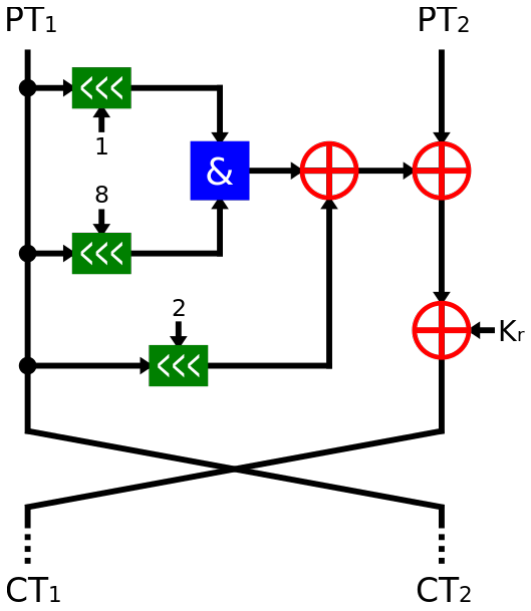


Figure 24: One round of SIMON [con22a]

Table 3: Parameters of SIMON cipher [AMM20]

Word size (n)	Block size (2n)	keywords (m)	Key size (mn)	Rounds (T)
16	32	4	64	32
24	48	3,4	72,96	36,36
32	64	3,4	96,128	42,44
48	96	2,3	96,144	52,54
64	128	2,3,4	128,192,256	68,69,72

6 Quantum SIMON

We now describe the design of the quantum circuit for SIMON as proposed by [AMM20] and call it QSIMON. Let the initial state be L_0, R_0. From equation 4, we can construct the path for i^{th} bit from R_0 which can be written as

$$R_2(i) = L_1(i) = R_0(i) \oplus (L_0(i+1)mod(n) \wedge L_0(i+8)mod(n)) \oplus L_0(i+2)mod(n) \oplus k_0$$

In quantum circuit, after initializing L_0, R_0, we can use the qubits of R_0 for storing L_1 and L_0 becomes R_1. This process can be continued for further rounds. This eliminates the need for ancilla qubits.

For computing $R_2(i)$, we can use the following operations:

1. CCX$(L_0(i+1)mod(n), L_0(i+8)mod(n), R_0(i))$
2. CX$(L_0(i+2)mod(n), R_0(i))$
3. CX$(k_0(i), R_0(i))$

The same process can be followed for L_2 and then for other rounds. Calculating the number of gates for one round of SIMON, we can see that we need n Toffoli gates and $2n$ CNOT gates for a single round. For j rounds, we need jn and $2jn$ Toffoli and CNOT gates respectively.

We define three functions that together form one round of the QSIMON as shown in Figure 25. Using these three functions we can create QSIMON. 2 round construction of QSMION is shown in Figure 26.

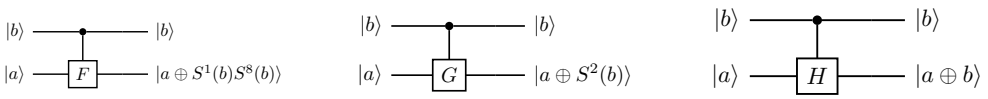


Figure 25: Subroutines for one round of QSIMON

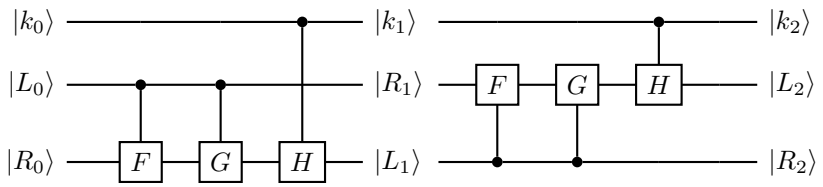


Figure 26: 2 Round of QSIMON

6.1 Key Expansion

Define a function as

$$R_q(a, b) = (S^{-i}(b) \oplus a, b)$$

This function will be used in the circuit of key expansion. The quantum circuit for this equation is shown in Figure 27

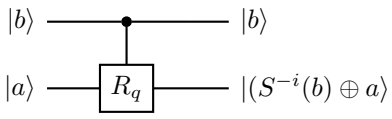


Figure 27: Quantum circuit for $R_q(a, b)$

Consider the case for $m = 2$. The key size is $2n$, so we have two keywords k_0, k_1 which are used in the first 2 rounds. For the rest of the rounds, round keys are derived from k_0, k_1. The third round key can be calculated using the following steps:

1. CX$(k_1(i-3)mod(n), k_0(i))$

2. CX$(k_1(i-4)mod(n), k_0(i))$
3. NOT$(k_0(i))$ if i^{th} bit of round constant is true

The above steps are illustrated in Figure 28

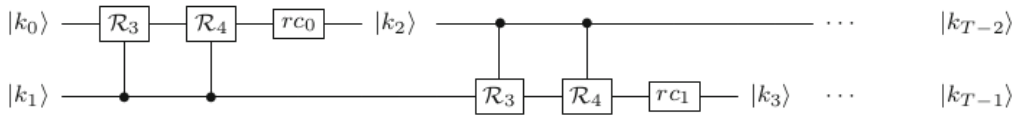


Figure 28: Key expansion for m = 2 [AMM20]

We can follow a similar procedure for $m = 3$ and 4. The quantum circuits for them is shown in Figure 29 and 30 respectively.

