Synthesizing Quantum Circuits of AES with Lower T-depth and Less Qubits

Zhenyu Huang[1,2] and Siwei Sun[3,4]

[1] SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3] School of Cryptology, University of Chinese Academy of Sciences, Beijing, China
[4] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China

Abstract. The significant progress in the development of quantum computers has made the study of cryptanalysis based on quantum computing an active topic. To accurately estimate the resources required to carry out quantum attacks, the involved quantum algorithms have to be synthesized into quantum circuits with basic quantum gates. In this work, we present several generic synthesis and optimization techniques for circuits implementing the quantum oracles of iterative symmetric-key ciphers that are commonly employed in quantum attacks based on Grover and Simon's algorithms. Firstly, a general structure for implementing the round functions of block ciphers in-place is proposed. Then, we present some novel techniques for synthesizing efficient quantum circuits of linear and non-linear cryptographic building blocks. We apply these techniques to AES and systematically investigate the strategies for depth-width trade-offs. Along the way, we derive a quantum circuit for the AES S-box with *provably* minimal T-depth based on some new observations on its classical circuit. As a result, the T-depth and width (number of qubits) required for implementing the quantum circuits of AES are significantly reduced. Compared with the circuit proposed in EUROCRYPT 2020, the T-depth is reduced from 60 to 40 without increasing the width or 30 with a slight increase in width. These circuits are fully implemented in Microsoft Q# and the source code is publicly available. Compared with the circuit proposed in ASIACRYPT 2020, the width of one of our circuits is reduced from 512 to 371, and the Toffoli-depth is reduced from 2016 to 1558 at the same time. Actually, we can reduce the width to 270 at the cost of increased depth. Moreover, a full spectrum of depth-width trade-offs is provided, setting new records for the synthesis and optimization of quantum circuits of AES.

Keywords: Quantum Circuit, T-depth, Grover's Algorithm, AES

1 Introduction

The rapid and fruitful development of the theory and practice on building computing machines that exploit quantum mechanical phenomena has made the research on algorithms running on quantum computers a topic with potential practical consequences. This especially attracts substantial attention from the cryptographic community, where the security of many primitives relies on the computational hardness of solving certain number theoretical or combinatorial problems. Shor's algorithm [Sho99] is probably the most influential research in this aspect. It will compromise the security of many widely deployed public-key cryptosystems (including RSA, DSA, and ECC) if large-scale quantum computers are ever built.

For symmetric-key ciphers, a trivial application of Grover's algorithm [Gro96] results in a quadratic speedup of the exhaustive search attack. If the attackers have access to the keyed quantum oracle, it is shown that many symmetric-key schemes can be broken with Simon's period-finding algorithm [KLLN16a, BLNS21]. Since the practical relevance of querying online keyed quantum oracles is questionable, some subsequent work investigates techniques limited to classical queries and offline quantum computations [HS18a, BHN$^+$19, CNS17]. Also, quantum attacks derived from dedicated cryptanalytic techniques are extensively studied [BNS19b, BNS19a, HS18b, KLLN16b, NS, HS]. To concretely estimate the complexities in the standard quantum circuit model [NC16], the quantum circuits for these attacks have to be constructed based on some basic quantum gates. Our community is especially interested in constructing efficient quantum circuits for cryptographic primitives fulfilling specific input-output behaviors since such circuits typically work as sub-circuits of the quantum attacks. The National Institute of Standards and Technology (NIST) used the complexity of the quantum circuit for AES with a bound of depth called `MAXDEPTH` as a baseline to categorize the post-quantum public-key schemes into different security levels in the call for proposals to the standardization of post-quantum cryptography [NIS16]. Note that when the quantum circuits are applied in exhaustive key search, we can use parallelization by dividing the search space, which naturally decreases the depth but increases the number of quantum gates and qubits, and in fact, to perform exhaustive key search attacks on AES on a quantum computer with Grover's algorithm under NIST's `MAXDEPTH` bound, parallelization is required for the majority of values of `MAXDEPTH`. For parallelization, we have the following observation.

Observation 1 *Let $\mathfrak{C}$ be the quantum circuit implementing the Grover oracle and $\mathbb{V}$ be the search space. If we divide $\mathbb{V}$ into k^2 equal parts and execute k^2 parallel Grover searches, the number of iterations of $\mathfrak{C}$ required in each search will decrease by a factor of k. The number of qubits required in each search will not change, while the number of gates used in each search will decrease by a factor of k. Therefore, the number of qubits required in all searches will increase by a factor of k^2, and the number of gates used in all searches will increase by a factor of k.*

Let $\mathfrak{C}'$ be a new quantum circuit whose depth is reduced to $depth(\mathfrak{C})/k$, and the numbers of qubits and gates are respectively increased by a factor less than k^2 and a factor less than k. Then, Observation 1 implies that using $\mathfrak{C}'$ in a parallel approach is better than applying $\mathfrak{C}$. This is the very reason that [JNRV20] claims: Grover's algorithm does not parallelize well, meaning that minimizing depth rather than width is crucial to make the most out of the available depth. Finally, NIST states that the `MAXDEPTH` restriction is motivated by the difficulty of running extremely long serial computations. Plausible values for `MAXDEPTH` range from 2^{40} logical gates (the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year) through 2^{64} logical gates (the approximate number of gates that current classical computing architectures can perform serially in a decade), to no more than 2^{96} logical gates (the approximate number of gates that atomic scale qubits with speed of light propagation times could perform in a millennium).

Related Work. To perform quantum attacks based on Grover's and Simon's algorithm, one has to implement the actual device that executes the attack, whose cost is evaluated in the quantum circuit model. From an attacker's perspective, it is important to reduce the cost. From a designer's perspective, it is important to have an accurate understanding of the cost to evaluate the security margin and to guide future designs. In particular, the classical and quantum implementations of AES receive most attention from our community.

The first quantum circuit of AES [GLRS16] was proposed by Grassl et al., where the so-called zig-zag structure was introduced to reduce the width (the number of qubits required) of the resulting quantum circuits. The width was further reduced in a follow-up work [ASAM18]. In [LPS19], Langenberg et al. presented improved quantum circuits for the S-box and key expansion of AES, leading to significantly improved AES circuits. At ASIACRYPT 2020, by tweaking the zig-zag structure, together with new quantum circuits for the AES S-box and its inverse constructed based on improved classical circuits Zou et al. significantly improved the width of the quantum circuit of AES [ZWS+20].

While the primary goal of the above works is to reduce the width, the cryptographic community is more concerned with the depth, since in NIST's ongoing post-quantum standardization effort, different security categories are defined according to the quantum resources needed to attack AES with a depth bound. At EUROCRYPT 2020, Jaques et al. proposed several techniques to improve the depth, and presented the currently known sallowest quantum circuit for AES [JNRV20]. Besides, we also see works considering the implementation of quantum circuits for other primitives (e.g., SHA-2 and SHA3 [AMG+16], LowMC [JNRV20], and ECC [BBvHL21]).

Note that this line of research is not only interested by the cryptographic community, but also contributes to a much broader subject known as synthesis and optimization of quantum circuits. As realistic quantum computers will likely require some fault tolerance schemes where the amount of error correction is proportional to the resources used, the effect of quantum circuit optimizations becomes even more profound than its classical counterpart. In fact, the industrial community has already invested huge resources to develop the tool chain for synthesis and optimization of quantum circuits [Mic, IBM].

Our Contributions. We propose an *in-place* quantum circuit for the (invertible) round-function of a block cipher or other iterative designs. With this type of in-place structure, the circuits implementing the round functions can be connected together to form the whole design without using additional ancilla qubits. In addition, we present a generic method for constructing such *in-place* circuits with *out-of-place* sub-circuits. Then, a systematic comparison of this structure with previous designs (the pipeline structure and the zig-zag structure) is made with respect to the depth and width (the number of qubits) requirements.

We then consider how to implement the building blocks of a symmetric-key cipher efficiently. Specifically, a SAT-based technique for synthesizing small linear components is presented, which can output the CNOT network with the minimal gate count. For nonlinear components, a systematic method for constructing a circuit mapping $|x\rangle|a\rangle$ to $|x\rangle|a \oplus f(x)\rangle$ based on a circuit mapping $|x\rangle|0\rangle$ to $|x\rangle|f(x)\rangle$ meeting certain clearly defined conditions with only additional CNOT gates is given. Based on this method, we present circuits for AES S-box and it inverse with both T-depth and width improved compared to the one given in [ZWS+20].

To further reduce the T-depth, we formulate a technique for converting an AND-depth-t classical circuit into a T-depth-t quantum circuit. We note that this is not a trivial conversion due to the peculiarities of a quantum circuit, and the natural order of the classical circuit has to be rearranged to achieve this goal. Based on this method with a new observation on the classical circuit for the AES S-box, we obtain two circuits for the AES S-box with T-depth-4 and T-depth-3, respectively, both of which have lower T-depth than the one presented at EUROCRYPT 2020 [JNRV20]. Since the degree of the algebraic normal form of the AES S-box is 7, with less than 3 stages of multiplications, one cannot generate polynomials with degree 7, which implies that our implementation reaches the theoretical lower bound.

By applying the method presented in this paper, we significantly improve the efficiency of the quantum circuits for AES. Compared with the circuit proposed in EUROCRYPT 2020, the T-depth is reduced from 60 to 40 without increasing the width and the T-gate count, or 30 with a slight increase in width and T-gate count. These circuits are fully implemented in Microsoft Q# and the source code is publicly available. Compared with the circuit proposed in ASIACRYPT 2020, the width of one of our circuits is reduced from 512 to 371, the number of Toffoli gates is reduced from 19788 to 17888, and the Toffoli-depth is reduced from 2016 to 1558 at the same time. Actually, we can reduce the width to 270 at the cost of increased depth. Moreover, by varying the local and global circuit structures, a full spectrum of depth-width trade-offs are provided and illustrated in Figure 19 (Section 8), setting new records for the synthesis and optimization of quantum circuits of AES.

2 Preliminaries on Synthesizing Quantum Circuits

The states of an n-qubit quantum system can be described by the unit vectors in $\mathbb{C}^{2^n}$. A quantum state is typically written as $|u\rangle$, and in this paper we denote it by $|u\rangle_n$ to emphasize that this state has n qubits. When n is clear from the context, we abbreviate $|0\cdots 0\rangle_n$ as $|0\rangle$.

A quantum algorithm manipulates the state of an n-qubit system through a series of unitary transformations and measurements, where a unitary transformation is a linear map U over $\mathbb{C}^{2^n}$ with $UU^\dagger = U^\dagger U = I$. Any unitary transformation can be constructed with a finite set of single-qubit and two-qubit unitary transformations through composition and tensor product. In the standard quantum circuit model [NC16], we call these simple single-qubit and two-qubit unitary transformations quantum gates. In particular, we consider how to synthesize a quantum circuit with the commonly used universal fault-tolerant gate set Clifford + T, which contains the Clifford gates:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \quad S = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}, \quad \texttt{CNOT} = \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix},$$

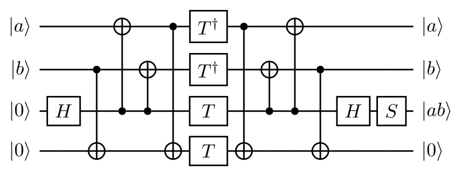
and the non-Clifford gate $T = \begin{pmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{pmatrix}$.

We also frequently employ the Pauli-X gate $X = HS^2H = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$ and the Toffoli gate

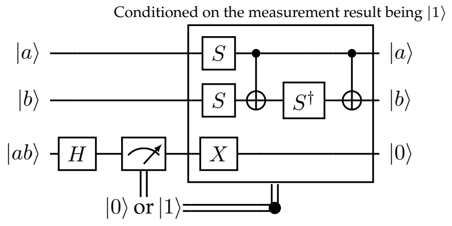
$$\text{ToF} = \begin{pmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0\end{pmatrix}.$$

In this work, we are mainly concerned with the quantum circuits that can compute a classical vectorial Boolean function when the input is in the computational basis. Since the Toffoli gate can be used to simulate a universal gate set for classical computation, all these circuits can be constructed by using only Toffoli gates with additional qubits (potentially set to appropriate values). For example, the multiplication operation $a \cdot b$ can be directly implemented by the Toffoli gate $|a\rangle|b\rangle|c\rangle \to |a\rangle|b\rangle|c \oplus a \cdot b\rangle$. There is another quantum circuit for implementing the functionality of a classical AND gate, and we call it a quantum AND gate. This gate together with its adjoint is illustrated in Figure 1.

Optimization Goals. The complexity of a quantum circuit can be measured in terms of its width (number of qubits), gate count, and depth. The cryptographic



(a) Quantum AND gate



(b) Quantum AND† gate

Fig. 1. The quantum AND gate together with its adjoint

community is mainly concerned with the depth metric, and in particular the T-depth is the most interested parameter of a quantum circuit. The reason can be summarized as follows.

Firstly, as indicated in [JNRV20] and our introduction, favoring lower depth at the cost of a slightly larger width in the circuit leads to costs that are smaller in several metrics than for the circuits presented in [GLRS16, ASAM18, LPS19]. Secondly, the time for fault tolerant quantum computation is proportional to one round of measurement per layer of T-gates, and so the runtime is dominated by T-depth, rather than gate count, circuit depth, or even measurement depth [Fow12]. This is why in the adjoint of the quantum AND gate (see Figure 1) we avoid using T-gates at the cost of quantum measurement.

The T-depth is defined as the minimum number of stages of parallel applications of T-gates in a circuit, where parallel T-gates are allowed when they are acting on different qubits. Note that we can implement the Toffoli gate with several different circuits based on the Clifford $+T$ gates with different T-depth, and these circuits with T-depth 1, 2, and 3 can be found in [Sel12, AMMR13].

3 The Round-In-Place Structure for Iterative Primitives

We start by reviewing the two main structures used in previous work on the quantum circuits for AES, including the pipeline structure [JNRV20] and the zig-zag structure [GLRS16] illustrated in Figure 2 and Figure 3, respectively. In the Figures we can see that each pair of neighbouring sub-circuits (represented as small rectangles) are not perfectly aligned horizontally, but forming a stepladder pattern. This interconnection pattern is due to the *out-of-place* nature of the circuit $\mathcal{R}_i$, which implements the i-th round function of AES, mapping $|k_i\rangle|x\rangle|0\rangle$ to $|k_i\rangle|x\rangle|O(\mathcal{R}_s)\rangle$, where $|k_i\rangle$ is the round key, $|x\rangle$ is the input state, and $|O(\mathcal{R}_i)\rangle$ is the output state of the round function. Here by *out-of-place* we mean that the output $|O(\mathcal{R}_i)\rangle$ of the round function is not carried in the qubits that encode the input $|x\rangle$.

