IACR Communications in Cryptology
ISSN 3006-5496, Vol. 2, No. 1, 57 pages.

https://doi.org/10.62056/ay11zo-3y

Check for updates

# Quantum Analysis of AES

Kyungbae Jang[1], Anubhab Baksi[a,2], Hyunji Kim[1], Gyeongju Song[1], Hwajeong Seo[b,1] and Anupam Chattopadhyay[3]

[1] Hansung University, Division of IT Convergence Engineering, Seoul, South Korea
[2] Lund University, Elektro- och informationsteknik, Lund, Sweden
[3] Nanyang Technological University, School of Computer Science and Engineering, Singapore

**Abstract.** Our work explores the key recovery attack using the Grover's search on the three variants of AES (-128, -192, -256). In total, we develop a pool of 26 implementations per AES variant (totaling 78), by taking the state-of-the-art advancements in the relevant fields into account.

We present the least Toffoli depth and full depth implementations of AES, thereby improving from Zou et al.'s Asiacrypt'20 paper by more than 97 percent for each variant of AES. We show that the qubit count - Toffoli depth product is reduced from theirs by more than 87 percent. Furthermore, we analyze the Jaques et al.'s Eurocrypt'20 implementations in detail, fix the bugs (arising from some problem of the quantum computing tool used), and report corrected benchmarks (which seem to improve from the authors' own bug-fixing, thanks to our architecture consideration).

To the best of our finding, our work improves from all the previous works (including the Asiacrypt'22 paper by Huang and Sun, the Asiacrypt'23 paper by Liu et al. and the Asiacrypt'24 paper by Shi and Feng) in terms of various quantum circuit complexity metrics. To be more precise, we estimate the currently best-known quantum attack complexities for AES-128 ($2^{156.2630}$), AES-192 ($2^{221.5801}$) and AES-256 ($2^{286.0731}$). Additionally, we achieve the least Toffoli depth - qubit count product for AES-128 (121920, improving from 130720 by Shi and Feng in Asiacrypt'24), AES-192 (161664, improving from 188880 by Liu et al. in Asiacrypt'23) and AES-256 (206528, improving from 248024 by Liu et al. in Asiacrypt'23) so far.

We further investigate the prospect of the Grover's search. We propose four new implementations of the S-box, one new implementation of the MixColumn; as well as five new architecture (one is motivated by the architecture by Jaques et al. in Eurocrypt'20, and the rest four are entirely our innovation). Under the MAXDEPTH constraint (specified by NIST), the circuit depth metrics (Toffoli depth, $T$-depth and full depth) become crucial factors and parallelization for often becomes necessary. We provide the least depth implementation in this respect that offers the best performance in terms of metrics like depth-squared - qubit count product, depth - gate count product.

**Keywords:** Quantum Implementation · Grover's Search · AES

This work was supported by (i) the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MIST) (No. RS-2019-II190033, Study on Quantum Security Evaluation of Cryptography based on Computational Quantum Complexity, the contribution to the project is around 66.7%); and (ii) the Wallenberg-NTU Presidential Post-doctorate Fellowship. We thank Da Lin (College of Science, National University of Defense Technology, Changsha, Hunan, PR China) for the kind support in preparation of the manuscript. An earlier version of this paper won the grand award at the Cryptography Paper Competition (cryptography application and utilization category) organized by the South Korean Government (국가암호공모전) in 2022.

E-mail: starj1023@gmail.com (Kyungbae Jang), anubhab.baksi@eit.lth.se (Anubhab Baksi), khj1594012@gmail.com (Hyunji Kim), thdrudwn98@gmail.com (Gyeongju Song), hwajeong84@gmail.com (Hwajeong Seo), anupam@ntu.edu.sg (Anupam Chattopadhyay)

[a] Most of the work was done while the author was with the Nanyang Technological University, Singapore.
[b] Corresponding author.



Received: 2025-01-12 Accepted: 2025-03-11

# 1 Introduction

In the current situation in the world of cryptography, quantum computers are considered an upcoming major threat. This is due to the innate nature of how the quantum computers can efficiently model and solve certain problems. There is an overlap between the problems efficiently solvable by a functional quantum computer and those act as the backbones to certain cryptographic systems. Those problems are hard to solve by a classical computer, hence considered secure as of now, but the security of those systems may be threatened if quantum computers become viable in the future. It is well-known that there will be severe consequence in the field of public key cryptography [GH19], still the secret key counterpart will likely not be completely unscathed either. Depending on the structure, a secret key cipher, too, can have severe security flaw against a quantum computer[1]

One serious way for this to manifest arises from the observation that, a lot of the post-quantum ciphers use some secret key ciphers internally as a component in one way or the other (apart from the standalone usage of the secret key ciphers). This is evident from the current portfolio of the Post-Quantum Cryptography (PQC) standardization[2] being organized by the US government's National Institute of Standards and Technology (NIST). While the core components of ciphers are based on a problem presumed to be quantum-safe, due to the usage of secret key ciphers, it may be possible for the attacker to bypass the overall security claim (i.e., by exploiting only the secret key component). In other words, it may just so happen that the secret key component becomes the security bottleneck of the a post-quantum cipher (despite the core components being secure) against a potent quantum computer. Therefore, it is probably a commendable plan to consider the quantum security of the secret key ciphers, to be on the safe side.