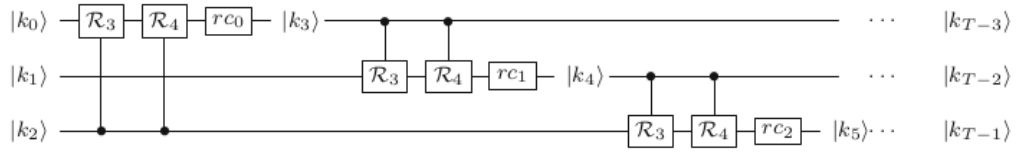


Figure 29: Key expansion for m = 3 [AMM20]

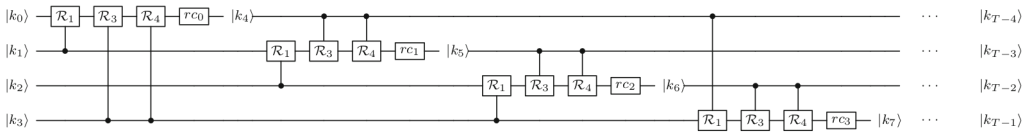


Figure 30: Key expansion for m = 4 [AMM20]

QSIMON for $m = 2$ is shown in Figure 31

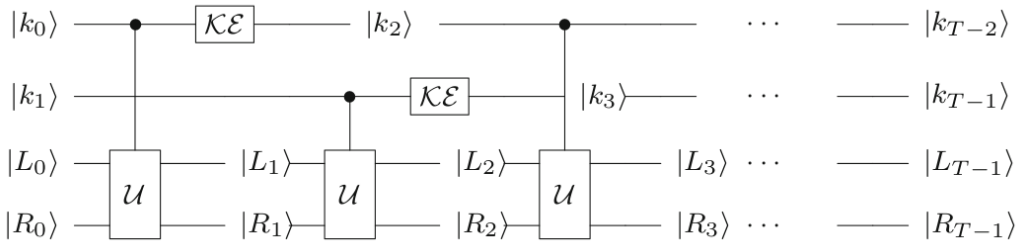


Figure 31: QSIMON for m = 2 [AMM20]

For 1 round we need 2n, 2n, and 3n CNOT gates for m = 2,3 and 4 respectively and n' NOT gates which depend on the count of true bits in round constant. For remaining T-m rounds, the key expansion procedure requires 2n(T-m), 2n(T-m), and 3n(T-m) CNOT gates for m = 2,3, and 4 respectively and n'(T-m) NOT gates.

6.2 Grover's Attack

We will discuss the implementation of Grover's Oracle for a successful Grover's Attack to obtain the unique key. If we have r chosen plaintext-ciphertext pairs then the Grover's

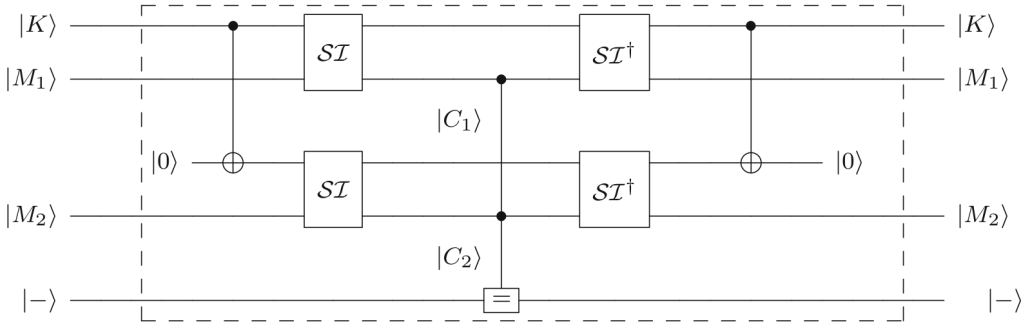


Figure 32: Grover's Attack on QSIMON [AMM20]

Oracle circuit can be implemented using r QSIMON oracles in parallel along with r QSIMON† oracles (reversibility). This implementation is shown in Figure 32

The authors were able to successfully recover the key for a round reduced version of SIMON. For finding a unique key we need precisely 2 plaintext-ciphertext pairs i.e. r = 2. Grover's Oracle compares $2n$ qubits (n qubits of C_1 and n qubits of C_2) using unbounded Toffoli gate [Qis22a]. This is when r = 2. In general, we require $2nr$ CNOT gates. mn qubits are needed for the master key. Therefore the total qubits complexity is of the order $O(mn + 2nr)$. The authors have provided tables that shows cost of each variant of SIMON [AMM20].

7 PRESENT

PRESENT[Bog+07] is an Ultra-lightweight block cipher and has a substitution permutation network. It has a block length of 64 bits and supports 80 and 128-bit key sizes. Figure 33 shows SP network for PRESENT cipher.

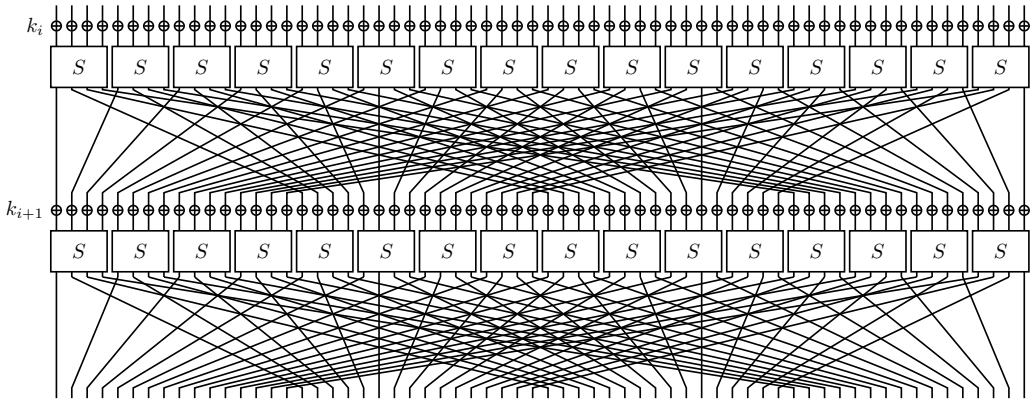


Figure 33: SP network for PRESENT cipher [Vik07]

7.1 Cipher Design

With 31 rounds, the Present-80 is an example of an SP network. A xor of the round key is followed by a 4-bit non-linear substitution layer and linear bit-wise permutation in each round. During each round, the 4-bit SBox is applied 16 times in parallel to the 64-bit input. The higher level pseudo-code for implementing the encryption algorithm is shown

in Figure 35a. A top-level description of the encryption algorithm can be found in Figure 35b.