It is easy to see that, since the round transformation is implemented by the out-of-place circuit $\mathcal{R}_s$, the pipeline structure needs lots of qubits to preserve the input states of all rounds, and the zig-zag structure is designed to reduce

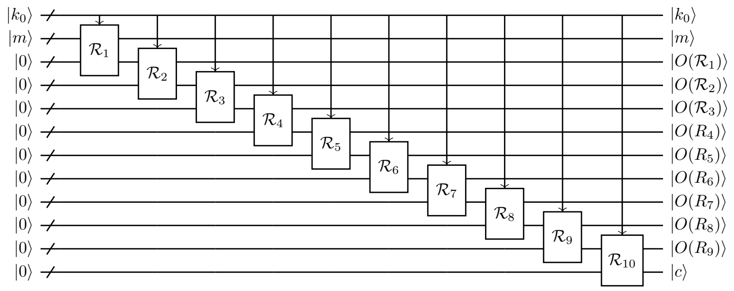


Fig. 2. The pipeline structure, where k_0 is the initial key, m is the plaintext, c is the ciphertext. For the sake of simplicity, the ancilla qubits and the key expansion process are omitted, and $|k_0\rangle$ is used as the round key in each round.

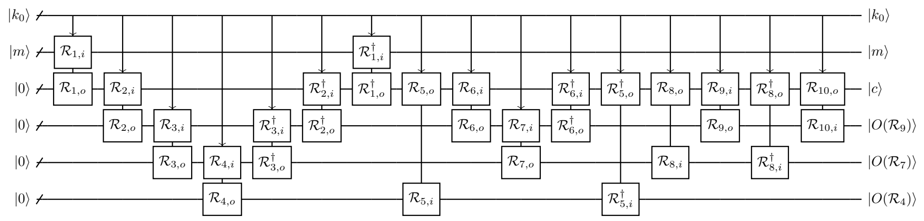


Fig. 3. The zig-zag structure, where $\mathcal{R}^\dagger_s$ is used to erase the redundant input states of $\mathcal{R}_{s+1}$.

the cost of qubits by using the reverse circuit for the last round to erase these inputs. In [JNRV20], the pipeline structure was used for designing low-depth circuits of AES, and in [GLRS16, ASAM18, LPS19], the zig-zag structure was used for designing low-width circuits of AES.

3.1 The Round-In-Place Structure

For iterative designs with invertible round functions, an *in-place* implementation (with some ancillae) of the round function maps $|k_s\rangle|x\rangle|0\rangle$ to $|k_s\rangle|O(\mathcal{R}_s)\rangle|0\rangle$. Such in-place implementations of the round functions can be connected naturally to form the compositions of the round functions without the need of additional qubits. However, directly designing an in-place circuit with low T-depth for the round transformation involved in typical ciphers is difficult. In contrast, a compact out-of-place circuit for a round transformation can be efficiently derived from a compact classical circuit by implementing additions by CNOT gates and multiplications by Toffoli gates.

A natural idea is to construct an in-place circuit based on out-of-place subcircuits, and this construction is depicted in Figure 4. This structure is a general form of the circuits used in [AMG+16], and later we will show that previous work neglect important things in implementing $U_{R^{-1}}$.

We write the classical round transformation of a symmetric cipher into the following form $\texttt{Round} : (x, k) \to (T(x, k), k)$. Let $\texttt{Round}^{-1} : (z, k) \to (T'(z, k), k)$ be the inverse of the round transformation, where $T'(T(x, k), k) = x$. Suppose we have an out-of-place circuit U_R for $\texttt{Round}$ and an out-of-place circuit $U_{R^{-1}}$ for $\texttt{Round}^{-1}$. Actually, by implementing the classical Boolean operations with Toffoli gates we can always construct such out-of-place circuits. Then the circuit in Figure 4 in-place implements $\texttt{Round}$ with some ancilla qubits.

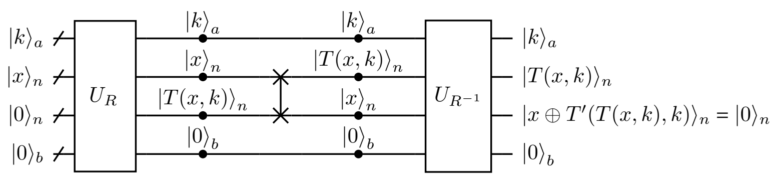


Fig. 4. In-place implementation of an invertible round transformation based on out-of-place sub-circuits

Figure 4 provides an efficient way to implement an invertible round transformation in-place. We call this kind of circuit an *out-of-place based* (abbreviated as OP-based) in-place circuit. Based on this circuit, we can implement an iterative cipher with the structure shown in Figure 5. We call this structure the OP-based round-in-place structure, abbreviated as the *round-in-place* structure.

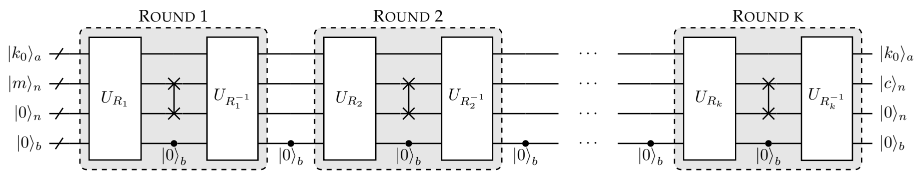


Fig. 5. The OP-based round-in-place structure

Remark 1. In Figure 4, the functionality of $U_{R^{-1}}$ is to compute T', and then XOR T' into the third register. Moreover, this functionality should work when the state on the third register is $|x\rangle$, which is related with the state in the second register. Therefore,

- If we use a $U_{R^{-1}}$ which only works for $|0\rangle$, the output will be wrong.
- If we use a $U_{R^{-1}}$ which works for any $|y\rangle$, the output will be correct. However, since the relationship between $|x\rangle$ and $|T(x,k)\rangle$ is not sufficiently used, such $U_{R^{-1}}$ will cost more quantum resources.

In [AMG+16], an in-place circuit of the χ function of SHA3 was presented with a structure which is similar with our OP-based in-place circuit. However, their implementation of χ^{-1} is a straightforward implementation of the classical circuit from Keccak tools, which is equivalent to a $U_{R^{-1}}$ that only works for $|0\rangle$, leading to incorrect output. In [ZWS+20], the implementation is equivalent to a $U_{R^{-1}}$ that works for any $|y\rangle$, which needs more quantum resources.

Remark 2. Note that $U_{R^{-1}}$ and $U_R^\dagger$ are different. $U_R^\dagger$ is the circuit for the adjoint of the unitary transformation U_R. It can be obtained by placing the adjoints of the gates in U_R in the reverse order, and thus $U_R^\dagger$ and U_R cost the same quantum resources in most times. In comparison, $U_{R^{-1}}$ is the circuit for the reverse transformation R^{-1}. Hence, the costs of U_R and $U_{R^{-1}}$ are different in general. However, in our context, R is for encryption, while R^{-1} is for decryption. For most symmetric ciphers, the complexities of the encryption and decryption are similar. Hence the quantum resources for implementing U_R and $U_{R^{-1}}$ are almost the same.

3.2 A Depth-Width Comparison of Different Structures

Given an iterative block cipher whose block size is n-bit, a rough estimation of the width of the circuits with the pipeline, zig-zag, and round-in-place structures, when they use the same out-of-place circuit U_R as their main component, can be easily obtained from Figure 2 to Figure 5. We suppose U_R requires n qubits for the input data block, n qubits for the output data block, and αn for the round key and ancillae. Note that in Figure 2 and Figure 3, we omit the possible ancilla qubits used in $\mathcal{R}_s$. Then, the widths of the three structures for implementing all r-round operations are presented in Table 1.

Table 1. The widths of different structures, where t is the minimal number such that $\sum_{i=1}^{t} i > r$.

Pipeline	Zig-zag	Round-in-place
$(r+\alpha+1)n$	$(t+1+\alpha)n \approx (\sqrt{2r}+\alpha)n$	$(2+\alpha)n$

To estimate the depths of these structures, we need to consider two different scenarios. In the first scenario, we build circuits for the Grover oracles used in exhaustive key search attacks. In the second scenario, we construct circuits for the encryption oracles used in [KLLN16a].

Circuits for Grover Oracles. First, we consider the Grover oracle: $|y\rangle|q\rangle \rightarrow |y\rangle|q \oplus f(y)\rangle$, where $f(y)$ is a Boolean function that outputs one bit 1 or 0. When given some pairs of plaintext and ciphertext, by constructing a Grover oracle with the key $|k\rangle$ as the input, one can use Grover's algorithm to search the correct key. For simplicity, we consider the case of using one pair of plaintext and ciphertext (m, c_0). In this case, the circuits of the Grover oracle based on different structures are shown in Figure 6. In this figure, the out-of-place sub-circuit E_{OP} denotes the encryption circuit generated by the pipeline or zig-zag structure, while the in-place sub-circuit E_{IP} denotes the encryption circuit generated by the round-in-place structure. Besides the ciphertext $|c\rangle$, E_{OP} outputs the redundant state $|r\rangle$ corresponding to those $O(R_s)$ in Figure 2. Since the plaintext m is fixed, $|m\rangle$ is a computational basis state, which can be viewed as ancilla qubits in this circuit. The sub-circuit "COM" compares $|c\rangle$ with the provided ciphertext c_0, if they are equal, then flips the target qubit $|q\rangle$. Apparently, in these two circuits, the depth of the oracle is roughly two times of the depth of the encryption circuit (E_{OP} or E_{IP}).

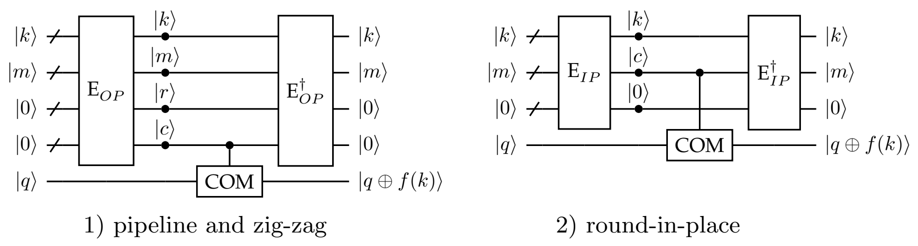
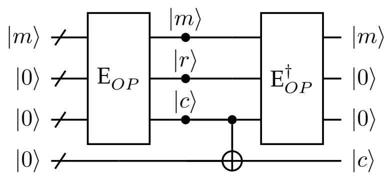


Fig. 6. The Grover oracle based on different structure

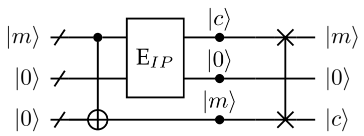
Circuits for Encryption Oracles. Now we consider the encryption oracle defined in [KLLN16a]: $|m\rangle|0\rangle \to |m\rangle|E(m)\rangle$, where m is the plaintext, and $E(m)$ is the corresponding ciphertext. For this oracle, if its input is a superposition $\sum_m |m\rangle|0\rangle$, then its output will be a superposition $\sum_m |m\rangle|E(m)\rangle$. Figure 7 shows how to construct quantum cryptographic oracles based on different structures. Here $|c\rangle = |E(m)\rangle$ is the ciphertext. Note that, in this oracle, we do not need to store $|k\rangle$ by qubits, since we can pre-compute all the round keys via classical computation, and add them on the input of each round by Pauli-X gates. For the pipeline and the zig-zag structures, since we need uncomputation to erase the redundant state $|r\rangle$, the depth of the oracle is twice of that of the encryption process. However, for the round-in-place structure, since we do not generate $|r\rangle$, we do not need the uncomputation process.

By summarizing the above discussion, we have the following results. Given a symmetric cipher with r rounds, suppose the depths (or T-depths) of U_R and $U_{R^{-1}}$ in Figure 4 are both d. This is reasonable according to Remark 2. If we ignore the cost of the compare process in the Grover oracle and the copy process in the quantum cryptographic oracle, then the depths (or T-depths) and the DW-costs (the product of depth and width) of the oracles based on these three structures are as shown in Table 2.

From the results in Table 1 and Table 2, we have the following observations.



1) pipeline and zig-zag



2) round-in-place

Fig. 7. The encryption oracle based on different structure

Table 2. The depths and DW-costs of the oracles based on different structures

Metric	Type	Pipeline	Zig-zag	Round-in-place
Depth	Grover	$2r \cdot d$	$2 \cdot \sum_{i=1}^{t}(2(t-i)+1) \approx 4r \cdot d$	$4r \cdot d$
	Encrypt	$2r \cdot d$	$\approx 4r \cdot d$	$2r \cdot d$
DW-cost	Grover	$2r(r+1+\alpha)nd$	$2r(\sqrt{2r}+\alpha)nd$	$4r(2+\alpha)nd$
	Encrypt	$2r(r+1+\alpha)nd$	$2r(\sqrt{2r}+\alpha)nd$	$2r(2+\alpha)nd$

Observation 2 *The round-in-place structure has the smallest width in any cases. For the Grover oracle, the pipeline structure has the lowest depth. When $r \leq 3+\alpha$, the DW-cost of the pipeline structure is lowest, and when $r > 3+\alpha$, the DW-cost of the round-in-place structure is lowest. For the quantum cryptographic oracle, the pipeline structure and the round-in-place structure have the same depth, and the DW-cost of the round-in-place structure is always the lowest.*

4 Synthesizing Optimal CNOT Circuits with SAT

It is well known that an invertible linear transformation over $\mathbb{F}_2^n$ can be implemented in-place with n qubits by the CNOT gates [PMH08]. Given an invertible transformation represented as a binary matrix, the PLU decomposition technique [GLRS16, GLRS16, JNRV20, ZWS+20] and the heuristic algorithm proposed at FSE 2020 [XZL+20] are typically employed to produce a compact CNOT circuit implementing the linear transformation. However, these methods are far from being optimal. In the following, we present a SAT-based method to generate the most compact CNOT circuit for invertible linear transformations over $\mathbb{F}_2^n$. Due to the difficulty of solving large scale SAT models in practice, the SAT-based technique only works when n is small.

The idea of our algorithm is to convert the problem of finding a circuit with k CNOT gates into the problem of solving a system of Boolean equations. Similar ideas were used in [FS10, Sto16, MSM18], where different classical and quantum circuit synthesis problems were considered. A brief introduction of our method is given in the following, and a detailed description can be found in Supplementary Material A.

Given a positive integer k and a linear transformation which can be expressed as n linear forms $L_i(x_1, x_2, \dots, x_n)$ in x_i, we generate a model with the following sets of variables: $B = (b_{ij})_{k\times n}$, $C = (c_{ij})_{k\times n}$, $F = (f_{ij})_{n\times n}$, and $\Psi = \{\psi_{i,j,s}\}_{k\times n\times n}$. These variables are of different semantics. For B and C, $b_{ij_1} = c_{i,j_2} = 1$ means the i-th gate is a CNOT gate with control wire j_1 and target wire j_2. For F, $f_{ij} = 1$ implies that the final output of the j-th wire is L_i. For Ψ, $\psi_{i,j,k} = 1$ indicates that after the i-th gate, the coefficient of x_k in the Boolean expression for the j-th wire is 1. We generate the equations of these variables to encode the gates and their outputs. It can be shown that the obtained system of equations has a solution (which can be tested with SAT solvers) if and only if there is a CNOT-circuit with k CNOT gates. By incrementally increasing k, we can identify the minimal k such that the corresponding equations can be satisfied simultaneously. According to our experiments, linear transformations with less than 9 variables can be solved in a reasonable time. Hence, we employ this method to identify the optimal CNOT sub-circuits in our implementations of the AES S-box presented in Section 5.