Ultimately, the NIST call for post-quantum ciphers specified five levels of security. Each of the levels is defined on secret key ciphers (variants of AES for PKE & KEM, and variants of SHA-3 for DS). As noted in [JNRV20, Section 1], this essentially calls for concrete and precise resource estimates that would be required by an attacker with a quantum computer at her disposal.

Therefore, finding the generic quantum security level for a secret key cipher is one of the main research directions. One of the main ways an attacker with a functional quantum computer can try to mitigate the security of the secret key ciphers is by running the Grover's search algorithm [Gro96]. As a rule of thumb, it reduces the time complexity of exhaustive key search to nearly the square-root bound (with a high probability).

Our work makes a detailed and systematic attempt to estimate the search complexity on the AES family (AES-128, AES-192 and AES-256) of block ciphers [DR02], thereafter finding the complexity for the Grover's search [Gro96]. In the process, we revisit recent research works to incorporate state-of-the art advancements in various related areas (including those which are reported recently like [XZL+20, LXZZ21, LSL+19, LWF+22, ZH22, LXX+23, LPZW23, YWS+24]). Our objective lies in reducing the cost in various metrics; such as qubit count, gate count, circuit depth (Toffoli depth, full depth) and/or cost-depth trade-off (Toffoli depth $\times$ qubit count, full depth $\times$ qubit count, among other options). In the process, we carefully weigh and choose from a number of possible options.

## Contribution and Organization

We discuss in detail about the considerations/choices that are made during design separately for AES in Section 3 and architecture for combined components in Section 4.

[1] However, it is to be mentioned that the quantum computers are the nowhere near to be considered a serious generic threat against the secret key ciphers (due to impractical resource requirement) as of yet, despite the paradigm growing in leaps and bound in the past few years.

[2] https://csrc.nist.gov/projects/post-quantum-cryptography.

We observe that the implementation by by Jaques et al.[3] [JNRV20] contains some Q# programming issue, which probably results in underestimating the resources for non-linear components; although the linear components are not affected. We patch the issues (such as impossible parallelism and inconsistencies from reported quantum resources) and estimate the correct quantum gates and depth from the number of qubits in Section 5. It is to be noted that the same Q# issue was reported in the Asiacrypt'20 [ZWS+20], Asiacrypt'22 [HS22], Asiacrypt'23 [LPZW23] and Indocrypt'22 [JBK+22a] papers.

Main results are consolidated in Section 6 (cost of the implemented quantum circuits) and Section 7 (cost for running the Grover's search). Comparison of our implementations with respect to the previous works are shown in Table 5 for the three variants of AES. Table 1 shows the overall performance gain of our work with respect to previous AES implementations. It can be seen that we make significant improvement over the Asiacrypt'20 paper [ZWS+20] (such as our Toffoli depth $TD$ is reduced by over 98% for AES-128) and also the bug-fixed version of the Eurocrypt'20 paper [JNRV20]. We also include the two implementations done in [HS22] for a quick comparison. In [HS22], the qubit count and Toffoli depth of the AES quantum circuit are determined by the number of parallel S-box implementations which is denoted by $p$ — as $p$ increases, the Toffoli depth decreases, but the number of qubits increases.

We develop multiple quantum implementations of the ciphers in the AES family (AES-128, AES-192 and AES-256), and report the least Toffoli depth $TD$ and full depth $FD$ (with moderate number of qubits $M$ and quantum gates $G$) and cost-depth trade-off ($TD$-$M$; $FD$-$M$; and $FD$-$G$) implementations so-far. By increasing the number of qubits by a less quantity, we reduce the full depth greatly, so that the overall produce is significantly reduced. Moreover, this low depth is highly advantageous for reducing the cost when parallelization is required due to the depth limit in Grover's search (see Appendix 2.4). Our quantum implementations offer the best trade-offs in terms of $TD^2$-$M$ and $FD^2$-$M$ (see Table 5 for various results), which are major metrics when considering parallel search. Optimization is done at three levels, namely individual component level (S-box, MixColumn etc.), architecture level (16 S-boxes to make 1 SubBytes, 4 MixColumn to make 1 MixColumns, and so on), and finally by sharing of resources among the modules.