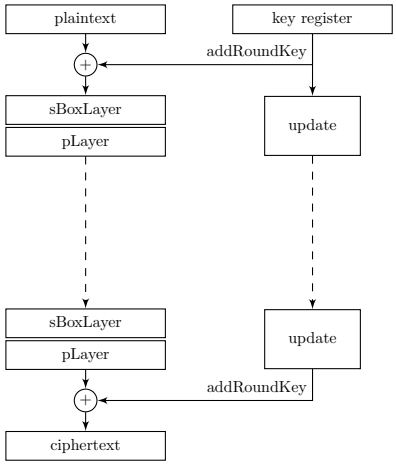
generateRoundKeys()
for $i = 1$ **to** 31 **do**

addRoundKey(STATE,K_i)

sBoxLayer(STATE)
pLayer(STATE)

addRoundKey(STATE,K_{32})

(a)



(b)

Figure 34: A top-level algorithmic description of present

7.1.1 Add Round Key

Provided the round key $K_i = k_{63}, k_{62} \dots k_0$ for $1 \leq i \leq 32$ and the current state $S = s_{63}, s_{62} \dots s_0$, addRoundKey performs the following operation

$$\begin{aligned} & S \to S \oplus K_i \\ \implies & s_t \to s_t \oplus k_t \end{aligned}$$

for $0 \leq t \leq 63$.

7.1.2 Substitution Layer

The substitution box is 4-bit to 4-bit mapping. Table 4 shows the mapping of Sbox in Hexadecimal notation. The 4-bit S-Box is independently repeated 16 times to cover the 64-bit block. The substitution box is a mapping $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ where $\mathbb{F}$ is a finite field.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Table 4: Present Sbox

7.1.3 Permutation Layer

The permutation layer is a bit permutation. The permutation function P(i) maps the i^{th} bit of input to P(i) in the output of the permutation layer. The Table 5 is the mapping of P(i) in tabular form.

7.1.4 Key schedule Algorithm

The Present cipher supports 80-bit or 128-bit long key, in this section we discuss the 80-bit key schedule algorithm. Firstly, the initial 80-bit key is stored in a key register K and is

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Table 5: Present pLayer

represented as $K = k_{79}k_{78}..k_0$. At any round i, PRESENT extracts the 64-bits round key $K_i = \mathrm{k}_{63}\mathrm{k}_{62}..\mathrm{k}_{0\mathrm{s}}$ from the current Key (left most 64 bits) register as follows :

$$K_i = \mathrm{k}_{63}\mathrm{k}_{62}..\mathrm{k}_{0\mathrm{s}} = k_{79}k_{78}..k_{16}$$

After round key extraction, the key register K is updated according to the following rules :

1. The contents of the key register K is rotated by 61-bits to the left.

$$[k_{79}k_{78}..k_0] = [k_{18}k_{17}..k_{20}k_{19}]$$

2. The 4-leftmost bits of the key register K is passed through the PRESENT S-box.

$$[k_{79}k_{78}k_{77}k_{76}] = S[k_{79}k_{78}k_{77}k_{76}]$$

3. The 5-bits of key register k, $k_{19}k_{18}k_{17}k_{16}k_{15}$ is exclusive-ored with the least significant bits of the round counter value i.

$$[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus round - counter$$

8 QPRESENT

We now describe the design of the quantum circuit for SIMON as proposed by [Jan+21b] and call it QPRESENT. QPRESENT only allocates qubits for plaintext and key and does not allocate ancilla qubits.

8.1 Add Round Key

Add round key can be implemented simply using 64 CNOT gates from key to plaintext.

8.2 Sbox

Instead of implementing the algebraic normal form of PRESENT Sbox, authors of [Jan+21b] used LIGHTER-R tool [Das+19] for optimized implementation of Sbox with no ancilla qubits and optimal circuit depth. The quantum circuit for Sbox is shown in Figure 36

The qubits at the output are permuted and therefore two SWAP gates are added at the end of the circuit.

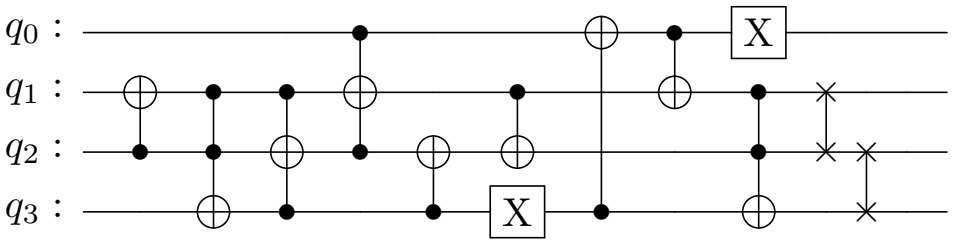


Figure 36: Sbox for QPRESENT

8.3 Permutation Layer

The permutation layer as shown in Table 5 can be implemented using only SWAP gates. Bits 0,21,42 and 63 do not change their positions. Other 60 bits can be grouped in 3-bit groups of 20 bits each. Each 3-bit group is a permutation. For example, 1 map to 16, 16 maps to 4, and then 4 maps to 1. This can be implemented as follows

$$SWAP(x_1, x_4)$$
$$SWAP(x_4, x_{16})$$

The same is followed for the remaining bits. Since we do not count SWAP gates in quantum resources as this operation can be achieved by rewiring, the quantum cost for the permutation layer of QPRESENT is zero.

8.4 Key schedule Algorithm

As described in Section 7.1.4, to implement the key schedule algorithm in the quantum circuit we use algorithm 8.4. The input is an 80-bit key and the output is a 64-bit round key.