5 In-Place Circuits for Nonlinear Components

It is well known that for any $\mathbb{F}_2^n \to \mathbb{F}_2^n$ permutation, there is an in-place quantum circuit implementing it using only Pauli-X, CNOT, and Toffoli gates with at most one ancilla qubit [SPMH03]. To obtain this in-place implementation with minimal width, one needs to solve a corresponding permutation factorization problem, which is computationally difficult for large permutations like the AES S-box. Moreover, the complexity in terms of the gate count and depth of the circuits produced by this method is typically far from being satisfactory. For example, in [GLRS16], the authors estimated that the in-place circuit of the AES S-box with 9 qubits would cost about 9695 T gates. In contrast, with the method provided in this section, we can construct an in-place circuit that requires only 22 qubits and 728 T gates.

In what follows, we consider special quantum circuits (named as $\mathfrak{C}^0$- and $\mathfrak{C}^*$-circuits) implementing a vectorial Boolean function, based on which in-place quantum circuits for nonlinear transformations with different shapes can be constructed.

5.1 $\mathfrak{C}^0$- and $\mathfrak{C}^*$-Circuits for a Vectorial Boolean Function

Given a vectorial Boolean function $f\colon \mathbb{F}_2^a \to \mathbb{F}_2^b$, a $\mathfrak{C}^0$-circuit for f is a quantum circuit mapping $|x\rangle_a|0\rangle_b|0\rangle_c$ to $|x\rangle_a|f(x)\rangle_b|0\rangle_c$ for any $x \in \mathbb{F}_2^a$, and a $\mathfrak{C}^*$-circuit for f is a quantum circuit mapping $|x\rangle_a|y\rangle_b|0\rangle_c$ to $|x\rangle_a|y \oplus f(x)\rangle_b|0\rangle_c$ for any $(x, y) \in \mathbb{F}_2^{a+b}$. Obviously, a $\mathfrak{C}^*$-circuit for f is always a $\mathfrak{C}^0$-circuit for f. Moreover, building a $\mathfrak{C}^0$-circuit is much easier than building a $\mathfrak{C}^*$-circuit since a $\mathfrak{C}^*$-circuit has more restrictions on its input-output behavior. For example, the circuits for the AES S-box proposed in [GLRS16, ASAM18, LPS19] are $\mathfrak{C}^0$-circuits, but not $\mathfrak{C}^*$-circuits.

Next, we present a generic method that can convert a $\mathfrak{C}^0$-circuit for f with some clearly defined properties (called *simplex* $\mathfrak{C}^0$-circuits) into a $\mathfrak{C}^*$-circuit for f efficiently. In particular, the obtained $\mathfrak{C}^*$-circuit do not increase the T-depth of the corresponding $\mathfrak{C}^0$-circuit.

Simplex $\mathfrak{C}^0$-Circuits. A $\mathfrak{C}^0$-circuit for f is *simplex* if it maps $|x\rangle_a|y\rangle_b|0\rangle_c$ to

$$|x\rangle_a|A(y) \oplus f(x)\rangle_b|0\rangle_c,$$

where $A : \mathbb{F}_2^b \to \mathbb{F}_2^b$ is an invertible linear function. We now consider the gate-level structures of (simplex) $\mathfrak{C}^0$-circuits, which gives some intuitive ideas on how to construct efficient simplex $\mathfrak{C}^0$-circuits and the sufficient condition for a $\mathfrak{C}^0$-circuit to be simplex.

Suppose we have a quantum circuit built with Pauli-X, CNOT and Toffoli gates. Let $|x_1, x_2, \cdots, x_a\rangle|y_1, y_2, \cdots, y_b\rangle|0\rangle_c$ and $|t_1, \cdots, t_a\rangle|z_1, \cdots, z_b\rangle|0\rangle_c$ be the input and output of the circuit with x_i, y_i and z_i in $\mathbb{F}_2$. For $j \in \{1, 2, \cdots, b\}$, we have

$$z_j(x, y) = \sum_{u,v} a^j_{u,v} x^u y^v + \sum_u b^j_u x^u + \sum_u d^j_u y^u + T^j(x) + L^j(y) + c^j,$$

where $a^j_{u,v}$, b^j_u, d^j_u, and $c^j \in \mathbb{F}_2$, $u \in \mathbb{F}_2^a$, $v \in \mathbb{F}_2^b$, T^j and L^j are linear functions, and x^u is the monomial $\prod_{u_i=1} x_i$. If this circuit is a $\mathfrak{C}^0$-circuit of the vectorial function $f(x) = (f_1(x), \cdots, f_b(x))$, then

$$z_j(x, 0) = \sum_u b^j_u x^u + T^j(x) + c^j = f_j(x),$$

which implies that

$$z_j(x, y) = f_j(x) + \sum_{u,v} a^j_{u,v} x^u y^v + \sum_s d^j_u y^u + L^j(y).$$

Consequently, if the quantum gates applied to the input qubits

$$|x_1, x_2, \cdots, x_a\rangle|y_1, y_2, \cdots, y_b\rangle|0\rangle_c$$

do not produce any nontrivial $x^u y^v$ and y^u terms whose degree are greater or equal to 2, the $\mathfrak{C}^0$-circuit must be simplex.

Now we analyze which operations may generate these $x^u y^v$ and y^u terms. We denote the set of the b output wires in this circuit as $\mathcal{W}$, and the set of other $a + c$ wires as $\mathcal{V}$. Then, the algebraic expressions of the initial states on the wires in $\mathcal{W}$ are y. All operations that operate on at least one wire in $\mathcal{W}$ can be classified into the following 7 types of operations illustrated in Figure 8.

Denote $\sum_i a^j_{u,v} x^u y^v + \sum_s d^j_u y^u + L^j(y)$ by $h_j(x, y)$, then $z_j(x, y) = h_j(x, y) + f_j(x)$. From the above analysis, to design a $\mathfrak{C}^0$-circuit, since there is no constraint on $h_j(x, y)$, we do not need to care about the generation of $x^u y^v$ and y^u. This means the above 7 types of operations are all permitted. The designer only needs

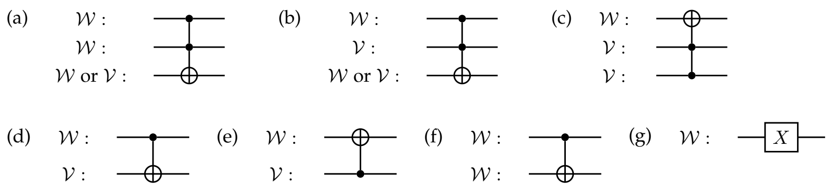


Fig. 8. Operations that operate on at least one wire in $\mathcal{W}$

to focus on efficiently constructing $f(x)$. From the view of algebraic expression, the qubits on the wires in $\mathcal{W}$ can be seen as newly defined variables. Additions and multiplications about these variables can be used to generate $f(x)$.

To design a simplex $\mathfrak{C}^0$-circuit, we should guarantee that $h_j(x, y) = L^j(y)$, which means that $x^u y^v$ and y^u should not appear. We have the following observation:

Observation 3 *If some $x^u y^v$ was generated, it is hard to eliminate this $x^u y^v$ in the following steps, unless we repeat the same Toffoli gate which generates it.*

This means generating $x^u y^v$ will likely increase the cost of the circuits. Hence a natural criterion for designing a compact $\mathfrak{C}^*$-circuit is to avoid generating $x^u y^v$.

Under this criterion, operations (a) and (b) should obviously be avoided. For operation (d), it can only be applied when we use the qubit on the target wire as a dirty qubit for some CNOT gates. Note that operation (e), (f), and (d) under this constrain can be described together as the following operation.

(h): Apply s CNOT gates, $s \geq 1$, which map $|u\rangle_a |w\rangle_b |v\rangle_c$ to $|u\rangle_a |L(u, v, w)\rangle_b |v\rangle_c$ for any u, v, w, where L is a linear function w.r.t. u, v, w.

Thus, our criterion for designing a compact $\mathfrak{C}^*$-circuit is: *only operations (c), (h), (g) can be applied on the output wires.* Note that without applying operation (a), y^{γ_s} will not be generated either. Therefore, under this criterion, if we construct U_f, which is a $\mathfrak{C}^0$-circuit of f, then the output of U_f is

$$|x\rangle |h_1(x, y) + f_1(x), \ldots, h_b(x, y) + f_b(x)\rangle |0\rangle,$$

where $h_k(x, y) = L_k(y)$ is a linear function with respect to y. Let $A(y) = (L_1(y), L_2(y), \ldots, L_b(y))$, then the output can be denoted by $|x\rangle |A(y) \oplus f(x)\rangle |0\rangle$, which implies this is a simplex $\mathfrak{C}^0$-circuit.

Converting Simplex $\mathfrak{C}^0$-circuits into $\mathfrak{C}^*$-circuits. Let U_f be a $\mathfrak{C}^0$-circuit of $f(x)$. We can construct a $\mathfrak{C}^*$-circuit of $f(x)$ as shown in Figure 9. Here $U_{A^{-1}}$ is an in-place sub-circuit, which implements $A^{-1}(y)$. $|y\rangle$ will be converted to $\left|A^{-1}(y)\right\rangle$ after passing $U_{A^{-1}}$, and the output of this modified circuit will be $|x\rangle \left|A(A^{-1}(y)) \oplus f(x)\right\rangle |0\rangle = |x\rangle |y \oplus f(x)\rangle |0\rangle$, which means this is a $\mathfrak{C}^*$-circuit of $f(x)$.

It is easy to see that the $\mathfrak{C}^*$-circuit constructed by the above method has the same width as the $\mathfrak{C}^0$-circuit U_f. Moreover, the numbers and the depths of Toffoli

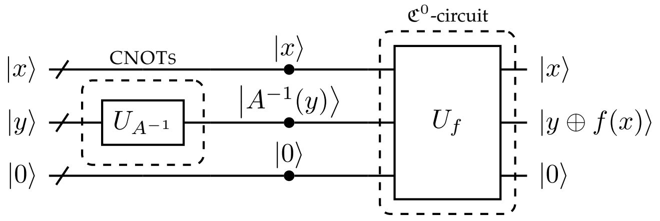


Fig. 9. A $\mathfrak{C}^*$-circuit based on a simplex $\mathfrak{C}^0$-circuit

gates (or T gates) are the same for these two circuits. From this construction, we can see that the $\mathfrak{C}^*$-circuit and the simplex $\mathfrak{C}^0$-circuit are almost the same. Hence, to efficiently construct a $\mathfrak{C}^*$-circuit, we should also follow the above criterion of designing a simplex $\mathfrak{C}^0$-circuit, and by this way we will always obtain a simplex $\mathfrak{C}^0$-circuit. Then, the process of designing a $\mathfrak{C}^*$-circuit of $f(x)$ can be summarized as following steps:

1) We design U_f, a $\mathfrak{C}^0$-circuit of $f(x)$, in which only operations (c), (g), and (h) can be applied on the wires in $\mathcal{W}$. Then U_f will be a simplex $\mathfrak{C}^0$-circuit.
2) We determine A. Note that, in a simplex $\mathfrak{C}^0$-circuit, operations (c), (g) will not generate any term containing y, which means A is determined by (h) operations. Hence, we can obtain A by computing the composition of the linear transformations corresponding to all (h) operations.
3) If A is identity, this is already a $\mathfrak{C}^*$-circuit . Otherwise, we implement A^{-1} by an in-place CNOT circuit $U_{A^{-1}}$, then construct a $\mathfrak{C}^*$-circuit as Figure 9.

For most S-box implementation problems, the number of the output wires is not bigger than 8, which means, by our SAT-based algorithm, we can make sure $U_{A^{-1}}$ uses the minimal number of CNOT gates.

5.2 In-Place Implementations of Nonlinear Transformations of Different Shapes with $\mathfrak{C}^0$- and $\mathfrak{C}^*$-Circuits

We show how to implement nonlinear transformations with typical shapes encountered in practice with $\mathfrak{C}^0$- and $\mathfrak{C}^*$-Circuits. In most symmetric ciphers, a nonlinear component can correspond to one of the classical invertible nonlinear transformations presented in Figure 10.

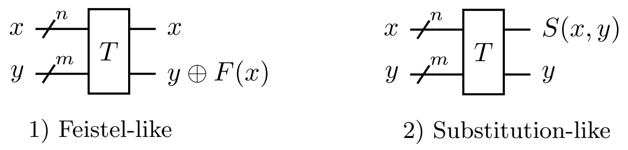


Fig. 10. Two kinds of classical invertible nonlinear transformations

Feistel-like Transformations. First, we consider Feistel-like classical invertible nonlinear transformations of the form

$$\Psi : (x, y) \mapsto (x, y \oplus F(x)).$$

The quantum circuit of this type of nonlinear transformation can be realized by a $\mathfrak{C}^*$-circuit of F, which in turn can be derived from a simplex $\mathfrak{C}^0$-circuit of F. We note that a $\mathfrak{C}^*$-circuit of F, mapping $|x\rangle|y\rangle|0\rangle$ to

$$|x\rangle|y \oplus F(x)\rangle|0\rangle = |\Psi(x, y)\rangle|0\rangle,$$

is an *out-of-place* implementation of F but an *in-place* implementation of Ψ. Feistel-like structures are frequently seen in Feistel ciphers, NFSR-based designs, and key-schedule algorithms of block ciphers.

For example, `SubByte` and the following XOR operation in the AES key schedule can be seen as a Feistel-like transformation. Therefore to in-place implement this transformation, we can construct a simplex $\mathfrak{C}^0$-circuit of the AES S-box, then extend it to a $\mathfrak{C}^*$-circuit. In the previous works about AES, a lot of proposed quantum circuits of the AES S-box are simplex $\mathfrak{C}^0$-circuits [ASAM18, LPS19, ZWS+20], since in these circuits only operations (c),(h),(g) are applied. Hence, by the method proposed in Section 5.1, we can easily extend them to the $\mathfrak{C}^*$-circuits. For example, based on the simplex $\mathfrak{C}^0$-circuit of the AES S-box proposed in [ZWS+20], we construct a compact $\mathfrak{C}^*$-circuit[5]. In this $\mathfrak{C}^*$-circuit, the in-place circuit implementing A^{-1} with minimum number of CNOT gates is achieved by our SAT-based algorithm. This circuit costs 10 CNOT gates, and in Supplementary Material E, we present the matrix corresponding to A^{-1} and the specific form of this circuit.

In Table 3, we compare the quantum resources used in our $\mathfrak{C}^*$-circuit and those used in the $\mathfrak{C}^*$-circuit proposed in [ZWS+20].

Table 3. Quantum resources for implementing the S-box of AES

	#ancilla	Toffoli-depth	#Toffoli	#CNOT	#Pauli-X	source
$\mathfrak{C}^0$-S-box	6	41	52	326	4	[ZWS+20]
$\mathfrak{C}^*$-S-box	7	60	68	352	4	[ZWS+20]
	6	41	52	336	4	This paper

Substitution-like Transformations. Next, we consider classical invertible substitution-like transformations of the form

$$\Phi : (x, y) \to (S(x, y), y).$$

[5] The C code for checking the correctness of this $\mathfrak{C}^*$-circuit is available at https://github.com/AES-quantum-circuit/AES-quantum-circuit

It is easy to see that the description of such nonlinear transformation is the same as that of the round transformation discussed in Section 3, and thus we can implement such nonlinear transformation by the OP-based in-place circuit in Figure 11.