We present three architecture for the implementation (two of which are designed by us from scratch), each targeting for a specific optimization (see Section 3.1 for related discussion):

1. The *regular* version (originally conceived in [JNRV20], but improved by us) uses the least qubit count in our work, and reduces Toffoli circuit depth compared to the previous works for all the 3 variants. The MixColumn implementation is taken from [YWS+24], which allows zero ancilla/garbage qubit and incurs 91 CNOT gates. This architecture was proposed by [JNRV20], though it has some programming/estimation issues related to non-linear operations, such as S-box, key schedule, and out-of-place MixColumn.

2. The *shallow* version (our innovation) runs all parallel-executable parts of AES simultaneously, including reverse operations. The depth of one round only counts SubBytes + MixColumns, which is optimal. The shallow version takes the least Toffoli depth and qubit count product ($TD$-$M$ cost) and least full depth and qubit count product ($FD$-$M$) with an improved pipeline architecture. According to [ZWS+20], this is an important notion of circuit complexity. The MixColumn implementation is taken from [SF24]. Note that this version was used in [LPZW23, SF24].

3. The *shallow/low depth* version (our innovation) looks for reducing the circuit depth

[3]Henceforth we use "JNRV" or "Eurocrypt'20" to indicate this paper (used mostly interchangeably, except in Table/Figure captions where we stick to "JNRV"). The same style is followed for other papers as well.

by introducing a new low quantum depth implementation of MixColumn (based on the classical implementation from [LSL+19], but optimized for quantum). This version can be considered when the parallelization of Grover's search is unavoidable under the constraints of depth.

In addition to these three architecture, we also take a look at the implementation by Jaques et al. in Eurocrypt'20[4] [JNRV20] for AES-128/-192/-256. As noted in Section 5 and Appendix C, that implementation contained bugs[5] (this was confirmed by the authors, most notably in [JNRV19] where they presented their own take on fixing the bug). In order to fix the bug, we further introduce two new architecture (both of which are our innovation), which we call fixed depth and fixed qubit versions. On top of that, we use both the in-place MixColumn from [JNRV20] is used or the Maximov's MixColumn implementation from [Max19] is used (both were used in [JNRV20]). In order to keep the modification at minimum, we reuse the same design choices made in [JNRV20]. For this reason, we reuse the S-box implementation as in [JNRV20], which was adopted from [BP12].

Orthogonal to these architecture, we also introduce three new S-box implementations (see Table 3). We improve four S-box implementations (from [HS22, LPZW23]) that incur the Toffoli depth of 4 and 3. More information about the process of finding the S-box implementations is detailed in Appendix D.1 (full depth reduction) and Appendix D.2 (ancilla qubit reduction). Among the four S-box implementations, three are used in our regular, shallow and shallow/low depth architecture; as those incur the least full depth (namely, 61, 67, 58 and 56) compared to the other implementations that we found during our literature survey. We also propose a new out-of-place implementation of AES MixColumn that takes only 8 quantum depth with reduced quantum gates and fewer ancilla qubits (see Table 4 for the benchmark and Appendices D.1 and D.2 for the idea), this is used in our shallow/low depth implementation.

Thus, our work presents five architecture (four of which are new and the rest one is revised by us). The shallow version (which is indeed our innovation) is used by Liu et al. in Asiacrypt'23 [LPZW23] and Shi et al. in Asiacrypt'24 [SF24], see Section 2.3.2 for more discussion about this.

In this work, we present 26 distinct implementations for each variant of AES (each of the following is considered with Toffoli and AND gates):

1. Regular version:
   (a) Our own 4 Toffoli depth S-box (low qubit count), MixColumn from [YWS+24].
   (b) Our own 4 Toffoli depth S-box (low full depth), MixColumn from [YWS+24].
   (c) Our own 3 Toffoli depth S-box, MixColumn from [YWS+24].
2. Shallow version:
   (a) Our own 4 Toffoli depth S-box (low qubit count), MixColumn from [SF24].
   (b) Our own 4 Toffoli depth S-box (low full depth), MixColumn from [SF24].
   (c) Our own 3 Toffoli depth S-box, MixColumn from [SF24].
3. Shallow/low depth version:
   (a) Our own 4 Toffoli depth S-box (low qubit count), our own MixColumn.
   (b) Our own 4 Toffoli depth S-box (low full depth), our own MixColumn.