KEY EXPANSION FOR QPRESENT($K = k_{79}k_{78}..k_0$)
1 $k = \mathrm{k}_{63}\mathrm{k}_{62}..\mathrm{k}_{0\mathrm{s}} = k_{79}k_{78}..k_{16}$
2 $k = k \gg 19$
3 $[k_{79}k_{78}k_{77}k_{76}] = S[k_{79}k_{78}k_{77}k_{76}]$
4 $[k_{19}k_{18}k_{17}k_{16}k_{15}] = X[k_{19}k_{18}k_{17}k_{16}k_{15}]$
5 **return** k

The operations will be performed on left-most 64 bits denoted by variable k on line 1. Instead of rotating 61 bits to left, rotate 19 bits to right using SWAP gates. The leftmost 4 bits are passed through Sbox and then the round constant is XORed with bits $k_{19}k_{18}k_{17}k_{16}k_{15}$ using NOT gate.

8.5 Grover's Attack

Similar to Grover's Attack on SIMON as described in Section 6.2, the quantum circuit for Grover's Attack can be used. Figure 37 shows the circuit.

We compare the cost of Quantum resources of QPRESENT 64/80 and QSIMON 64/128 shown in Table 6.

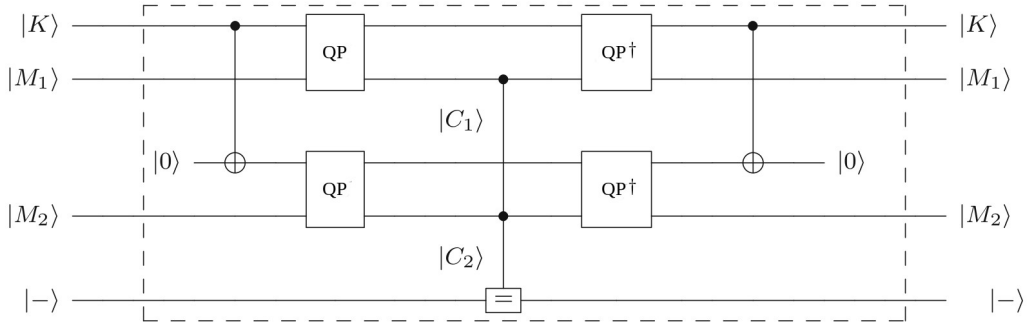


Figure 37: Grover's Attack on QPRESENT for r = 2

Table 6: Comparison of cost for QPRESENT 64/80 [Jan+21b] and QSIMON 64/128[AMM20]

Cipher	Qubits	X	CX	CCX	Ancilla	Depth
QPRESENT 64/80	144	1118	4683	2108	-	311
QSIMON 64/128	192	1216	7396	1408	-	2643

Since the key sizes and number of rounds of both the ciphers are different, the comparison is not fruitful. There is no Sbox in the case of SIMON which can lower its cost complexity as Sbox's quantum circuit is a bottleneck in circuit's depth but also the cost of permutation layer in case of PRESENT is zero due to the use of only SWAP gates.

9 Grover on AES

This section describes the work of [Jaq+20] in brief for AES-128. They have studied quantum circuits under the MAXDEPTH constraint.

Let $N = 2^k$ be the key search space and let $M \geq 1$ is the number of solutions that exists in N. We know that $sin^2(\theta) = \frac{M}{N}$ for $0 < \theta \leq \frac{\pi}{2}$ and for $M \ll N$ we have $\theta \approx \sqrt{\frac{M}{N}}$. The probability of finding one of the M solutions after t iterations is defined as

$$p(t) = sin^2((2t+1)\theta)$$

which after solving for 1 gives $t \approx \frac{\pi}{4\theta} = \frac{\pi}{4}\sqrt{\frac{M}{N}}$.

9.1 Key search

Let $E_K(m) = c$ denote the encryption of message $m \in \{0,1\}^n$ by key $K \in \{0,1\}^k$ to ciphertext c. For r plaintext-ciphertext pairs we have $E_K(m_i) = c_i$. Then the Grover's Oracle is defined as:

$$f(K) = \begin{cases} 1 & E_K(m_i) = c_i \\ 0 & else \end{cases}$$

It is possible that multiple keys other than K lead to the same ciphertext from the given plaintext. The authors call them spurious keys.

The problem is to find the optimal number r such that the probability of finding a spurious key is minimal. Assume K is the actual key we're interested in and K' is spurious. Then the probability that given $K \neq K'$, $E_K(m) = E_{K'}(m)$ is given as:

$$P_{K \neq K'}(E_K(m) = E_{K'}(m)) = \frac{1}{2^n}$$

for r plaintext-ciphertext pairs, we have:

$$p = P_{K \neq K'}((E_K(m_1), E_K(m_2), \ldots, E_K(m_r)) = (E_{K'}(m_1), E_{K'}(m_2), \ldots, E_{K'}(m_r)))) = \left(\frac{1}{2^n}\right)^r = 2^{-nr}$$

Let Y be a random variable that describes the count of spurious keys for given key K and r plaintext-ciphertext pairs. Y is then distributed binomially:

$$P(Y = y) = \binom{2^k - 1}{y} p^y (1-p)^{2^k - 1 - y}$$

We can approximate this to poission distribution with $\lambda = (2^k - 1)p = (2^k - 1)2^{-rn}$

$$P(Y = y) = \frac{e^{-\lambda}\lambda^k}{y!} \approx \frac{e^{-2^{k-rn}} 2^{(k-rn)y}}{y!}$$

The probability that no spurious key exists i.e. $P(Y = 0) \approx e^{-2^{k-rn}}$. We need to maximise this probability which will happen when $rn > k$ or we can choose $r = \lceil \frac{k}{n} \rceil$.

9.2 Parallelization of Grover

Two ways have been described in [KHJ18] for parallelizing Grover's algorithm denoted by inner and outer parallelization. In outer parallelization, multiple instances of the full Grover's algorithm are run on different machines simultaneously for a reduced number of iterations. The outputs can then be checked classically. In the case of inner parallelization, the search space is divided into multiple disjoint subsets and each machine is assigned one subset. Since each machine has a smaller search space, this reduces the number of iterations on each machine.

[Zal99] found that there is a gain of $\sqrt{S}$ in the number of iterations for S parallel machines. This is inefficient as we gain only $\sqrt{S}$ factor in the depth of the quantum circuit whereas the width has become S times the original. [Jaq+20] uses inner parallelization.