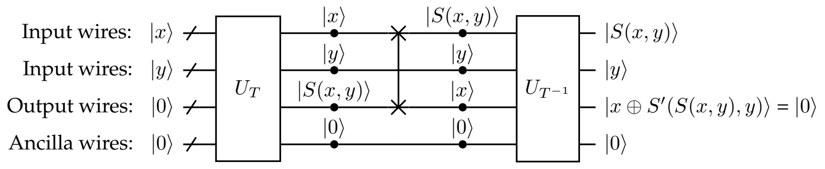


Fig. 11. An OP-based in-place circuit for a substitution-like nonlinear transformation, where S' is a function satisfying $S'(S(x,y),y) = x$ for any y.

We now consider how to implement the building blocks of the circuit depicted in Figure 11. For U_T, it can be implemented as a $\mathfrak{C}^0$-circuit of S. For $U_{T^{-1}}$, a circuit which mapping $|S(x)\rangle|y\rangle|x\rangle|0\rangle$ to $|S(x)\rangle|y\rangle|x \oplus S'(S(x),y)\rangle|0\rangle$, one may attempt to implement this part as a $\mathfrak{C}^*$-circuit of S'. We now show that this is an overshoot.

For $U_{T^{-1}}$, if we set $z = S(x)$, then $S'(z,y) = x$, and $U_{T^{-1}}$ maps $|z\rangle|y\rangle|S'(z,y)\rangle|0\rangle$ to $|z\rangle|y\rangle|0\rangle|0\rangle$. Suppose U_{T_0} is a circuit that maps $|z\rangle|y\rangle|0\rangle|0\rangle$ to $|z\rangle|y\rangle|S'(z,y)\rangle|0\rangle$, then obviously, $U_{T_0}^{\dagger}$, the reverse circuit of U_{T_0}, is equivalent to $U_{T^{-1}}$. Therefore, to implement a substitution-like transformation, we only need to design a $\mathfrak{C}^0$-circuit of S, and a $\mathfrak{C}^0$-circuit of S', whose reverse circuit is used.

6 A Method for Constructing Low T-Depth Circuits

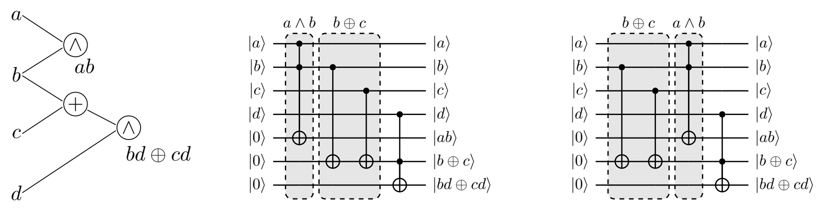
As discussed in Section 2, the T-depth is the most concerned parameter. In our context, T gates only appear in the Toffoli gates, the quantum AND gates and their adjoints, which are employed to implement the quantum correspondences of the multiplications in the classical computation. In the following, we first show that there always exists a quantum circuit with T-depth equal to the AND-depth of the corresponding classical circuit. Therefore, we can first construct a classical circuit with low AND-depth, and then convert it into a low T-depth quantum circuit.

6.1 Classical AND-depths v.s. Quantum T-depths

The AND-depth of a classical circuit (a.k.a. the multiplicative depth) constructed with AND, XOR, and NOT gates is the largest number of AND gates on any path from a primary input to a primary output. For example, the AND-depth of the classical circuit shown in Figure 12 is 1.

The readers may think that it is trivial to build a quantum version of a given classical circuit such that the T-depth of the quantum circuit is equal to the

AND-depth of the classical circuit by just properly replacing the classical AND gates with Toffoli gates or quantum AND gates, all of which have T-depth 1 implementations. However, for quantum circuits, a direct copy as the "b" signal in Figure 12 is not allowed and a qubit cannot be used in different quantum gates simultaneously. Therefore, a quantum circuit obtained from a classical one by the simple replacement strategy mentioned above maintaining the natural order of operations may result in increased T-depth. For example, the quantum circuits with different Toffoli-depth depicted in (2) and (3) of Figure 12 both implement the functionality of the classical circuit given in (1) of Figure 12. Next, we show that the AND-depth of a classical circuit set a lower bound for the T-depth of its quantum counterpart, and this lower bound is achievable.



(1) classical circuit (2) T-depth-2 (3) T-depth-1

Fig. 12. Quantum implementations of a classical circuit with multiplicative depth 1

Theorem 1. *Given a classical circuit with AND-depth s, the T-depth of the quantum circuit implementing all the nodes of the classical circuit is not smaller than s. Moreover, with sufficiently many ancillae, we can construct a quantum circuit implementing all the nodes of the classical circuit with T-depth s.*

A constructive proof of Theorem 1 can be found in Supplementary Material B, which also provides a generic method to convert an AND-depth t classical circuit into a T-depth t quantum circuit. We illustrate this method with the classical circuit given by Example 1.

Example 1. $M_4 = M_1 \cdot M_2$, $M_5 = M_2 \cdot M_3$, $M_6 = M_4 \oplus M_3$, $M_7 = M_5 \oplus M_1$, $M_8 = M_7 \cdot M_2$, $M_9 = M_7 \cdot M_3$, $M_{10} = M_8 \oplus M_6$, $M_{11} = M_{10} \oplus M_9$, $M_{12} = M_7 \cdot M_6$, $M_{13} = M_{11} \cdot M_3$

The AND-depth of the circuit given by Example 1 is 3. Before we present the method for building the corresponding T-depth-3 quantum circuit, we define the AND-depth for each intermediate node (or signal) appearing in the circuit. We call a variable an AND-variable if it represents the output of an AND gate. In Example 1, M_4, M_5, M_8, M_9, M_{12}, and M_{13} are AND-variables.

Let M_i and M_j be two AND-variables. M_j is said to be an AND-successor of M_i if $M_j = M_i \cdot M_k$ for some k, or $M_j = M_u \cdot M_v$ for some u and v such that M_u is generated from M_i by some XOR operations, which is denoted by $M_i \to M_j$. Also, we call M_i is an AND-predecessor of M_j. In our Example 1, we have $M_5 \to M_8$ and $M_8 \to M_{13}$, forming a directed path $M_5 \to M_8 \to M_{13}$. By generating all such paths, a directed acyclic graph is obtained with the nodes representing the AND-variables. Then, the AND-depth of an AND-variable M, denoted as $d_\wedge(M)$ is defined as k, if M is the k-th node in the longest path containing M. Apparently, the AND-depth of a classical circuit is equal to the maximum AND-depth of its AND-variables. It is easy to see that, for an AND-variable M, if M has no AND-predecessor, then $d_\wedge(M) = 1$, otherwise $d_\wedge(M) = 1 + \max_{v \in \mathrm{Pre}(M)} d_\wedge(v)$, where $\mathrm{Pre}(M)$ denotes the set of all predecessors of M. In Example 1, $d_\wedge(M_4) = d_\wedge(M_5) = 1, d_\wedge(M_8) = d_\wedge(M_9) = d_\wedge(M_{12}) = 2$, and $d_\wedge(M_{13}) = 3$.

Now, we are ready to describe our method for building the quantum circuit. Note that since we aim at reducing the T-depth, in our constructions, we always use the quantum AND gate with T-depth 1 and its adjoint with T-depth 0 depicted in Figure 1 whenever possible, while in the figures illustrating the quantum circuits, we use Toffoli gates due to the compactness of its visualization. The circuit generated by our method has the following features, Firstly, for AND-variables with the same AND-depth, a layer of quantum AND gates which generate these AND-variables are applied in parallel. Secondly, before the quantum AND layer, all necessary input are generated with a CNOT network. In particular, when a variable is used as inputs of different AND gates of the subsequent AND layer, we can copy it into an ancilla qubit with the application of a CNOT gate, and clean the effect of the CNOT gate after the quantum AND layer.

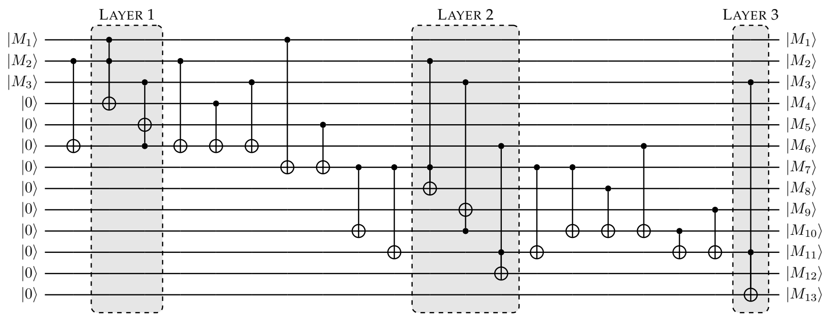


Fig. 13. An AND-depth-3 implementation for the classical circuit in Example 1

Figure 13 present a quantum circuit corresponding to Example 1. In this circuit, we have three layers of quantum AND gates, within each layer the gates are applied in parallel. In Layer 1, we generate M_4 and M_5. In Layer 2, we generate M_8, M_9, and M_{12}. In Layer 3, we generate M_{13}. The variables required by Layer 1 are M_1, M_2, and M_3. Since $M_4 = M_1 \cdot M_2$, and $M_5 = M_2 \cdot M_3$, M_2 is needed in two different AND gates. Therefore, before Layer 1, we copy $|M_2\rangle$ to another qubit by a CNOT gate. This is an idle qubit, which will be used to store $|M_6\rangle$. We clean this qubit after Layer 1. The variables required by Layer 2 are M_2, M_3, M_6, and M_7. Hence, we have to generate M_6 and M_7. We do this by applying 4 CNOT gates before Layer 2. Similarly, M_7 is required for computing M_8, M_9, and M_{12}. We copy it into two idle qubits by 2 CNOT gates before Layer 2, which are cleaned after Layer 2. The variables required by Layer 3 are M_2, and M_{11}. Thus, before Layer 3, we apply 4 CNOT gates to generate M_{11}. This leads to a quantum circuit computing all the nodes in Example 1 with T-depth 3.

6.2 A Trick for Reducing the AND-depth of Classical Circuits

According to the discussion of the previous section, low AND-depth classical circuits imply low T-depth quantum circuits. In this section, we show how to reduce the AND-depth of a classical circuit without changing the functionalities of its primary outputs based on a simple observation. Let $M_4 = M_1 \cdot M_2$ and $M = M_4 \cdot M_3$ with $d_\wedge(M_1) = 2$, $d_\wedge(M_2) = 1$, and $d_\wedge(M_3) = 1$. Then $d_\wedge(M_4) = 3$, and $d_\wedge(M) = 4$. In addition, We can deduce that $M = (M_1 \cdot M_2) \cdot M_3$. Obviously, we also have $M = M_1 \cdot (M_2 \cdot M_3)$. Therefore, if we first compute $M_4 = M_2 \cdot M_3$, and then $M = M_1 \cdot M_4$. The AND-depth of M is reduced from 4 to 3.

We now show how this idea works for a more complicated case based on Example 1, where M_1, M_2, M_3 are primary inputs and M_{12}, M_{13} are primary outputs. For M_{13}, we have

$$M_{13} = (M_{10} \oplus M_9)M_3 = (M_8 \oplus M_6 \oplus M_9)M_3 = M_7(M_2M_3) \oplus M_6M_3 \oplus M_7M_3.$$

We modify the circuit by using the following steps to generate M_{13}: $N_1 = M_2 \cdot M_3$, $N_2 = M_6 \cdot M_3$, $N_3 = M_7 \cdot M_3$, $N_4 = M_7 \cdot N_1$, $N_5 = N_4 \oplus N_2$, $M_{13} = N_5 \oplus N_3$. M_{13} is not an AND-variable anymore. It is easy to check that $d_\wedge(N_1) = 1$, $d_\wedge(N_2) = 2$, $d_\wedge(N_3) = 2$, and $d_\wedge(N_4) = 2$. Therefore, the AND-depth of this new circuit is 2. The modified circuit is given by Example 2, where M_{12} and M_{13} are the primary outputs.

Example 2. $M_4 = M_1 \cdot M_2$, $M_5 = M_2 \cdot M_3$, $M_6 = M_4 \oplus M_3$, $M_7 = M_5 \oplus M_1$, $M_8' = M_6 \cdot M_3$, $M_9' = M_7 \cdot M_3$, $M_{10}' = M_7 \cdot M_5$, $M_{11}' = M_{10}' \oplus M_8'$, $M_{12} = M_7 \cdot M_6$, $M_{13} = M_{11}' \oplus M_9'$.

More generally, given a classical circuit, we can try to reduce its AND-depth as follows. Firstly, for a M' which is not an AND-variable, we extend the definition of $d_\wedge(M')$, by setting $d_\wedge(M')$ to be $\max_i\{d_\wedge(M_i)\}$, where M_i is an AND-variables and there is a path from M_i to M' in the classical circuit.

For an AND-variable M with $d_\wedge(M) = d \geq 3$, we have $M = M_1M_2$ for some M_1 with $d_\wedge(M_1) = d - 1$. If $d_\wedge(M_2) \leq d - 3$, we further decompose M_1 to variables with lower AND-depth. That is we write M_1 as $\sum_{i,j} M_i'M_j' + \sum_k M_k'$, where $d_\wedge(M_i') \leq d-2, d_\wedge(M_j') \leq d-2, d_\wedge(M_k') \leq d-2$, for any i, j, k. Then, we have

$$M = \sum_{i,j} M_i'(M_j'M_2) + \sum_k M_k'M_2.$$

For any M_i' with $d_\wedge(M_i^1) = d-2$, if the corresponding M_j' always satisfies $d_\wedge(M_j^1) \leq d-3$, then by constructing some new operations which generate those $M_j'M_2$ first, we can reduce the AND-depth of M from d to $d-1$.

6.3 *T*-depth-4 and *T*-depth-3 Quantum Circuits for the AES S-box

Firstly, based on the classical circuit proposed in [BP12] with AND-depth 4 (see Supplementary Material C), we can build a quantum circuit for the AES S-box with T-depth 4 by employing the method given in Section 6.1. In comparison, the T-depth of the quantum circuit presented by Jaques et al. [JNRV20] based on the same classical circuit is 6, and its width is the same as ours.

Furthermore, based on the trick given in Section 6.2, we transform the classical circuit proposed in [BP12] into the circuit shown in Supplementary Material D with AND-depth 3, based on which a T-depth-3 quantum circuit for the AES S-box can be constructed. Note that, the algebraic degree of the AES S-box is 7. With two layers of multiplications, we can only obtain polynomials with degree 4. This means that we need at least three layers of multiplications to generate the output of the AES S-box. Therefore, if we implement the AES S-box by firstly designing a classical reversible circuit, and then decomposing each gate into the Clifford+T gates, the minimum T-depth is 3, which is achieved by our circuit. A comparison of our circuits and the one presented in [JNRV20] is presented in Table 4. The Q# code for our T-depth-4 and T-depth-3 quantum circuits are available at https://github.com/AES-quantum-circuit/AES-quantum-circuit.