[4] Our work (the publicly available version) [JBK+22b] was mentioned by the first author of this paper during an invited talk at CHES'24: https://www.youtube.com/watch?v=eB4po9Br1YY&t=2060s.

[5] To be more precise, the bug was caused due to an inherent issue in Q# and not related to their coding. We privately contacted the authors, and the first author acknowledged our fix in a private correspondence.

**Table 1:** Performance comparison of AES quantum implementations.

| | AES | Toffoli depth ($TD$) | Qubit count ($M$) | $TD \times M$ | Full depth ($FD$) | $FD \times M$ |
|---|---|---|---|---|---|---|
| 128 | GLRS [GLRS16] | 12672 (99.68) | 984 (−67.72) | 12469248 (99.02) | 110799 (99.3) | 109026216 (97.83) |
| | LPS [LPS20] | 1880 (97.87) | 864 (−71.65) | 1624320 (92.49) | 28927 (97.32) | 24992928 (90.54) |
| | ZWSLW [ZWS+20] | 2016 (98.02) | 512 (−83.2) | 1032192 (88.19) | N/A | N/A |
| | HS [HS22] †18 | 820 (95.12) | 492 (−83.86) | 403440 (69.78) | N/A | N/A |
| | HS [HS22] †9 | 1558 (97.43) | 374 (−87.73) | 582692 (79.08) | N/A | N/A |
| | LXXZZ [LXX+23] | 476 (91.60) | 474 (−84.45) | 225624 (45.96) | N/A | N/A |
| | LPZW [LPZW23] | 40 (0) | 3688 (17.35) | 147520 (17.35) | 840 (7.62) | 3097920 (23.64) |
| | SF [SF24] | 40 (0) | 3268 (6.73) | 130720 (6.73) | N/A | N/A |
| | ❄ ✳✚ | 2394 (98.33) | 1656 (−45.67) | 3964464 (96.92) | 33320 (97.67) | 55177920 (95.71) |
| | ❄ ✲✛ | 114 (64.91) | 5088 (40.09) | 580032 (78.98) | 1612 (51.86) | 8201856 (71.16) |
| | ❁ | 40◎◈ | 3048◎ | 121920◎ | 776◎ | 2365248◎ |
| 192 | GLRS [GLRS16] | 11088 (99.57) | 1112 (−66.98) | 12329856 (98.69) | 96956 (99.04) | 107815072 (97.09) |
| | LPS [LPS20] | 1640 (97.07) | 896 (−73.40) | 1469440 (89.00) | 25556 (96.35) | 22898176 (86.29) |
| | ZWSLW [ZWS+20] | 2022 (97.63) | 640 (−81.00) | 1294080 (87.51) | N/A | N/A |
| | LXXZZ [LXX+23] | 572 (91.61) | 538 (−84.03) | 307736 (47.47) | N/A | N/A |
| | LPZW [LPZW23] | 48 (0) | 3944 (17.10) | 189312 (14.60) | 1010 (7.72) | 3983440 (21.20) |
| | ❄ ✳✚ | 2682 (98.21) | 1976 (−41.33) | 5299632 (96.95) | 37328 (97.50) | 73760128 (95.74) |
| | ❄ ✲✛ | 138 (65.22) | 5664 (40.54) | 781632 (79.32) | 1936 (51.86) | 10965504 (71.37) |
| | ❁ | 48◎◈ | 3368◎ | 161664◎ | 932◎ | 3138976◎ |
| 256 | GLRS [GLRS16] | 14976 (99.63) | 1336 (−63.77) | 20007936 (98.97) | 130929 (99.17) | 174921144 (97.71) |
| | LPS [LPS20] | 2160 (97.41) | 1232 (−66.59) | 2661120 (92.24) | 33525 (96.76) | 41302800 (90.30) |
| | ZWSLW [ZWS+20] | 2292 (97.56) | 768 (−79.18) | 1760256 (88.27) | N/A | N/A |
| | LXXZZ [LXX+23] | 646 (91.33) | 602 (−83.68) | 388892 (46.89) | N/A | N/A |
| | LPZW [LPZW23] | 56 (0) | 4456 (17.24) | 249536 (17.24) | 1176 (7.65) | 5240256 (23.57) |
| | ❄ ✳✚ | 3306 (98.31) | 2296 (−37.74) | 7590576 (97.28) | 46012 (97.64) | 105643552 (96.21) |
| | ❄ ✲✛ | 162 (65.43) | 6240 (40.90) | 1010880 (79.57) | 2264 (52.03) | 14127360 (71.65) |
| | ❁ | 56◎◈ | 3688◎ | 206528◎ | 1086◎ | 4005168◎ |

Parenthesized numbers show % (positive) improvement reported in this work.