9.2.1 Why inner parallelization?

If we use outer parallelization, then the probability that we find the correct key after t iterations is $p_S(t) = 1 - (1 - p(t))^S$. In each machine the number of iterations will be $t_S = \frac{\pi}{4\theta\sqrt{S}}$. Table 7 highlights some of the probabilities for different values of S.

Table 7: Probabilities of finding the correct key for different count of parallel machines

S	$p_S(t)$
1	≈ 1
2	0.96
3	0.94

We can find a general expression by using the series expansion of $sin(x)$ for larger values of S.

$$p_S(t_S) = 1 - \left(1 - \frac{\pi^2}{4S} + O\left(\frac{1}{S^2}\right)\right)^S$$

As S tends to ∞, the above value approaches to $1 - e^{\frac{-\pi^2}{4}} \approx 0.91$. This implies by just by increasing the number of parallel machines S, one cannot get a probability near 1 for

finding the correct key. For the case of inner parallelization, the correct key exists in one of the subsets only, and with t_S iterations, the machine has a near 1 probability of finding it and other machines will not find it. Therefore inner parallelization has a higher chance of obtaining the correct key as compared to outer parallelization.

One thing to note is spurious keys. If spurious keys are present in a subset with the correct key not in that subset, then the spurious key can be discarded classically after the experiment. The probability that spurious keys fall into the subset with the correct key is given by:

$$\sum_{y=1}^{\infty} P(Y = y) = 1 - e^{-\frac{2^{k-rn}}{S}}$$

In the above equation, we have replaced 2^k by $2^k/S$ which is the subset size. In essence, increasing S makes the above probability small.

9.3 Quantum Circuit Design

The authors use AND gate (Figure 38) instead of Toffoli gate(Figure 39). The decomposition of the Toffoli gate is to 7 T gates, 8 Clifford gates with a T-depth(number of T gates in a row[Ves20]) 4 and total depth 11 whereas AND gate used 4 T gates, 11 Clifford gates with T-depth 1 and total depth 8. AND gate uses one ancilla qubit which is released after the operation.

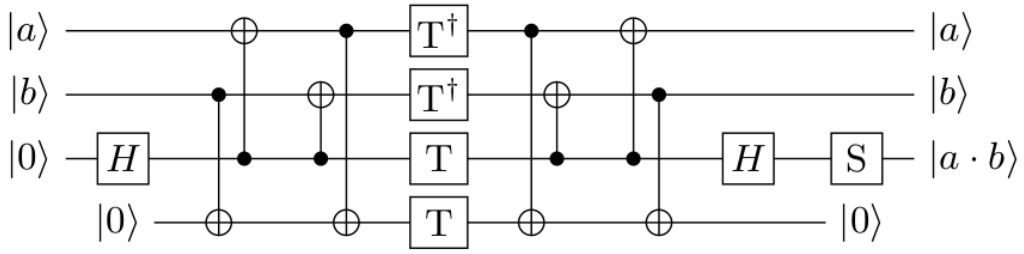


Figure 38: AND Gate [Jaq+20]

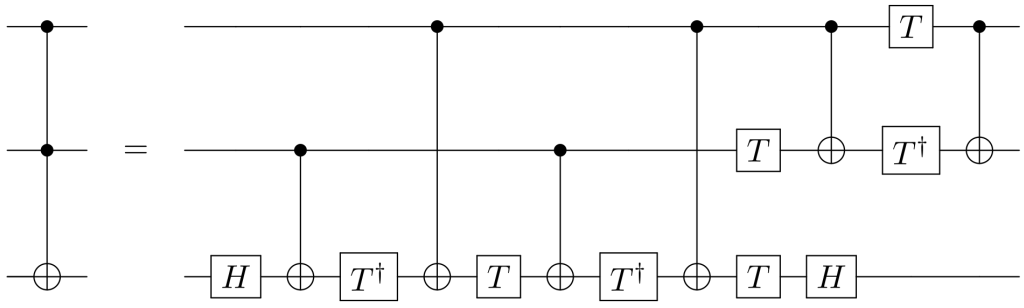


Figure 39: Toffoli gate decomposition[con22b]

9.3.1 Quantum circuit for linear maps

Linear mapping $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ are building blocks of AES. If a linear map is invertible then it can be computed in place using PLU decomposition, where L and U are lower and upper triangular matrices implemented using CNOT gates and P is a permutation matrix for rewiring.

9.3.2 Cost Metrics

Assume a depth limit D_{max}. The authors consider two types of cost: G-cost for the total number of gates and DW-cost which is the product of the depth and width of the circuit. Using these metrics we compute the cost of Grover's attack.

Let $N = 2^k$ be the key search space. Assume that G is the oracle for Grover has G_G gates with G_D depth using G_W qubits. $S = 2^s$ is the count of parallel machines. The probability of finding a a correct key after t iterations is given as $p(t) = sin^2((2t+1)\theta)$. This gives the number of iterations for a probability p i.e. $t_p = \frac{sin^{-1}(\sqrt{p})/\theta - 1}{2} \approx \frac{sin^{-1}(\sqrt{p})}{2}\sqrt{\frac{N}{S}}$. Assume $sin^{-1}(\sqrt{p})/2 = c_p$. Therefore the total depth for t_p iterations of Grover is:

$$D = t_p G_D \approx c_p 2^{\frac{k-s}{2}} G_D \tag{6}$$

The total gate cost over all S machines (G-cost) is:

$$G = t_p G_G S \approx c_p 2^{\frac{k+s}{2}} G_G \tag{7}$$

The total width $W = G_W S$ qubits. Therefore the DW-cost is :

$$DW \approx c_p 2^{\frac{s+k}{2}} G_D G_W \tag{8}$$

From these equations, we can see that reducing S results in a reduction in DW-cost and G-cost. But if the depth is a constraint then the attacker has to parallelize the circuit. For this case, we look into optimization of Grover's Oracle under the depth limit.