Table 4. Quantum resources for different AES S-box circuits

#CNOT	#1qClifford	#T	# Measure	T-depth	Full depth	Width	Source
664	205	136	34	**6**	**117**	136	[JNRV20]
718	208	136	34	**4**	**109**	136	T-depth-4
1395	467	312	78	**3**	**113**	218	T-depth-3

Remark 3. The widths presented in Table 4 are not obtained by the Q# resource estimator. As mentioned in the latest ePrint version of [JNRV20] and https://github.com/microsoft/qsharp-runtime/issues/192. There was a bug in Q# that produces conflict width and depth estimations, and this issue was solved in the latest version of Q#. However, when Q# tries to optimize the T-depth, the width obtained is not optimal (https://github.com/microsoft/qsharp-runtime/pull/446). Therefore, we manually estimate the widths to obtain more accurate figures. The specific estimation process can be found in Supplementary Material F.

7 Efficient Quantum Circuits for AES

To implement an iterative block cipher, we proceed as follows. Firstly, we choose the pipeline structure or the round-in-place structure according to our optimization objective (low depth or low width). Then, implement the linear layers with Xiang et al.'s method, the PLU decomposition method, or the SAT-based technique presented in this paper. For the nonlinear components, construct $\mathfrak{C}^0$-circuits and then convert them to $\mathfrak{C}^*$ circuits. Finally, plug these sub-circuits into the round-level structure. We will show case this procedure with AES-128, and all the techniques can be easily extended to AES-192, AES-256, and other iterative block ciphers.

7.1 Low-width Quantum circuits for AES

First of all, we choose the round-in-place structure according to observation 2 of Section 3. Then, we show how to implement the building blocks of AES in-place. The `ShiftRow` and `RotByte` operations can be easily implemented by rewiring. For the `MixColumns` operation, regarded as a 32×32 binary matrix, we employ the in-place circuit from [XZL+20], which requires 92 CNOT gates. In the following, we consider the implementations of the S-boxes, for which different circuits are used in different situations.

S-boxes in the Key Schedule Data Path. Since the S-box is immediately followed by an XOR operation in the key schedule (a Feistel-like transformation), we only need a a $\mathfrak{C}^*$-circuit of the S-box. In our implementation, we used the $\mathfrak{C}^*$-circuit introduced in Section 5.2.

For the sake of simplicity, we call a $\mathfrak{C}^*$-circuit (or $\mathfrak{C}^0$-circuit) of the AES S-box a $\mathfrak{C}^*$ (or $\mathfrak{C}^0$) S-box. Figure 14 illustrates the structure of the in-place circuit for the AES key schedule. In this figure, `SubByte` represents the sub-circuit for the parallel application of four $\mathfrak{C}^*$ S-boxes. Note that while the implementations of the S-boxes are improved in this paper, the high-level structure is attributed to [JNRV20].

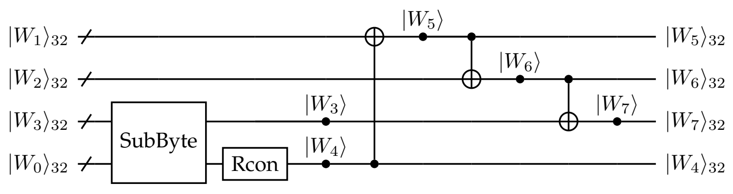


Fig. 14. An in-place circuit for generating the first round key

S-boxes in the Encryption Data Path. In the encryption process of AES, `ByteSub` can be regarded as a substitution-like transformation defined in Section 4, and thus we can implement it with the OP-based in-place circuit. This means we need to construct a $\mathfrak{C}^0$ S-box and a $\mathfrak{C}^0$ S-box^{-1}. Here, we use the $\mathfrak{C}^0$ S-box proposed in [ZWS+20], based on which a $\mathfrak{C}^0$ S-box^{-1} can be constructed as follows.

Suppose $x \in \mathbb{F}_2^8$ is the input of the S-Box, then y, the output of the S-box, is equal to $LS_0(x)+c$, where L is a linear function and $S_0(x)$ is the inverse of x in $\mathbb{F}_2^8$. Hence, we have $x = S_0^{-1}L^{-1}(y+c) = S_0L^{-1}(y+c) = L^{-1}(LS_0)L^{-1}(y+c)$.

Suppose U_0 is the circuit that implements $|x\rangle|0\rangle|0\rangle \rightarrow |x\rangle|LS_0(x)\rangle|0\rangle$. Obviously, it can be generated from a $\mathfrak{C}^0$ S-box by deleting the last 4 Pauli-X gates. Then, it is easy to check that the circuit in Figure 15 is a $\mathfrak{C}^0$-circuit of S-box^{-1}.

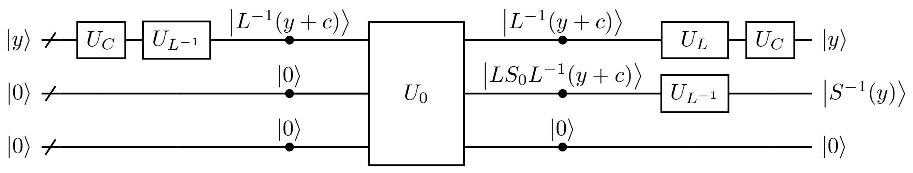


Fig. 15. The circuit for implementing the S-box^{-1} of AES

In Figure 15, U_C is the circuit consisting of 4 Pauli-X gates, and it implements constant addition of c. U_L and $U_{L^{-1}}$ are the circuits consisting of CNOT gates, and they implement the linear transformation L and L^{-1} respectively. According to our SAT-based method, L can be implemented by 14 CNOT gates. Consequently, we can implement a $\mathfrak{C}^0$-circuit of S-Box^{-1} with 6 ancilla qubits, 52 Toffoli gates, 41 Toffoli depth, 368 CNOT gates, and 8 NOT gates[6]. In Supplementary Material E, we present the specific form of the matrix corresponding to L, and the quantum circuit that implements L with minimal number of CNOT gates.

Based on the above circuits, we can in-place implement `ByteSub` in each round by 16 OP-based in-place circuits of the S-box. We suppose these 16 in-place circuits are implemented in parallel, then our implementation of `ByteSub` has the following two phases:

Phase 1: Implement 16 $\mathfrak{C}^0$ S-box, denoted by $\texttt{ByteSub}_1$;
Phase 2: Implement 16 reverse circuits of $\mathfrak{C}^0$ S-box^{-1}, denoted by $\texttt{ByteSub}^{-1}$.

Note that in the key schedule process of each round, we need to apply 4 $\mathfrak{C}^*$ S-boxes. Obviously, by applying 2 of them in **Phase 1**, and another 2 of them in **Phase 2**, we can reduce the DW-cost of the whole circuit. Under this strategy, our implementation of the i-th round of AES can be illustrated by Figure 16.

In this figure, k_{i-1} denotes the round key in the $(i-1)$-th round, and c_{i-1} denotes the output state of the $(i-1)$-th round. The last step of each round

[6] The C code for checking the correctness of this $\mathfrak{C}^0$-circuits of S-box $^{-1}$ is available at https://github.com/AES-quantum-circuit/AES-quantum-circuit

is `AddRoundKey`, which can be implemented by applying 128 CNOT gates in parallel, and denoted by a CNOT gate in Figure 16. For the final round, we do not need the sub-circuit `Mixcol`. Round 0, which performs a bitwise XOR of k_0 to the plaintext, can be implemented by applying 128 CNOT gates in parallel.

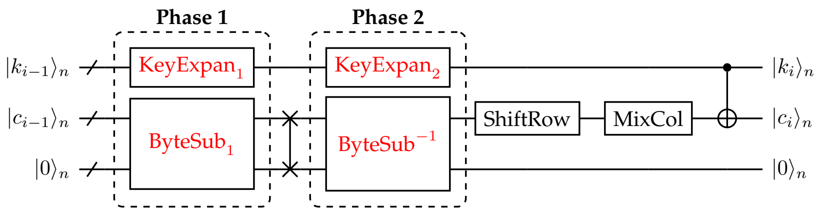


Fig. 16. The in-place implementation of the i-th round of AES-128

Note that the ancilla qubits used in these S-box circuits are ignored in this figure. From Table 3, we know that the number of ancilla qubits used in our out-of-place ($\mathfrak{C}^0$ or $\mathfrak{C}^*$) S-box circuit is 6, and the Toffoli-depth of this circuit is 41. We implement 18 out-of-place S-box circuits simultaneously in a phase, and the third register uses $8 \times 16 = 128$ qubits to store the outputs of the 16 out-of-place S-box circuits in `ByteSub`$_1$, hence the width of this one round circuit is $18 \times 6 + 3 \times 128 = 492$, and the Toffoli-depth is $41 \times 2 = 82$.

Apparently, for this one round circuit, we can make a tradeoff between width and depth by reducing the number of S-box circuits applied in parallel. In Figure 16, the Toffoli-depth for sequentially implementing one $\mathfrak{C}^*$ S-box in `KeyExpan`$_1$ and one $\mathfrak{C}^*$ S-box in `KeyExpan`$_2$ is the same as the Toffoli-depth of an OP-based in-place S-box circuit. Hence, we see two sequential $\mathfrak{C}^*$ S-box as a whole circuit, and in the following call such circuit and the OP-based in-place S-box circuit, *double-depth S-box circuits*. In this case, the process of **Phase 1** and **Phase 2** in Figure 16 implements 18 double-depth S-box circuits in parallel. Now suppose we implement p double-depth S-box circuits in parallel, where $p|18$.

- If $p = 9$, the Toffoli-depth is $82 \times 2 = 164$. In the 9 double-depth S-box circuits applied in the same layer, one of them is in `KeyExpan` and eight of them are in `ByteSub`, hence the width is $128 \times 2 + 6 + (8 + 6) \times 8 = 374$.
- If $18/p \geq 3$, the Toffoli-depth is $82 \times 18/p = 1476/p$. the widest part of such one round circuit is a phase in which all p double-depth S-box circuits are in `ByteSub`, and the width is $2 \times 128 + (8 + 6)p = 256 + 14p$.

Table 5 present the numbers of different quantum gates used in each component and one round. Obviously, these numbers are irrelevant to p.

Implement the Grover Oracle. If we want to construct a Grover oracle to search k_0, since the plaintext m is fixed and Round 0 is adding k_0 on m, we can apply Pauli-X gates on some specific ones of the wires carrying $|k_0\rangle$ to

Table 5. Quantum resources for implementing different components of AES

	`KeyExpan`	`MixCol`	`AddRoundKey`	`ByteSub`$_1$	`ByteSub`$^{-1}$	One Round
#Toffoli	208	0	0	832	832	1872
#CNOT	1440	368	128	5216	6096	13248
# Pauli-X	48	0	0	64	128	240

obtain $|k_0 \oplus m\rangle$, then when $|k_0\rangle$ is needed later, apply Pauli-X gates on these wires again to convert $|k_0 \oplus m\rangle$ back to $|k_0\rangle$. Therefore, we can use the circuit in Figure 17 to implement Round 0 and Round 1 together.

Compare to Figure 16, in this circuit, we do not need to implement `ByteSub`$^{-1}$, hence we can save $16 \times 52 = 832$ Toffoli gates. Note that, the a qubits in the third register, which will be used in the following rounds, is idle in these two rounds. Hence if $p \geq 9$, we have $a > 96$, then the 16 S-box circuits in `ByteSub`$_1$ can be implemented in parallel (need 96 ancilla qubits), Similarly, `KeyExpan`, which contains 4 S-box circuits, can be implemented in parallel. However, `ByteSub`$_1$ and `KeyExpan` can not be implemented simultaneously. Therefore, if $p \geq 9$, the width and Toffoli-depth of these two rounds are 384 and 82 respectively. Moreover, these two rounds use $256+64+48 = 368$ Pauli-X gates, $5126+1440+368+128 = 7152$ CNOT gates and $208+832 = 1040$ Toffoli gates.

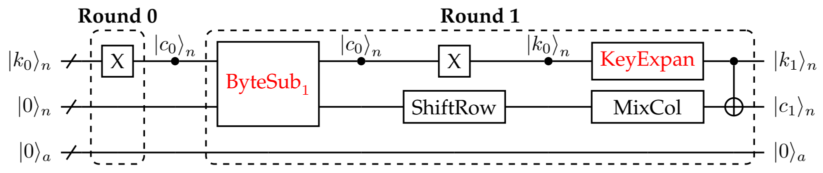


Fig. 17. The implementation of the round 0 and round 1 of AES

Then, by combining the circuits in Figure 17 and Figure 16, we can implement encryption circuit of the AES Grover oracle. In Table 6, we present the quantum resources needed for this circuit with $p = 18$ and $p = 9$, and compare our results with the results presented in [ZWS$^+$20].

Table 6. Quantum resources for implementing AES-128

Width	Toffoli-Depth	#Toffoli	#CNOT	#Pauli-X	source
512	2016	19788	128517	4528	[ZWS$^+$20]
492	820	17888	126016	2528	$p = 18$
374	1558	17888	126016	2528	$p = 9$

7.2 Low-depth Quantum Circuits for AES

To reduce the depth, we should use the pipeline structure. First, we consider the nonlinear components. Since the round transformation is implemented out-of-place in the pipeline structure, for implementing `ByteSub`, we only need a low T-depth $\mathfrak{C}^0$ S-box. For the AES key schedule, since it is the Feistel-like non-linear transformation, as shown in Section 7.1, it can be implemented in-place with the circuit in Fig. 14. It is easy to see that compared to other out-of-place implementations of the AES key schedule, the depth of such in-place implementation is also lower, since no extra operations are needed. As a consequence, we also in-place implement the AES key schedule in our shallower circuit of AES based on a $\mathfrak{C}^*$ S-box.

In Section 6.3, we presented two circuits of the AES S-box, which have T-depth 4 and T-depth 3 respectively. In these two circuits, the output wires are only be used as target wires. Therefore, these circuits are both $\mathfrak{C}^*$-circuits, and can be used in `ByteSub` and the key schedule.

For `MixColoumns`, we use the in-place circuit in Section 7.1, which has the optimal width, and the lowest CNOT-count until now. The Q# resource estimator shows that the depth of this in-place circuit is 30. In our resource estimation model, the CNOT-depth metric is less important than other metrics. For these reasons, we use this in-place circuit to implement `MixColoumns`.

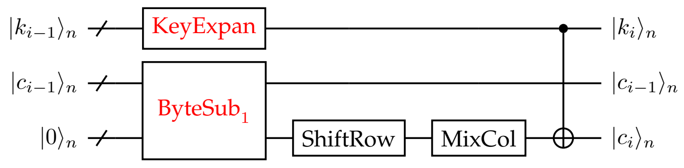


Fig. 18. The out-of-place implementation of the i-th round of AES-128

Fig. 18 presents our implementation of the i-th round. In `KeyExpan` and $\texttt{ByteSub}_1$, 20 S-box circuits are applied in parallel. The round 0 is implemented by applying 128 CNOT gates in parallel, which maps $|k_0\rangle|m\rangle$ to $|k_0\rangle|c_0\rangle$. Note that, we don't use the circuit in Fig. 17 to implement the round 0 and round 1 together. The reason is in Fig. 17, `KeyExpan` and $\texttt{ByteSub}_1$ cannot be applied in parallel, hence the T-depth and full depth will be higher.