†: Choice of $p$.

☆: Regular version (using Toffoli gate).
◎: Shallow version (using Toffoli gate).
◈: Shallow/low depth version (using Toffoli gate).
❁: S-box with Toffoli depth 4 (low qubit count).

❄: Bug-fixed JNRV [JNRV20] (using Toffoli gate).

✳: Bug-fixed depth.
✲: Bug-fixed qubit count.
✚: In-place MixColumn [JNRV20].
✛: Maximov's MixColumn [Max19].

(c) Our own 3 Toffoli depth S-box, our own MixColumn.

4. Bug-fixing of JNRV (Eurocrypt'20) [JNRV20]:

   (a) Fixed depth: S-box from [BP12], in-place MixColumn [JNRV20].
   (b) Fixed depth: S-box from [BP12], Maximov's MixColumn [Max19].
   (c) Fixed qubit count: S-box from [BP12], in-place MixColumn [JNRV20].
   (d) Fixed qubit count: S-box from [BP12], Maximov's MixColumn [Max19].

As a consequence of our analysis, the state-of-the-art bounds of the quantum security level [NIS16] is updated in Section 7. The cost for the Grover's search for each implementation can be observed from Table 10 (Table 10(a) with Toffoli and 10(b) with AND gates), and Table 11 shows a synopsis of bounds for quantum security levels. We conclude in Section 8, where we present the other AES related quantum analysis with respect to the updated security level (refer to Figure 9 for a quick view). Some additional information/discussion can be found in Appendices A (a brief description on the AES variants), B (novelty/new construction are summarized, which works as an annex to this part), C (detailed discussion on the Eurocrypt'20 [JNRV20] bug), D (more details about

our depth and qubit reduced S-box/MixColumn implementations), and E (per-round break-up of quantum resource requirement). Additionally, Section 2 covers the background/prerequisite information; in particular, Section 2.2 is about the Grover's search algorithm, and Section 2.4 is about discussion on the requirement/specification by the US government's NIST.

Our source codes are written in ProjectQ[6], which is a Python-based open-source framework for quantum computing. All our relevant source codes can be accessed online as an open-source project[7].

It may be noted that, the recent works by [LPZW23] and [SF24] took inspiration from (an earlier version of) our paper (which is available as [JBK+22b]). For instance, the authors in [LPZW23] used our shallow architecture for AES (Section 4.2 and Figure 7(b)). However, the results in (the current version of) our paper are superior to those works, owing to the choices made in the revision. For instance, we present improved S-box and MixColumn (out-of-place) implementations and achieve a greater reduction in qubit count compared to [LPZW23] by incorporating the sharing method[8] and key schedule techniques from [LPZW23, SF24]. Additionally, we introduce a last-round optimization technique (in Section 4.3), which reduces the number of output qubits required in the final round of the shallow architecture.

# 2 Background

## 2.1 Quantum Computing Basics

### Gates (Top Level and Decomposition Level)

The Hadamard gate ($H$) creates superposition, mapping $|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|1\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Applying the Hadamard gate twice results in the identity operation, i.e., $H^2 = I$. Some of the classical gates have quantum counterparts, as shown in Figure 1. The $X$ gate (Pauli-$X$) flips qubit states, where $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$ (classical NOT operation). The CNOT gate applies a conditional NOT operation, flipping the target qubit if the control qubit is $|1\rangle$ (classical XOR operation). When applied to superposition states, it can generate entanglement. The Toffoli gate (CCNOT) extends this concept, flipping the target qubit only when both control qubits are $|1\rangle$ (classical AND-controlled XOR operation). It is a key component in quantum computing and is universal for classical reversible computation. The Clifford + $T$ gate set; composed of $H$, $S$ (phase)[9], CNOT and $T$ gates; forms a universal gate set, allowing any unitary transformation. Though the Clifford gates alone are not universal, adding the $T$ gate enables universal quantum computation. Also the Clifford gates are relatively easy to implement; $T$ gates (and their $T$-depth) introduce higher resource costs, making their minimization essential in fault-tolerant quantum computing. For a more comprehensive understanding of quantum computing, interested readers are redirected to refer to other resources, such as [BJ24].

It can be stated that the Toffoli gate is decomposed in terms of the Clifford and $T$ gates, the cost and depth of such a decomposition varies based on the method [Sel13, AMM+13, HLZ+17]. Further, a Clifford gate can refer to CNOT and 1qCliff gates. Also, the $T$-depth, an important factor in error correction, is determined by $T$ gates when Toffoli gate is decomposed. After designing the quantum circuit, we need to decompose the Toffoli gates to estimate detailed quantum resources. In this paper, when estimating detailed

[6] Homepage: https://projectq.ch/.