G_D is the depth of Grover's oracle. Given depth limit D_{max}. We can run at most $t_{max} = D_{max}/G_D$ iterations of G. For probability p of finding the correct key, we calculate S i.e. $p = sin^2((2t_{max}+1)\sqrt{\frac{S}{N}})$. This gives:

$$S = \frac{(sin^{-1}(\sqrt{p}))^2 N}{(2\frac{D_{max}}{G_D}+1)^2} \approx c_p^2 2^k \frac{G_D^2}{D_{max}^2} \tag{9}$$

With this value we can recompute total gate cost (G-cost) from equation 7

$$G = c_p^2 2^k \frac{G_D G_G}{D_{max}} \tag{10}$$

and DW-cost

$$DW = c_p^2 2^k \frac{G_D^2 G_W}{D_{max}} \tag{11}$$

10 Quantum AES

We now describe the design of the quantum circuit for AES-128 as proposed by [Jaq+20] and call it QAES.

10.1 Sbox

The authors compared various previously proposed Sbox designs on the G-cost and DW-cost metrics by reconstructing them. [BP11] Sbox was effective in terms of G-cost and DW-cost hence they chose it. Table 8 shows the comparison of costs of various Sboxes.

Table 8: Comparison of cost of Sboxes

Sbox	CNOT	Clifford	T	M	T-depth	full depth	width	DW
[Gra+15]	8683	1028	3584	0	217	1692	44	74,448
[BP10]	818	264	164	41	35	497	41	20,377
[BP11]	654	184	136	34	6	101	137	13,837

10.2 Add round key

This operation can be implemented simply using 128 CNOT gates from the key to the state.

10.3 ShiftRows

Since this operation is based on the permutation of bytes, it can be implemented using SWAP gates only and requires zero cost.

10.4 Mix Column

The authors compared two variants of the mix column. One is the in-place version which does not require the use of ancilla qubits which saves the width and the other is not in-place and requires ancilla qubits but saves the depth of the circuit. Table 9 shows comparison of these two variants.

Table 9: Comparison of cost of mix column variants

MC	CNOT	Clifford	T	M	T-depth	full depth	width	DW
In place	1108	0	0	0	0	111	128	14,208
[Max19]	1248	0	0	0	0	22	318	6,996

Since the authors are working in a depth restricted environment, the authors chose [Max19] mix column variant due to its low DW cost. The DW cost is mainly affected by the G_D^2 term and therefore it is crucial to minimize the depth of the oracle used, here its mix column.

10.5 Key Expansion

The authors generate keys on the fly which do not require ancilla qubits. The circuit for key expansion is shown in Figure 40

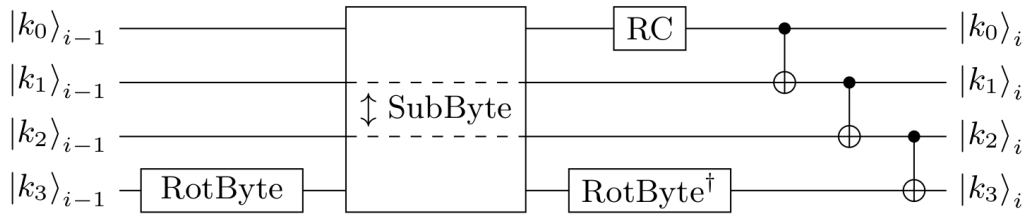


Figure 40: AES-128 key expansion [Jaq+20]

$|k_j\rangle_i$ represent the j^{th} word (4 bytes) of the i^{th} round key. First $|k_3\rangle_{i-1}$ (last word) is rotated and then passed through AES Sbox. The output of this operation is collected

at the first word i.e. the output is XORed with $|k_0\rangle_{i-1}$. The result is then added with a round constant for that round. The next steps involve XORing with other words of $(i-1)^{th}$ round key to obtaining words of i^{th} round key which is performed using CNOT gates. The dotted lines in the Sbox represent that the corresponding wires have not been used in Sbox operation.

10.6 QAES-128 circuit

Figure 41 shows the QAES circuit. Each wire represents 4 words (128 qubits). $|k\rangle$ represents the master key and m represents the message. BS is Sbox, SR is shift rows and MC is mix column. Here we have used an in-place version of mix columns.

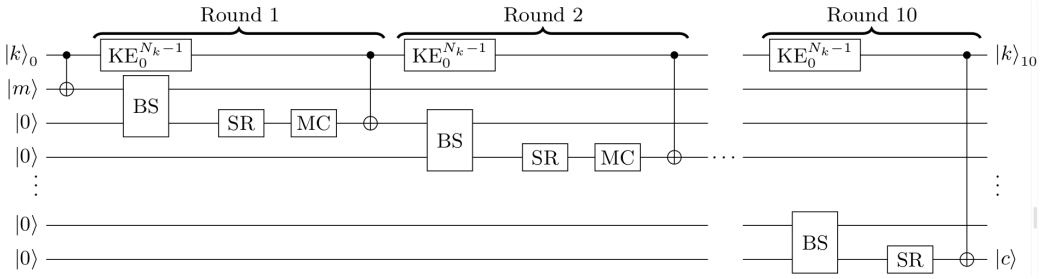


Figure 41: QAES-128 [Jaq+20]

Table 10 shows the cost of QAES-128 for both variants of mix columns. The in-place version of the mix column has lower DW-cost and G-cost compared to [Max19] version.