We implemented our AES circuits by Q# based on the code proposed in [JNRV20], and our code of `Mixcolumn` and the S-box (`https://github.com/AES-quantum-circuit/AES-quantum-circuit`). Table 7 shows the quantum resources of our circuits based on different S-box implementations. Same as in [JNRV20], the results presented here are the quantum resources required for implementing the forward circuit, which outputs the ciphertext, and the reverse circuit, which is used for uncomputation. As in Table 4, except the width[7], other

[7] In Supplementary Material F, we show how to obtain these values of widths.

values are obtain from Q# resource estimator. We can see that, similarly as the results of the S-box circuits, the T-depths and the full depths of our circuits are all lower than those in [JNRV20].

Table 7. Quantum resources for implementing AES and AES†

#CNOT	#1qClifford	#T	#M	T-depth	Full depth	width	source
291150	83116	54400	13600	**120**	**2827**	3936	[JNRV20]
298720	83295	54400	13600	**80**	**2198**	3936	with T-depth-4 S-box
570785	189026	124800	31200	**60**	**2312**	5576	with T-depth-3 S-box

8 General Width and T-depth Trade-offs

By combing different S-box circuits with different structures, and adjusting the number of S-box circuits applied in parallel, we can have a spectrum of trade-offs between width and T-depth.

Since we consider the T-depth, the Toffoli gates in the out-of-place S-box circuits used in Section 7.1 should be decomposed into Clifford+T gates. Note that, we cannot replace these Toffoli gates with quantum AND gates, since the output wires of the multiplication operations are not initialized to $|0\rangle$. Here, we use the T-depth-1 Toffoli gate proposed in [Sel12], where 4 ancilla qubits are required. In these S-box circuits, we apply at most two Toffoli gates in parallel, hence we need 8 extra ancilla qubits. In all, we have a Clifford+T implementation of the S-box (or the S-box^{-1}) with $8+6=14$ ancilla qubits and T-depth 41. We name these S-box circuits as Circuit 0. Moreover, to use the T-depth-4 (or the T-depth-3) S-box circuit in the round-in-place structure, we need to construct a $\mathfrak{C}^0$-circuit of the S-box^{-1}. Obviously, we can construct such circuit with T-depth-4 (or T-depth-3), since the nonlinear parts in the classical circuits of the S-box and S-box^{-1} are the same. We name these T-depth-4 circuits as Circuit 1, and these T-depth-3 circuits as Circuit 2.

We obtain the trade-off curve shown in Figure 19 by applying Circuit 0, Circuit 1, and Circuit 2 in different structures. In this figure, Strategy 1 , 2, 3, respectively correspond to the use of Circuit 0, Circuit 1, Circuit 3 in the round-in-place structure. Strategy 4, 5, 6 respectively correspond to the use of Circuit 0, Circuit 1, Circuit 3 in the pipeline structure. Different points on a curve are obtained by applying different number of S-box circuits in parallel. We also list the results of previous works [ZWS$^+$20,JNRV20,GLRS16,LPS19] in this figure. For [ZWS$^+$20], since the T-depth is not presented, here we decompose the Toffoli gate by the same Clifford+T gates as in Circuit 0, hence slightly increase the width. For the point corresponding to [JNRV20], the width is fixed as we mentioned. The detailed process for estimating these T-depths and widths is presented in Supplementary Material G.

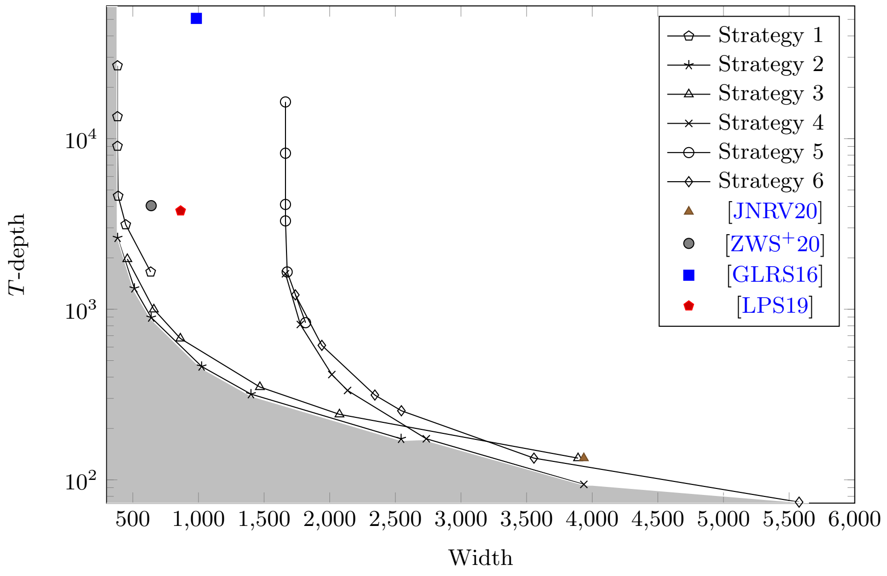


Fig. 19. The width and T-depth for implementing the Grover oracle of AES-128

9 Conclusion and Discussion

We propose the round-in-place structure for the quantum circuits of iterative ciphers, and manage to find a generic way to efficiently realize this structure in practice. We give guidelines in how to synthesize quantum circuits with specific optimization objectives based on a detailed analysis of the pipeline, zig-zag, and round-in-place structures. Moreover, new techniques for implementing the quantum circuits for linear and non-linear building blocks are presented. In particular, based on a new observation on the classical circuit of the AES S-box, we obtain a quantum circuit for the AES S-box with T-depth 3, reaching its theoretical minimum. Based on these techniques and results, we produce significantly improved quantum circuits for AES with respect to both depth and width. Finally, we conjecture that without optimizing across the natural hierarchical boundaries formed by the round functions of AES, the T-depth of the quantum circuit cannot be further improved.

References

AMG+16. Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John M. Schanck. Estimating the cost of generic quantum preimage attacks on SHA-2 and SHA-3. In Roberto Avanzi and Howard M. Heys, editors, *Selected Areas in Cryptography - SAC 2016*, volume 10532 of *Lecture Notes in Computer Science*, pages 317–337. Springer, 2016.

AMMR13. Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 32(6):818–830, 2013.

ASAM18. Mishal Almazrooie, Azman Samsudin, Rosni Abdullah, and Kussay N. Mutter. Quantum reversible circuit of AES-128. *Quantum Inf. Process.*, 17(5):112, 2018.

BBvHL21. Gustavo Banegas, Daniel J. Bernstein, Iggy van Hoof, and Tanja Lange. Concrete quantum cryptanalysis of binary elliptic curves. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):451–472, 2021.

BHN$^+$19. Xavier Bonnetain, Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki, and André Schrottenloher. Quantum attacks without superposition queries: The offline Simon's algorithm. In *ASIACRYPT 2019, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, pages 552–583, 2019.

BLNS21. Xavier Bonnetain, Gaëtan Leurent, María Naya-Plasencia, and André Schrottenloher. Quantum linearization attacks. In *Advances in Cryptology - ASIACRYPT 2021, Singapore, December 6-10, 2021, Proceedings, Part I*, pages 422–452, 2021.

BNS19a. Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. On quantum slide attacks. In *SAC 2019*, pages 492–519, 2019.

BNS19b. Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. Quantum security analysis of AES. *IACR Trans. Symmetric Cryptol.*, 2019(2):55–93, 2019.

BP12. Joan Boyar and René Peralta. A small depth-16 circuit for the AES s-box. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 287–298. Springer, 2012.

CNS17. André Chailloux, María Naya-Plasencia, and André Schrottenloher. An efficient quantum collision search algorithm and implications on symmetric cryptography. In *ASIACRYPT 2017, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, pages 211–240, 2017.

Fow12. Austin G Fowler. Time-optimal quantum computation. *arXiv preprint arXiv:1210.4626*, 2012.

FS10. Carsten Fuhs and Peter Schneider-Kamp. Synthesizing shortest linear straight-line programs over GF(2) using SAT. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 71–84. Springer, 2010.

GLRS16. Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying grover's algorithm to AES: quantum resource estimates. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - PQCrypto 2016*, volume 9606 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2016.

Gro96. Lov K. Grover. A fast quantum mechanical algorithm for database search. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, 1996*, pages 212–219. ACM, 1996.

HS. Akinori Hosoyamada and Yu Sasaki. Finding hash collisions with quantum computers by using differential trails with smaller probability than birthday bound. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106, pages 249–279. Springer.

HS18a. Akinori Hosoyamada and Yu Sasaki. Cryptanalysis against symmetric-key schemes with online classical queries and offline quantum computations. In *CT-RSA 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*, pages 198–218, 2018.

HS18b. Akinori Hosoyamada and Yu Sasaki. Quantum Demiric-Selçuk Meet-in-the-Middle Attacks: Applications to 6-Round Generic Feistel Constructions. In *SCN 2018*, pages 386–403, 2018.

IBM. IBM Qiskit. Open-source quantum development. https://https://qiskit.org/.

JNRV20. Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing Grover oracles for quantum key search on AES and lowmc. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020*, volume 12106 of *Lecture Notes in Computer Science*, pages 280–310. Springer, 2020.

KLLN16a. Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 Proceedings*, pages 207–237. Springer, 2016.

KLLN16b. Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Quantum differential and linear cryptanalysis. *IACR Trans. Symmetric Cryptol.*, 2016(1):71–94, 2016.

LPS19. Brandon Langenberg, Hai Pham, and Rainer Steinwandt. Reducing the cost of implementing AES as a quantum circuit. *IACR Cryptol. ePrint Arch.*, page 854, 2019.

Mic. Microsoftt Q#. Quantum development. https://devblogs.microsoft.com/qsharp/.

MSM18. Giulia Meuli, Mathias Soeken, and Giovanni De Micheli. Sat-based {CNOT, T} quantum circuit synthesis. In Jarkko Kari and Irek Ulidowski, editors, *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings*, pages 175–188, 2018.

NC16. Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2016.

NIS16. NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016. Available at https://csrc.nist.gov/projects/post-quantum-cryptography.

NS. María Naya-Plasencia and André Schrottenloher. Optimal merging in quantum k-xor and k-xor-sum algorithms. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106, pages 311–340. Springer.

PMH08. Ketan N. Patel, Igor L. Markov, and John P. Hayes. Optimal synthesis of linear reversible circuits. *Quantum Inf. Comput.*, 8(3):282–294, 2008.

Sel12. Peter Selinger. Quantum circuits of t-depth one. *CoRR*, abs/1210.0974, 2012.

Sho99. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.*, 41(2):303–332, 1999.

SPMH03. Vivek V. Shende, Aditya K. Prasad, Igor L. Markov, and John P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 22(6):710–722, 2003.

Sto16. Ko Stoffelen. Optimizing S-Box implementations for several criteria using SAT solvers. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 140–160. Springer, 2016.

XZL+20. Zejun Xiang, Xiangyong Zeng, Da Lin, Zhenzhen Bao, and Shasha Zhang. Optimizing implementations of linear layers. *IACR Trans. Symmetric Cryptol.*, 2020(2):120–145, 2020.

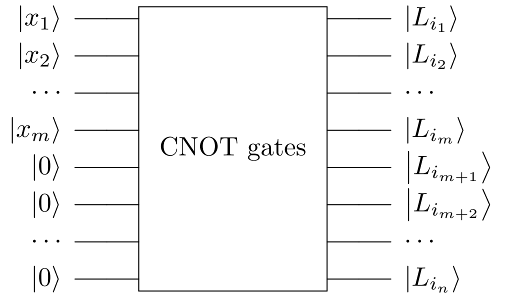
ZWS+20. Jian Zou, Zihao Wei, Siwei Sun, Ximeng Liu, and Wenling Wu. Quantum circuit implementations of AES with fewer qubits. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020*, pages 697–726. Springer, 2020.

Supplementary Material

A A SAT-based Method for Finding Optimal CNOT Circuits

Suppose we have m Boolean variables $x_1, x_2, \ldots, x_m$. We want to obtain n linear forms $L_1(x_1, x_2, \ldots, x_m), L_2(x_1, x_2, \ldots, x_m), \ldots, L_n(x_1, x_2, \ldots, x_m)$ by using operations $y_i = y_i \oplus y_j$, where $y_1, y_2, \ldots, y_m, y_{m+1}, y_{m+2}, \ldots, y_n$ are initialized to $x_1, x_2, \ldots, x_m, 0, 0, \ldots, 0$ respectively.

The above process is equivalent to the following quantum circuit, where $\{i_1, i_2, \ldots, i_n\}$ is a permutation of $\{1, 2, \ldots, n\}$. Our purpose is finding a such circuit with minimal number of CNOT gates.



Suppose we want to find a circuit with k CNOT gates, then we can define 5 matrices:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{k1} & b_{k2} & \cdots & b_{kn} \end{bmatrix}, \quad C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{k1} & c_{k2} & \cdots & c_{kn} \end{bmatrix},$$

$$F = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nn} \end{bmatrix}, \quad \Phi = \begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1n} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_{k1} & \varphi_{k2} & \cdots & \varphi_{kn} \end{bmatrix}.$$

A is the coefficient matrix such that $A(x_1, x_2, \ldots, x_m)^T = (L_1, L_2, \ldots, L_n)^T$. B and C are matrices used to represent the XOR operations. In each row of B or C, there is exactly one nonzero entry. Moreover, if in the i-th row, we have $b_{ij_1} = c_{ij_2} = 1$, then it means at the i-th step the operation is $y_{j_2} = y_{j_2} \oplus y_{j_1}$. Based on the above definitions, we can generate the following equations about the entries of B and C:

$$EQN_b = \left\{ \begin{array}{c} b_{ij_1} b_{ij_2} = 0, \\ \\ b_{i1} \oplus b_{i2} \oplus \cdots \oplus b_{in} \oplus 1 = 0, \\ \\ \text{for } 1 \le i \le k, 1 \le j_1 \ne j_2 \le n \end{array} \right\}$$

$$EQN_c = \left\{ \begin{array}{c} c_{ij_1} c_{ij_2} = 0, \\ \\ c_{i1} \oplus c_{i2} \oplus \cdots \oplus c_{in} \oplus 1 = 0, \\ \\ \text{for } 1 \le i \le k, 1 \le j_1 \ne j_2 \le n \end{array} \right\}$$

Now we consider F, which is a matrix describing the relations between the final expression of y_i and L_i. That is, if $y_j = L_i(x_1, x_2, \ldots, x_m)$, then f_{ij} is set to 1. This means there is exactly one nonzero entry in each row of F. Similarly, we have the following equations about f_{ij}:

$$EQN_f = \left\{ \begin{array}{c} f_{ij_1} f_{ij_2} = 0, \\ \\ f_{i1} \oplus f_{i2} \oplus \cdots \oplus f_{in} \oplus 1 = 0, \\ \\ \text{for } 1 \le i \le n, 1 \le j_1, j_2 \le n, j_1 \ne j_2 \end{array} \right\}$$

$\varPhi$ is a matrix whose entries are linear forms of $x_1, x_2, \ldots, x_m$. φ_{ij} is the expression of y_j after the i-th operation. Then we introduce a group of Boolean variables $\psi_{j,i,s}$, $1 \le j \le n, 1 \le i \le k, 1 \le s \le m$. The value of $\psi_{i,j,s}$ is the coefficient of φ_{ij} w.r.t. x_s . Hence, we have the following relations:

- If $c_{ij} = 0$, then $\psi_{i,j,s} = \psi_{i-1,j,s}$,
- If $c_{ij} = 1$, then $\psi_{i,j,s} = \sum\limits_{t=1}^{n} b_{it} \psi_{i-1,t,s} + \psi_{i-1,j,s}$.