[7] https://github.com/starj1023/AES_QC.

[8] We apply the sharing method described in [LPZW23] to our implementations (with our modification), and manage to achieve a greater reduction in qubit count than reported in their paper.

[9] This gate is used for phase adjustment. This is different from the $X$, CNOT and Toffoli gates (which respectively correspond to the classical NOT, XOR and AND operations).

quantum resources, the Toffoli gate is decomposed into (8 Clifford gates + 7 $T$ gates), $T$-depth 4, and full depth 8 following one of the methods in [AMM+13] (see Figure 2). Along with this decomposition, various approaches exist for decomposing Toffoli gates (see, for example, [CBC23] for further details).

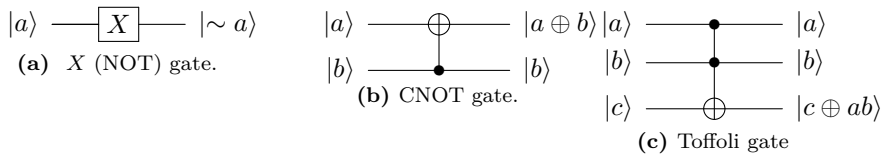
Additionally, we adopt the quantum AND gates from [JNRV20] (see Figure 10 in Appendix C). This AND gate is decomposed into (11 Clifford gates + 4 $T$ gates), $T$-depth 1, and full depth 8, and requires 1 ancilla qubit. The reverse of the AND gate which does the un-compute operation (i.e., AND$^\dagger$ gate) is designed according to the measured value of the target qubit of the AND$^\dagger$ gate. This AND$^\dagger$ gate is counted as (5 Clifford gates + 1 measurement gate) in resource estimation. Although not adopted in our work, there is another version of the AND gate [Gid18] that does not require an ancilla qubit, but has a $T$-depth of 2.

### Notations

Throughout this paper, we use the following shorthand notations: #NOT (reversible NOT gate count), #CNOT (CNOT count), #Toffoli (Toffoli count), $TD$ (Toffoli depth), $\#T$ ($T$ count), $Td$ ($T$-depth), #1qCliff as Clifford gate count, #Measure (Measurement count), $G$ (total gates), $FD$ (full depth) and $M$ (qubit count). The full depth is related to the execution time of circuits [BC17]. The importance of depth is also noted in NIST's post-quantum security requirements. In estimating the complexity of quantum attacks, NIST used only the number of gates and depth as metrics, not the number of qubits [NIS16].

### Objective

We optimize AES for quantum computers; keeping an eye on the qubit count, Toffoli depth and full depth. Further, we also consider the Toffoli depth × qubit count, the $TD$-$M$ cost, and full depth × qubit count, the $FD$-$M$ cost as metrics for trade-off. Our AES quantum circuits attain the least Toffoli and full depths, $TD$-$M$ and $FD$-$M$ costs, significantly contributing to the advancement of the state-of-the-art.




**(a)** $X$ (NOT) gate. **(b)** CNOT gate. **(c)** Toffoli gate

**Figure 1:** Basic quantum gates.

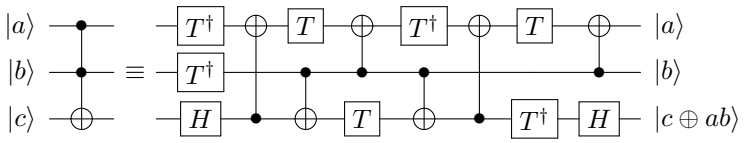



**Figure 2:** Toffoli decomposition in AMMRR.

### Methodology

We first use Toffoli gates to verify the simulation results of the implemented quantum circuit. Since ProjectQ allows classical simulation of Toffoli gates, we can verify test vectors for large-scale quantum circuits. A Toffoli gate can be simulated classically and

decomposed only when estimating resources. On the other hand, classical simulation of AND gates is not supported. Therefore, we adopt a method of verifying the implemented quantum circuit using Toffoli gates, and then replacing the top part with AND gates to estimate resources.