Table 10: QAES-128 cost of both variants of mix columns

MC	CNOT	Clifford	T	M	T-depth	full depth	width	DW
QAES-128 In place	2,91,150	83,116	54,400	13,600	120	2,827	1,785	50,46,195
QAES-128 [Max19]	2,93,730	83,236	54,400	13,600	120	2,094	2,937	61,50,078

10.7 Grover's Attack

Figure 42 shows the Grover Oracle for 1 iteration. FwAES implies the forward AES which outputs the ciphertexts without cleaning the qubits. FwAES† circuit is the reversible operation of FwAES which cleans the qubits. The attack requires $r = 2$ plaintext-ciphertext pair for finding the unique key. The oracles cost is shown in Table 11

Table 11: QAES-128 Grover's Oracle cost of both variants of mix columns (r=2)

MC	CNOT	Clifford	T	M	T-depth	full depth	width	DW
QAES-128 In place	5,85,051	1,69,184	1,09,820	27,455	121	2,815	3,329	93,71,135
QAES-128 [Max19]	5,89,643	1,68,288	1,09,820	27,455	121	2,096	5,633	1,18,06,768

10.7.1 Cost Estimates

With the help of Table 10 and the equations 6,7, and 8 we can calculate the cost estimates for Grover's Attack without depth restriction. Using equations 9, 10, and 11 we can calculate the cost estimates for Grover's Attack with depth restriction.

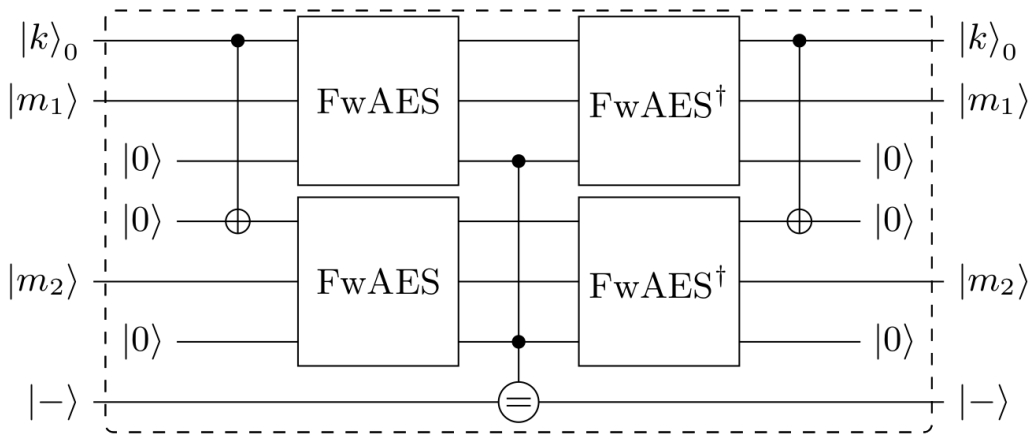


Figure 42: Grover's Attack on QAES-128 [Jaq+20]

Let's calculate the cost estimates of Grover's Attack on QAES-128 in place with r = 2 with and without depth constraint. The total gate cost G_G is the sum of all the gates shown in Table 10 i.e.

$$G_G = 5,85,051 + 1,69,184 + 1,09,820 + 27,455 = 8,91,510 \approx 1.7 \times 2^{19}$$
$$G_D = 2,815 \approx 1.37 \times 2^{11}$$
$$G_W = 3,329 \approx 1.62 \times 2^{11}$$

Therefore the cost estimates without depth constraints keeping $S = 1$ is:

$$D \approx 1.37 \times 2^{11} \times 2^{64} = 1.37 \times 2^{75}$$
$$G \approx 1.7 \times 2^{19} \times 2^{64} = 1.7 \times 2^{83}$$
$$DW \approx 1.37 \times 2^{11} \times 1.62 \times 2^{11} \times 2^{64} \approx 1.1 \times 2^{87}$$

Similarly, the cost estimates with depth constraint of 2^{40} are:

$$S \approx 2^{128} \times 1.37^2 \times 2^{22} \times 2^{-80} \approx 1.87 \times 2^{70}$$
$$G \approx 2^{128} \times 1.37 \times 2^{11} \times 1.7 \times 2^{19} \times 2^{-40} \approx 1.16 \times 2^{119}$$
$$DW \approx 2^{128} \times 1.37^2 \times 2^{22} \times 1.62 \times 2^{11} \times 2^{-40} \approx 1.52 \times 2^{122}$$

NIST[16] has proposed a maximum of 2^{170}/MAXDEPTH quantum gates for AES-128 but this does not take into account the effects of parallelization. For MAXDEPTH = 2^{40}, [16] has bounded the count of quantum gates by $2^{170}/2^{40} = 2^{130}$. From the above calculation of G-cost we can see that the number of gates required by AES-128 is much less after parallelization (2^{119}).

11 Conclusion

Applications of Grover's search algorithm in Quantum cryptanalysis are studied in this paper. We briefly studied hardware and software-friendly ciphers. The Quantum circuit for SAES was modeled and then the quantum resources were optimized. As a result, the quantum architecture is not a barrier to a quantum adversary carrying out any potential quantum attack. We designed a Quantum circuit for SIMON and PRESENT ciphers which are hardware friendly with a minimal number of qubits. We then discussed the

parallelization of Grover's algorithm and its cost estimates which are used in modeling the quantum circuit and estimating the quantum resources for AES-128. AES-128 was studied with and without depth constraints under the rules proposed by NIST [16].

We implemented SAES and QSAES in python using Qiskit[ANI+21] and libquantum[Wei] to verify the results. We were unable to run a successful Grover's Attack as it was time-consuming. It seems that current frameworks are not yet optimized for running a full Grover's Attack to find a unique key using minimum resources and have a scope of optimization. libquantum written in C has issues when running codes across different machines. The code and obtained results are open-sourced [Dah22].

References

[BHT97] Gilles Brassard, Peter Hoyer, and Alain Tapp. "Quantum Algorithm for the Collision Problem". *ACM SIGACT News (Cryptology Column)* 28 (June 1997). DOI: 10.1145/261342.261346.

[Zal99] Christof Zalka. "Grover's quantum searching algorithm is optimal". *Phys. Rev. A* 60 (4 Oct. 1999), pp. 2746–2751. DOI: 10.1103/PhysRevA.60.2746. URL: https://link.aps.org/doi/10.1103/PhysRevA.60.2746.

[Sch+00] Bruce Schneier et al. *The Twofish Team's Final Comments on AES Selection*. 2000. URL: https://web.archive.org/web/20100102041117/http://schneier.com/paper-twofish-final.pdf.