This induces the following equations:

$$EQN_\psi = \left\{ \begin{array}{c} \psi_{i,j,s} + \sum\limits_{t=1}^{n} c_{ij} b_{it} \psi_{i-1,t,s} + \psi_{i-1,j,s} = 0, \\ \\ \text{for } 1 \le i \le k, 1 \le j \le n, 1 \le s \le m \end{array} \right\}$$

Furthermore, since $\{\varphi_{k1}, \varphi_{k2}, \ldots, \varphi_{kn}\}$ is a permutation of $\{L_1, L_2, \ldots, L_n\}$, we have $\psi_{k,j,s} = a_{is}$, if $f_{ij} = 1$. This can be represented by the following equations:

$$EQN_a = \left\{ \begin{array}{c} f_{i,j}(\psi_{k,j,s} + a_{is}) = 0 \\ \\ \text{for } 1 \le i \le n, 1 \le j \le n, 1 \le s \le m \end{array} \right\}$$

It is obvious that, finding a circuit with k CNOT gates which outputs $|L_{i_1}\rangle|L_{i_2}\rangle \cdots |L_{i_n}\rangle$, is equivalent to finding a common solution of the Boolean equations $\{EQN_b, EQN_c, EQN_f, EQN_\psi, EQN_a\}$.

In our experiments, we used the SAT-solver Cryptominisat as a solver for solving this kind of equation system[8]. If the solver returns `Unsatisfiable`, it means that no circuit with k CNOT gates can implement these linear forms. Then we have the following algorithm to find k_{min}, the minimal number of CNOT gates, and the corresponding B, C, F. Furthermore, from B, C, F, we can achieve a circuit with k_{min} CNOT gates.

Algorithm 1:

```
1 Set $k$ to 1;
2 Solve the corresponding Boolean equation system
    $\{EQN_b, EQN_c, EQN_f, EQN_\psi, EQN_f\}$;
3 if Unsatisfiable then
4     $k \leftarrow k + 1$ and goto 2
5 else
6     return $k$ and the matrices $B$, $C$, $F$
```

B The Proof of Theorem 1

Proof. If $M_i \rightarrow M_j$, it is obvious that the AND gate that outputs M_i should be applied before the AND gate that outputs M_j. Consequently, if we have a longest path $M_{k_1} \rightarrow M_{k_2} \rightarrow \cdots \rightarrow M_{k_s}$, then the AND gates which generate $M_{k_1}, M_{k_2}, \ldots, M_{k_s}$ should be applied one by one. This means the AND-depth and the T-depth of the quantum circuit which implements all these $M_{k_1}, M_{k_2}, \ldots, M_{k_s}$ are not smaller than s.

In the following, we show how to construct a quantum circuit with AND-depth s. Based on different AND-depths, we can divide all these M_i into disjoint sets $Set_1 = \{M_1^1, M_2^1, \ldots, M_{r_1}^1\}, Set_2 = \{M_1^2, M_2^2, \ldots, M_{r_2}^2\}, \ldots, Set_s = \{M_1^t, M_2^t, \ldots, M_{r_t}^t\}$, where $M_j^i \in Set_i$ has AND-depth i. We show that all $M_j^1 \in Set_1$, $1 \leq j \leq r_1$, can be achieved by applying r_1 AND gates in parallel. Obviously, M_j^1 is not the AND-successor of any other variable. This means the variables required to generate these M_j^1 can be achieved only by additions, which can be implemented by CNOT and Pauli-X gates. Assume there are two m-variables which need the same variable in the corresponding multiplication operations, for example $M_1^1 = M_1 \cdot M_2$ and $M_2^1 = M_1 \cdot M_3$. To generate M_1^1 and M_2^1 in parallel, we can copy M_1 to an ancilla qubit by a CNOT gate. Note

[8] `https://github.com/msoos/cryptominisat/`

that we assumed there are enough ancilla qubits, thus this operation always works. Therefore, by applying some CNOT gates, we can guarantee the variables needed to generate all these M_j^1 are on different wires. Then, by applying s_1 AND gates in parallel, we can achieve all these M_j^1. This means by a circuit with AND-depth 1, we can generate all these M_j^1. After this step, we can clean these ancilla qubits by CNOT gates.

Now we prove our conclusion by induction. Suppose we can generate all variables in $Set_1, Set_2, \ldots, Set_{d-1}$ by a circuit with $d-1$ AND layers. For any M_j^d in Set_d, suppose M_j^d is the AND-successor of some M_1. If $d_\wedge(M_1) \geq d$, then there is a path such that M_j^d is after the d-th position. In this case, we can construct a longer path than the longest path, a contradiction. This means $d_\wedge(M_1) < d$, thus M_1 is in some Set_i, $1 \leq i < d$. We can induce that all the variables needed for generating M_j^d can be achieved by applying the previous circuit which has $d-1$ AND layers. Similarly as the case of Set_1, if a variable is needed in several multiplication operations, we copy it to ancilla qubits by CNOT gates. Then by applying s_d AND gates in parallel, we can obtain all the variables in Set_d. Obviously, this circuit generates all variables in $Set_1, Set_2, \ldots, Set_d$, and has d AND layers. Finally, we can construct a circuit which implements the classical circuit with s AND layers, then by implementing each quantum AND gate with a T-depth-1 circuit, we can obtain a T-depth s circuit. This proves our theorem.

C Boyar and Peralta's Classical Circuit for the AES S-box

Table 8. Boyar and Peralta's classical circuit for the AES S-box

Top Linear Part:				
$T_1 = U_0 + U_3$	$T_2 = U_0 + U_5$	$T_3 = U_0 + U_6$	$T_4 = U_3 + U_5$	$T_5 = U_4 + U_6$
$T_6 = T_1 + T_5$	$T_7 = U_1 + U_2$	$T_8 = U_7 + T_6$	$T_9 = U_7 + T_7$	$T_{10} = T_6 + T_7$
$T_{11} = U_1 + U_5$	$T_{12} = U_2 + U_5$	$T_{13} = T_3 + T_4$	$T_{14} = T_6 + T_{11}$	$T_{15} = T_5 + T_{11}$
$T_{16} = T_5 + T_{12}$	$T_{17} = T_9 + T_{16}$	$T_{18} = U_3 + U_7$	$T_{19} = T_7 + T_{18}$	$T_{20} = T_1 + T_{19}$
$T_{21} = U_6 + U_7$	$T_{22} = T_7 + T_{21}$	$T_{23} = T_2 + T_{22}$	$T_{24} = T_2 + T_{10}$	$T_{25} = T_{20} + T_{17}$
$T_{26} = T_3 + T_{16}$	$T_{27} = T_1 + T_{12}$			
Nonlinear Part:				
$M_1 = T_{13} \cdot T_6$	$M_2 = T_{23} \cdot T_8$	$M_3 = T_{14} + M_1$	$M_4 = T_{19} \cdot U_7$	$M_5 = M_4 + M_1$
$M_6 = T_3 \cdot T_{16}$	$M_7 = T_{22} \cdot T_9$	$M_8 = T_{26} + M_6$	$M_9 = T_{20} \cdot T_{17}$	$M_{10} = M_9 + M_6$
$M_{11} = T_1 \cdot T_{15}$	$M_{12} = T_4 \cdot T_{27}$	$M_{13} = M_{12} + M_{11}$	$M_{14} = T_2 \cdot T_{10}$	$M_{15} = M_{14} + M_{11}$
$M_{16} = M_3 + M_2$	$M_{17} = M_5 + T_{24}$	$M_{18} = M_8 + M_7$	$M_{19} = M_{10} + M_{15}$	$M_{20} = M_{16} + M_{13}$
$M_{21} = M_{17} + M_{15}$	$M_{22} = M_{18} + M_{13}$	$M_{23} = M_{19} + T_{25}$	$M_{24} = M_{22} + M_{23}$	$M_{25} = M_{22} \cdot M_{20}$
$M_{26} = M_{21} + M_{25}$	$M_{27} = M_{20} + M_{21}$	$M_{28} = M_{23} + M_{25}$	$M_{29} = M_{28} \cdot M_{27}$	$M_{30} = M_{26} \cdot M_{24}$
$M_{31} = M_{20} \cdot M_{23}$	$M_{32} = M_{27} \cdot M_{31}$	$M_{33} = M_{27} + M_{25}$	$M_{34} = M_{21} \cdot M_{22}$	$M_{35} = M_{24} \cdot M_{34}$
$M_{36} = M_{24} + M_{25}$	$M_{37} = M_{21} + M_{29}$	$M_{38} = M_{32} + M_{33}$	$M_{39} = M_{23} + M_{30}$	$M_{40} = M_{35} + M_{36}$
$M_{41} = M_{38} + M_{40}$	$M_{42} = M_{37} + M_{39}$	$M_{43} = M_{37} + M_{38}$	$M_{44} = M_{39} + M_{40}$	$M_{45} = M_{42} + M_{41}$
$M_{46} = M_{44} \cdot T_6$	$M_{47} = M_{40} \cdot T_8$	$M_{48} = M_{39} \cdot U_7$	$M_{49} = M_{43} \cdot T_{16}$	$M_{50} = M_{38} \cdot T_9$
$M_{51} = M_{37} \cdot T_{17}$	$M_{52} = M_{42} \cdot T_{15}$	$M_{53} = M_{45} \cdot T_{27}$	$M_{54} = M_{41} \cdot T_{10}$	$M_{55} = M_{44} \cdot T_{13}$
$M_{56} = M_{40} \cdot T_{23}$	$M_{57} = M_{39} \cdot T_{19}$	$M_{58} = M_{43} \cdot T_3$	$M_{59} = M_{38} \cdot T_{22}$	$M_{60} = M_{37} \cdot T_{20}$
$M_{61} = M_{42} \cdot T_1$	$M_{62} = M_{45} \cdot T_4$	$M_{63} = M_{41} \cdot T_2$		
Bottom Linear Part:				
$L_0 = M_{61} \oplus M_{62}$	$L_1 = M_{50} \oplus M_{56}$	$L_2 = M_{46} \oplus M_{48}$	$L_3 = M_{47} \oplus M_{55}$	$L_4 = M_{54} \oplus M_{58}$
$L_5 = M_{49} \oplus M_{61}$	$L_6 = M_{62} \oplus L_5$	$L_7 = M_{46} \oplus L_3$	$L_8 = M_{51} \oplus M_{59}$	$L_9 = M_{52} \oplus M_{53}$
$L_{10} = M_{53} \oplus L_4$	$L_{11} = M_{60} \oplus L_2$	$L_{12} = M_{48} \oplus M_{51}$	$L_{13} = M_{50} \oplus L_0$	$L_{14} = M_{52} \oplus M_{61}$
$L_{15} = M_{55} \oplus L_1$	$L_{16} = M_{56} \oplus L_0$	$L_{17} = M_{57} \oplus L_1$	$L_{18} = M_{58} \oplus L_8$	$L_{19} = M_{63} \oplus L_4$
$L_{20} = L_0 \oplus L_1$	$L_{21} = L_1 \oplus L_7$	$L_{22} = L_3 \oplus L_{12}$	$L_{23} = L_{18} \oplus L_2$	$L_{24} = L_{15} \oplus L_9$
$L_{25} = L_6 \oplus L_{10}$	$L_{26} = L_7 \oplus L_9$	$L_{27} = L_8 \oplus L_{10}$	$L_{28} = L_{11} \oplus L_{14}$	$L_{29} = L_{11} \oplus L_{17}$
$S_0 = L_6 \oplus L_{24}$	$S_1 = L_{16} \oplus L_{26} \oplus 1$	$S_2 = L_{19} \oplus L_{28} \oplus 1$	$S_3 = L_6 \oplus L_{21}$	$S_4 = L_{20} \oplus L_{22}$
$S_5 = L_{25} \oplus L_{29}$	$S_6 = L_{13} \oplus L_{27} \oplus 1$	$S_7 = L_6 \oplus L_{23} \oplus 1$		

In this circuit, $U_0, U_1, \ldots, U_7$ are the input and $S_0, S_1, \ldots, S_7$ are the output.