## 2.2 Quantum Key Search using Grover's Algorithm

For a secret-key cipher using an $k$-bit key, $2^k$ queries are required for the exhaustive key search. The Grover's search [Gro96] is a well-known quantum algorithm that recovers the key with a high probability in about $\lfloor \frac{\pi}{4}\sqrt{2^k} \rfloor$ queries. The procedure can be briefly described as follows:

1. A $k$-qubit key ($K$) is prepared in superposition $|\psi\rangle$ by applying the Hadamard gates. All states of qubits have the same amplitude:

$$|\psi\rangle = H^{\otimes k}\,|0\rangle^{\otimes k} = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) = \frac{1}{2^{k/2}}\sum_{x=0}^{2^k-1}|x\rangle$$

2. The cipher ($Enc$) is implemented as a quantum circuit and placed in Grover's oracle ($U_{f(x)}$). In the oracle, the known plaintext ($p$) is encrypted with the key in a superposition state using a comparator function ($f(x)$). As a result, the ciphertext in superposition over all key values is generated. The function $f(x)$ compares the ciphertext with the known ciphertext ($c$), where $f(x) = 1$ if $Enc_K(p) = c$, and $f(x) = 0$ otherwise. When the ciphertext matches the known ciphertext (i.e., $f(x) = 1$), the oracle returns the solution key by flipping its phase using a $Z$ gate.

$$f(x) = \begin{cases} 1 \text{ if } Enc_K(p) = c \\ 0 \text{ if } Enc_K(p) \neq c \end{cases} \tag{1}$$

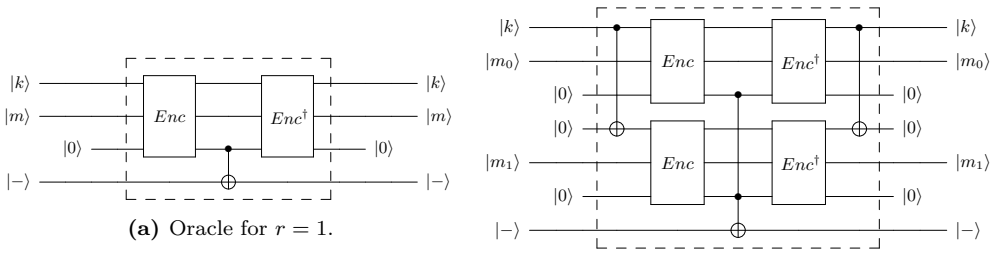
$$U_f(|\psi\rangle\,|-\rangle) = \frac{1}{2^{k/2}}\sum_{x=0}^{2^k-1}(-1)^{f(x)}\,|x\rangle\,|-\rangle \tag{2}$$

3. Lastly, the diffusion operator[10] amplifies the amplitude of the solution marked by the oracle. The diffusion operator is implemented as follows: ($H$ gate layer $\rightarrow$ $X$ gate layer $\rightarrow$ $k$-qubit controlled $Z$ gate $\rightarrow$ $X$ gate layer $\rightarrow$ $H$ gate layer). In [Per19], a simple technique was introduced by which a constant number of $X$ gates are used for the diffusion operator. If a constant number of $X$ gates are applied before the Hadamard gates in Step 1, the diffusion operator is implemented as ($H$ gate layer $\rightarrow$ $k$-qubit controlled $Z$ gate $\rightarrow$ $H$ gate layer).

The Grover's search executes Equations (1), (2) and diffusion operator in a series to sufficiently increase the amplitude of the solution and observes it at the end. For an $k$-bit key, the optimal number of iterations of the Grover's search algorithm is roughly $\lfloor \frac{\pi}{4}\sqrt{2^k} \rfloor$ [BBHT98], which is about $\sqrt{2^k}$. In the process, an exhaustive key search that requires $2^k$ queries in a classic computer is reduced to roughly $\sqrt{2^k}$ queries in a quantum computer (this works with a high probability).

In the exhaustive key search, $r = \lceil k/n \rceil$ (plaintext, ciphertext) pairs are needed to recover a unique key that is not a spurious key (see Section 7 for details). Figure 3 shows the Grover's oracle of exhaustive key search. Encryption$^\dagger$ is defined as the reverse operation of encryption, which reverts to the state before encryption.

[10] Since the diffusion operator is generally generic, it does not require any special implementation techniques.




**(a)** Oracle for $r = 1$.

**(b)** Oracle for $r = 2$.

**Figure 3:** Schematic view for key search using Grover's algorithm.

## 2.3 Related Works

Quantum analysis of symmetric key ciphers with respect to the Grover's search algorithm is one of the major research direction now-a-days. Some of the prominent examples include, but not limited to, AES [JNRV20, ZWS+20, LPS20, BNPS19, LPZW23, HS22][11], SPECK [JCK+20], PRESENT and GIFT [JSK+21a], SHA-2 and SHA-3 [ADMG+17], SM3 [SJK+21, ZLW+22], RECTANGLE and KNOT [BJS+21], DEFAULT [JBB+23], ARIA [CS20], few Korean ciphers [JSK+22, JSK+21b], SPECK and LowMC [JBK+22a], CHAM [YJBS23], ASCON [RBC23, OJBS23, OJS25].