[BHT06] Gilles Brassard, Peter Hoyer, and Alain Tapp. "Quantum cryptanalysis of hash and claw-free functions". Vol. 28. June 2006, pp. 163–169. DOI: 10.1007/BFb0054319.

[Bog+07] A. Bogdanov et al. "PRESENT: An Ultra-Lightweight Block Cipher". *Cryptographic Hardware and Embedded Systems - CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 450–466. ISBN: 978-3-540-74735-2.

[Vik07] A. BogdanovL. R. KnudsenG. LeanderC. PaarA. PoschmannM. J. B. RobshawY. SeurinC. Vikkelsoe. "PRESENT: An Ultra-Lightweight Block Cipher" (2007). URL: https://link.springer.com/chapter/10.1007/978-3-540-74735-2_31.

[Ber08] Daniel J. Bernstein. "ChaCha, a variant of Salsa20". 2008. URL: https://cr.yp.to/chacha.html.

[Che+08] Donny Cheung et al. "On the Design and Optimization of a Quantum Polynomial-Time Attack on Elliptic Curve Cryptography". *Theory of Quantum Computation, Communication, and Cryptography*. Ed. by Yasuhito Kawano and Michele Mosca. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 96–104. ISBN: 978-3-540-89304-2.

[BP10] Joan Boyar and René Peralta. "A New Combinational Logic Minimization Technique with Applications to Cryptology". May 2010, pp. 178–189. ISBN: 978-3-642-13192-9. DOI: 10.1007/978-3-642-13193-6_16.

[BP11] Joan Boyar and René Peralta. "A depth-16 circuit for the AES S-box." Vol. 2011. Jan. 2011, p. 332. ISBN: 978-3-642-30435-4. DOI: 10.1007/978-3-642-30436-1_24.

[Bea+13] Ray Beaulieu et al. *The SIMON and SPECK Families of Lightweight Block Ciphers*. Cryptology ePrint Archive, Report 2013/404. https://ia.cr/2013/404. 2013.

[Gra+15] Markus Grassl et al. "Applying Grover's algorithm to AES: quantum resource estimates" (Dec. 2015).

[16] *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. 2016. URL: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[Ber17] Daniel J. Bernstein (2017). URL: https://blog.cr.yp.to/20171017-collisions.html.

[Alm+18] Mishal Almazrooie et al. "Quantum Grover Attack on the Simplified-AES". *Proceedings of the 2018 7th International Conference on Software and Computer Applications*. ICSCA 2018. Kuantan, Malaysia: Association for Computing Machinery, 2018, pp. 204–211. ISBN: 9781450354141. DOI: 10.1145/3185089.3185122. URL: https://doi.org/10.1145/3185089.3185122.

[KHJ18] Panjin Kim, Daewan Han, and Kyung Jeong. "Time–space complexity of quantum search algorithms in symmetric cryptanalysis: applying to AES and SHA-2". *Quantum Information Processing* 17 (Oct. 2018). DOI: 10.1007/s11128-018-2107-3.

[Das+19] Vishnu Asutosh Dasu et al. "LIGHTER-R: Optimized Reversible Circuit Implementation For SBoxes". *2019 32nd IEEE International System-on-Chip Conference (SOCC)*. 2019, pp. 260–265. DOI: 10.1109/SOCC46988.2019.1570548320.

[Max19] Alexander Maximov. "AES MixColumn with 92 XOR gates". *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 833.

[AMM20] Ravi Anand, Arpita Maitra, and Sourav Mukhopadhyay. "Grover on SIMON". Apr. 2020.

[Jaq+20] Samuel Jaques et al. "Implementing Grover Oracles for Quantum Key Search on AES and LowMC". May 2020, pp. 280–310. ISBN: 978-3-030-45723-5. DOI: 10.1007/978-3-030-45724-2_10.

[Ves20] Martin Vesely. *What meaning is Depth and T-Depth in Quantum Circuit?* 2020. URL: https://physics.stackexchange.com/questions/597781/what-meaning-is-depth-and-t-depth-in-quantum-circuit.

[ANI+21] MD SAJID ANIS et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. DOI: 10.5281/zenodo.2573505.

[Jan+21a] Kyung-Bae Jang et al. "Grover on Simplified AES". *2021 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. 2021, pp. 1–4. DOI: 10.1109/ICCE-Asia53811.2021.9642017.

[Jan+21b] Kyungbae Jang et al. "Efficient Implementation of PRESENT and GIFT on Quantum Computers". *Applied Sciences* 11.11 (2021). ISSN: 2076-3417. DOI: 10.3390/app11114776. URL: https://www.mdpi.com/2076-3417/11/11/4776.

[Qua21] IBM Quantum. *IBMQ Simulators*. 2021. URL: https://quantum-computing.ibm.com/.

[con22a] Wikipedia contributors. *Simon (cipher)*. Wikipedia, 2022. URL: https://en.wikipedia.org/wiki/Simon_(cipher).

[con22b] Wikipedia contributors. *Toffoli Gate*. Wikipedia, 2022. URL: https://en.wikipedia.org/wiki/Toffoli_gate.

[Dah22] Gopal Ramesh Dahale. *QSKC-Grover*. https://github.com/Gopal-Dahale/QSKC-Grover. 2022.

[Gor22] Steven Gordon. *Cryptography Study Notes (Chapter 9)*. 2022. URL: https://sandilands.info/crypto/.

[Qis22a] Qiskit. "MCMT" (2022). URL: https://qiskit.org/documentation/stubs/qiskit.circuit.library.MCMT.html.

[Qis22b] Qiskit. "Transpiler (qiskit.transpiler)" (2022). URL: https://qiskit.org/documentation/apidoc/transpiler.html#:~:text=Overview%5C%C2%5C%B6,present%5C%20day%5C%20noisy%5C%20quantum%5C%20systems..

[Wei] Hendrik Weimer. URL: http://www.libquantum.de/.