D An AND-depth-3 Classical Circuit for the AES S-box

Table 9. The nonlinear part of an AND-depth-3 classical circuit of the AES S-box

Nonlinear Part:				
$M_1 = T_{13} \cdot T_6$,	$M_2 = T_{23} \cdot T_8$,	$M_3 = T_{14} \oplus M_1$,	$M_4 = T_{19} \cdot U_7$,	$M_5 = M_4 \oplus M_1$,
$M_6 = T_3 \cdot T_{16}$,	$M_7 = T_{22} \cdot T_9$,	$M_8 = T_{26} \oplus M_6$,	$M_9 = T_{20} \cdot T_{17}$,	$M_{10} = M_9 \oplus M_6$,
$M_{11} = T_1 \cdot T_{15}$,	$M_{12} = T_4 \cdot T_{27}$,	$M_{13} = M_{12} \oplus M_{11}$,	$M_{14} = T_2 \cdot T_{10}$,	$M_{15} = M_{14} \oplus M_{11}$,
$M_{16} = M_3 \oplus M_2$,	$M_{17} = M_5 \oplus T_{24}$,	$M_{18} = M_8 \oplus M_7$,	$M_{19} = M_{10} \oplus M_{15}$,	$M_{20} = M_{16} \oplus M_{13}$,
$M_{21} = M_{17} \oplus M_{15}$,	$M_{22} = M_{18} \oplus M_{13}$,	$M_{23} = M_{19} \oplus T_{25}$,	$M_{24} = M_{22} \oplus M_{23}$,	$M_{25} = M_{22} \cdot M_{20}$,
$M_{26} = M_{21} \oplus M_{25}$,	$M_{27} = M_{20} \oplus M_{21}$,	$M_{28} = M_{23} \oplus M_{25}$,	$M_{29} = M_{20} \cdot M_{23}$,	$M_{30} = M_{27} \oplus M_{25}$,
$M_{31} = M_{21} \cdot M_{22}$,	$M_{32} = M_{24} \oplus M_{25}$,	$N_1 = M_{24} \cdot T_6$,	$N_2 = M_{23} \oplus M_{32}$,	$N_3 = M_{26} \oplus M_{31}$,
$W_1 = N_3 \cdot N_1$,	$W_2 = N_2 \cdot T_6$,	$M_{33} = W_1 \oplus W_2$,	$N_4 = M_{24} \cdot T_8$,	$N_5 = M_{32} \cdot T_8$,
$W_3 = N_4 \cdot M_{31}$,	$M_{34} = W_3 \oplus N_5$,	$N_6 = M_{24} \cdot U_7$,	$N_7 = M_{23} \cdot U_7$,	$W_4 = N_6 \cdot M_{26}$,
$M_{35} = W_4 \oplus N_7$,	$N_8 = M_{21} \oplus M_{30}$,	$N_9 = M_{28} \oplus M_{29}$,	$N_{10} = M_{27} \cdot T_{16}$,	$W_5 = N_8 \cdot T_{16}$,
$W_6 = N_9 \cdot N_{10}$,	$M_{36} = W_5 \oplus W_6$,	$N_{11} = M_{27} \cdot T_9$,	$N_{12} = M_{30} \cdot T_9$,	$W_7 = M_{29} \cdot N_{11}$,
$M_{37} = W_7 \oplus N_{12}$,	$N_{13} = M_{21} \cdot T_{17}$,	$N_{14} = M_{27} \cdot T_{17}$,	$W_8 = M_{28} \cdot N_{14}$,	$M_{38} = W_8 \oplus N_{13}$,
$N_{15} = M_{21} \oplus M_{23}$,	$N_{16} = M_{27} \cdot T_{15}$,	$N_{17} = M_{24} \cdot T_{15}$,	$W_9 = N_{15} \cdot T_{15}$,	$W_{10} = N_{16} \cdot M_{28}$,
$W_{11} = N_{17} \cdot M_{26}$,	$M'_{39} = W_9 \oplus W_{10}$,	$M_{39} = M'_{39} \oplus W_{11}$,	$N_{18} = M_{30} \oplus M_{32}$,	$N_{19} = N_{15} \oplus N_{18}$,
$N_{20} = M_{28} \oplus M_{29}$,	$N_{21} = M_{26} \oplus M_{31}$,	$N_{22} = M_{27} \cdot T_{27}$,	$N_{23} = M_{24} \cdot T_{27}$,	$W_{12} = N_{19} \cdot T_{27}$,
$W_{13} = N_{20} \cdot N_{22}$,	$W_{14} = N_{21} \cdot N_{23}$,	$M'_{40} = W_{12} \oplus W_{13}$,	$M_{40} = M'_{40} \oplus W_{14}$,	$N_{24} = M_{27} \cdot T_{10}$,
$N_{25} = M_{24} \cdot T_{10}$,	$W_{15} = M_{29} \cdot N_{24}$,	$W_{16} = N_{18} \cdot T_{10}$,	$W_{17} = M_{31} \cdot N_{25}$,	$M'_{41} = W_{15} \oplus W_{16}$,
$M_{41} = M'_{41} \oplus W_{17}$,	$N_{26} = M_{24} \cdot T_{13}$,	$W_{18} = N_3 \cdot N_{26}$,	$W_{19} = N_2 \cdot T_{13}$,	$M_{42} = W_{18} \oplus W_{19}$,
$N_{27} = M_{24} \cdot T_{23}$,	$N_{28} = M_{32} \cdot T_{23}$,	$W_{20} = N_{27} \cdot M_{31}$,	$M_{43} = W_{20} \oplus N_{28}$,	$N_{29} = M_{24} \cdot T_{19}$,
$N_{30} = M_{23} \cdot T_{19}$,	$W_{21} = N_{29} \cdot M_{26}$,	$M_{44} = W_{21} \oplus N_{30}$,	$N_{31} = M_{27} \cdot T_3$,	$W_{22} = N_8 \cdot T_3$,
$W_{23} = N_9 \cdot N_{31}$,	$M_{45} = W_{22} \oplus W_{23}$,	$N_{32} = M_{27} \cdot T_{22}$,	$N_{33} = M_{30} \cdot T_{22}$,	$W_{24} = M_{29} \cdot N_{32}$,
$M_{46} = W_{24} \oplus N_{33}$,	$N_{34} = M_{21} \cdot T_{20}$,	$N_{35} = M_{27} \cdot T_{20}$,	$W_{25} = M_{28} \cdot N_{35}$,	$M_{47} = W_{25} \oplus N_{34}$,
$N_{36} = M_{27} \cdot T_1$,	$N_{37} = M_{24} \cdot T_1$,	$W_{26} = N_{15} \cdot T_1$,	$W_{27} = N_{36} \cdot M_{28}$,	$W_{28} = N_{37} \cdot M_{26}$,
$M'_{48} = W_{26} \oplus W_{27}$,	$M_{48} = M'_{48} \oplus W_{28}$,	$N_{38} = M_{27} \cdot T_4$,	$N_{39} = M_{24} \cdot T_4$,	$W_{29} = N_{19} \cdot T_4$,
$W_{30} = N_{20} \cdot N_{38}$,	$W_{31} = N_{21} \cdot N_{39}$,	$M'_{49} = W_{29} \oplus W_{30}$,	$M_{49} = M'_{49} \oplus W_{31}$,	$N_{40} = M_{27} \cdot T_2$,
$N_{41} = M_{24} \cdot T_2$,	$W_{32} = M_{29} \cdot N_{40}$,	$W_{33} = N_{18} \cdot T_2$,	$W_{34} = M_{31} \cdot N_{41}$,	$M'_{50} = W_{32} \oplus W_{33}$,
$M_{50} = M'_{50} \oplus W_{34}$				

The nonlinear part of our AND-depth-3 circuit is present in Table 9. The top linear part and bottom linear part are the same as those in Boyar and Peralta's classical circuit. In this table and Table 8, $M_1, M_2, \ldots, M_{30}$ are generated by the same operations. The operations labeled by the purple color are new operations. $M_{33}, M_{34}, \ldots, M_{50}$ in this table and $M_{46}, M_{47}, \ldots, M_{63}$ in Table 8 have the same algebraic expressions. In our quantum implementation, we can rewrite $M'_{39} = W_9 \oplus W_{10}$, $M_{39} = M'_{39} \oplus W_{11}$ as $M_{39} = W_9 \oplus W_{10}$, $M_{39} = M_{39} \oplus W_{11}$, and by this way, we can save one qubit and one CNOT gate. Similarly, we can rewrite the equations containing $M'_{40}, M'_{41}, M'_{48}, M'_{49}, M'_{50}$ to save qubits and CNOT gates.

E The Matrices and Circuits in Section 5.2 and Section 7.1

$$M_{A^{-1}} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&1&0&0&1&0&0&1 \\ 1&0&1&0&0&0&1&1 \\ 0&0&0&1&1&0&0&1 \\ 0&0&0&0&1&0&0&0 \\ 1&0&1&0&0&1&1&0 \\ 1&0&1&0&0&1&0&0 \\ 1&0&0&0&0&0&0&1 \end{bmatrix}$$

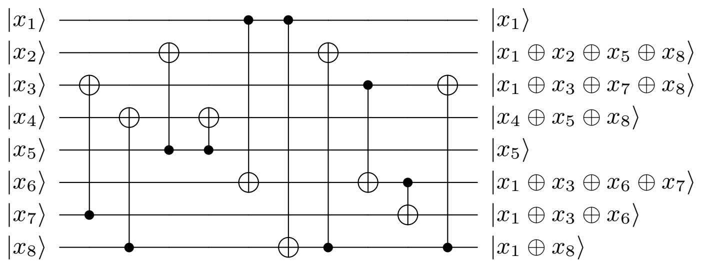

$|x_1\rangle$
$|x_2\rangle$
$|x_3\rangle$
$|x_4\rangle$
$|x_5\rangle$
$|x_6\rangle$
$|x_7\rangle$
$|x_8\rangle$
$|x_1\rangle$
$|x_1 \oplus x_2 \oplus x_5 \oplus x_8\rangle$
$|x_1 \oplus x_3 \oplus x_7 \oplus x_8\rangle$
$|x_4 \oplus x_5 \oplus x_8\rangle$
$|x_5\rangle$
$|x_1 \oplus x_3 \oplus x_6 \oplus x_7\rangle$
$|x_1 \oplus x_3 \oplus x_6\rangle$
$|x_1 \oplus x_8\rangle$

Fig. 20. The circuit for implementing A^{-1}

$$M_L = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix}$$

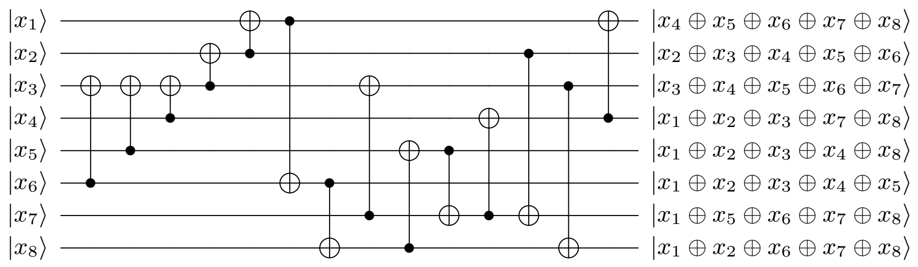

$|x_1\rangle$
$|x_2\rangle$
$|x_3\rangle$
$|x_4\rangle$
$|x_5\rangle$
$|x_6\rangle$
$|x_7\rangle$
$|x_8\rangle$
$|x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8\rangle$
$|x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6\rangle$
$|x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7\rangle$
$|x_1 \oplus x_2 \oplus x_3 \oplus x_7 \oplus x_8\rangle$
$|x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_8\rangle$
$|x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5\rangle$
$|x_1 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8\rangle$
$|x_1 \oplus x_2 \oplus x_6 \oplus x_7 \oplus x_8\rangle$

Fig. 21. The circuit for implementing L

F Width Analysis

F.1 Widths of the S-box circuits in Section 6.3

In our T-depth-4 circuit, we need 8 qubits for the input $\{U_0, U_1, \ldots, U_7\}$, 8 qubits for the output $\{S_0, S_1, \ldots, S_7\}$, and 120 qubits for intermediate variables $\{T_1, T_2 \ldots, T_{27}\}$, $\{M_1, M_2, \ldots, M_{63}\}$, $\{L_0, L_1, \ldots, L_{29}\}$. It is easy to see that the layer which needs the most qubits is the 4-th AND layer. Before this layer, we should copy 9 variables to ancilla qubits, and in this layer, we should apply 18 AND gates in parallel. Note that for implementing an AND gate with the T-depth-1 circuit in Fig. 1, we need 1 ancilla qubit. This means for these two steps we need 27 ancilla qubits. At this moment there are 38 idle qubits, which are the qubits used to store $\{L_1, L_2, \ldots, L_{30}, S_0, S_1, \ldots, S_7\}$ in Table 8, hence we do not need extra ancilla qubits. This implies that the width of the whole circuit is $8 + 8 + 120 = 136$.

In our T-depth-3 circuit, we need 16 qubits for the input and output, and 182 qubits for the intermediate variables. In this circuit, the second AND layer and the third AND layer all need a lot of ancilla qubits. In the second AND layer, we should apply 33 AND gates in parallel. Before this layer, we should copy some variables to 41 ancilla qubits. This means in the second AND layer, we need 74 ancilla qubits. At this moment there are 56 idle qubits, which are the qubits used to store $\{M_{33}, M_{34}, \ldots, M_{50}, L_1, L_2, \ldots, L_{30}, S_0, S_1, \ldots, S_7\}$ in Table 9. This means 18 extra ancilla qubits are needed. In the third AND layer, we should apply 36 AND gates in parallel, and we have to copy some variables to 22 ancilla qubits. This means in the third AND layer, we need 58 ancilla qubits. At this moment, there are 38 idle qubits, which are the qubits used to store $\{L_1, L_2, \ldots, L_{30}, S_0, S_1, \ldots, S_7\}$, thus we need 20 extra ancilla qubits. Therefore, the third AND layer needs more ancilla qubits than the second AND layer. This implies the width of the whole circuit is $16 + 182 + 20 = 218$.

For the T-depth-6 circuit proposed in [JNRV20], at most 9 AND gates are applied in parallel, and no copy operation is used. Hence only 9 ancilla qubits are needed. Obviously, we have enough idle qubits, thus the width of the whole circuit is $8 + 8 + 120 = 136$.

F.2 Widths of the AES circuits in Section 7.2

We need 128 qubits for storing the round key and another 128 qubits for storing the plaintext. Since we use the pipeline structure, for each round 128 qubits are used to store the round output, hence we need 1280 qubits for 10 rounds.

Now, consider the ancilla qubits required in the nonlinear blocks. We need to apply the 4 S-boxes in `KeyExpan` and the 16 S-boxes in $\texttt{ByteSub}_1$ in parallel. The T-depth-4 S-box circuit needs $136 - 16 = 120$ ancilla qubits, and the T-depth-3 S-box circuit needs $218 - 16 = 202$ ancilla qubits as shown in Table 4. Hence, for implementing 20 S-boxes in parallel, we need 2400 ancilla qubits when we use the T-depth-4 S-box circuit, while we need 4040 ancilla qubits when we use the T-depth-3 S-box circuit. In summary, the width of our implementation with the T-depth-4 S-box circuit is $256 + 1280 + 2400 = 3936$, and the width of our implementation with the T-depth-3 S-box circuit is $256 + 1280 + 4040 = 5576$. It is easy to see that the width of the circuit in [JNRV20] is also 3936.

G Depth-width Trade-off Analysis for Implementing Grover Oracle

We consider the Grover oracle based on one pair of plaintext and ciphertext. The sub-circuit which compares the ciphertext and obtains the oracle output can be implemented by applying 128 Pauli-X gates and a multiple controlled Toffoli (MCT) gate. Here, we should implement an MCT gate which has 128 control qubits and 1 target qubit. Since we have plenty of ancilla qubits at this step, we use a trivial low-depth Toffoli gate decomposition which has Toffoli-depth 7 for the forward circuit. This implementation needs 127 ancilla qubits. For the round-in-place structure, we can use the qubits in the third register. If in each round we apply p double-depth S-box circuits in parallel and each out-of-place S-box circuit needs a ancilla qubits, then there are more than $\frac{8}{9}p(a+8)+\frac{1}{9}pa = pa+\frac{64}{9}p$ qubits in the state $|0\rangle$ at this moment as shown in Fig. 16. This means we need no more than $127 - ap - \frac{64}{9}p$ extra ancilla qubits. Similarly, for the pipeline structure, suppose we apply p out-of-place S-box circuits in parallel, we need $127 - ap$ extra ancilla qubits.

Now suppose the out-of-place S-box circuit has T-depth d and uses a ancilla qubits, then we have the following results.

- For the round-in-place structure, suppose p double-depth S-box circuits are applied in parallel, then the T-depth of the circuit implementing the Grover oracle is $2(18d \cdot 18/p + 2d + 7)$. The width of this circuit is:
  1) $256 + pa + \frac{64}{9}p$, if $18/p < 3$ and $pa + \frac{64}{9}p \geq 127$;
  2) 383, if $18/p < 3$ and $pa + \frac{64}{9}p < 127$;
  3) $256 + p(8 + a)$, if $18/p \geq 3$ and $p(8 + a) \geq 127$;
  4) 383, if $18/p \geq 3$ and $p(8 + a) < 127$.
- For the pipeline structure, the T-depth of the circuit implementing the Grover oracle is $2(10d \cdot 20/p + 7)$. The width of this circuit is $1536 + pa$ if $pa > 127$, and is 1663, if $pa \leq 127$.