### 2.3.1 Reflection on HS (Huang and Sun in Asiacrypt'22)

The content of this paper only revolves with AES-128, and can be summarized as:

- They improved from the Asiacrypt'20 paper's [ZWS+20] qubit count and performance.
- They chose an improved S-box implementation atop the Eurocrypt'20 implementation [JNRV20] with proposal for a quick fix for the qubit count.

We think there is some scope for improvement on the patch done by [HS22] on the Eurocrypt'20 implementation (based on the Q# code[12]). Also, the number of qubits was estimated manually in [HS22, Table 7] in the bug-fix of [JNRV20]. Not counting the bug-fix, they only proposed two versions, for AES-128 in total (Toffoli depth 3 and 4 S-box implementations, both using the MixColumn implementation from [XZL+20]), whereas we implemented eight versions.

In our paper the main contributions are, low depth implementations of AES and a thorough bug-fixing of the Eurocrypt'20 implementations. In summary, in comparison with the work by Huang and Sun in Asiacrypt'22 [HS22], we note the following points. Our approaches are mostly disjoint from that of [HS22]. Reducing the Toffoli depth is a major focus in their work, which we pursue through our shallow version. As it can be seen from Table 1, our results are indeed better than those are reported in [HS22]. Further, we cover optimized quantum implementations of AES-192 and AES-256 as well.

[11] The authors of [BNPS19] considered various quantum attacks on AES; however, as far as we can tell their work only made some estimates but did not present any concrete implementation.

[12] Previously hosted at https://github.com/AES-quantum-circuit/AES-quantum-circuit, but it seems no longer accessible.

#### 2.3.2 Reflection on LPZW (Liu, Preneel, Zhao and Wang in Asiacrypt'23) and SF (Shi and Feng in Asiacrypt'24)

The recent Asiacrypt'23 paper by Liu et al. [LPZW23] and Asiacrypt'24 paper by Shi and Feng [SF24] directly adopted the shallow version introduced in this paper.

The other contribution in [LPZW23] is as summarized as:

- They proposed a new S-box implementation (see Table 3 for its quantum benchmark).
- They reduced the number of qubits by sharing idle ancilla qubits [LPZW23, Section 4.3] in the shallow version, similar to what we reduced ancilla qubits for MixColumns in the shallow/low depth version.

The other contribution in [SF24] can be summarized as:

- They proposed a new in-place MixColumn implementation with a quantum depth of 10 [SF24, Appendix A] (see Table 4). This implementation takes the least quantum depth for an in-place implementation.
- They removed the 32 ancilla qubits originally used in the key schedule of AES-128 [SF24, Section 6.2].

In this regard, the latest results in this paper improve upon their results by incorporating the 10-depth MixColumn and key schedule implementations from [SF24], while further reducing the full depth and qubit/gate count of the S-box implementations reported in [LPZW23].

**Ancilla Qubit Reduction by LPZW.** In [LPZW23], the authors employed our shallow architecture and made an improvement by reducing the required number of qubits for two ancilla sets. The concept is the same as what we did for implementing MixColumns in the shallow/low-depth version. Ancilla qubits for implementing MixColumns in the shallow version could be saved by utilizing the idle state ancilla qubits in SubBytes (see Section 3.7). In [LPZW23], this efficient sharing technique was also applied between SubBytes and SubBytes$^\dagger$ on the shallow version. Consequently, they reduced the number of ancilla qubits by sharing the idle state ancilla qubits of SubBytes and SubBytes$^\dagger$. [LPZW23][13].

We present new S-box implementations based on [LPZW23] by reducing the full depth, quantum gates, and ancilla qubits (see Table 4 for the benchmark and Appendices D.1 and D.2 for details). Furthermore, the last-round optimization technique for the shallow and shallow/low versions is newly presented in this work (Section 4.3 and Figure 7(c)), resulting in a further reduction in the qubit count.

### 2.4 NIST Security Levels

The following security levels were defined by NIST [NIS16] to assess the post-quantum security:

① Level 1: Cipher is at least as hard to break as AES-128.

② Level 2: Cipher is at least as hard to break as SHA-256.

③ Level 3: Cipher is at least as hard to break as AES-192.

④ Level 4: Cipher is at least as hard to break as SHA-384.

⑤ Level 5: Cipher is at least as hard to break as AES-256.

[13] Note that when we applied the sharing technique in our implementations, we achieved a greater reduction in qubit count compared to [LPZW23].

It may be noted that, the security levels do not consider the key-dependent tag. Therefore, additional security levels may be considered in the future scope [OJBS23, Section 2.3].

NIST recommended that a given cipher should achieve some minimum security level to provide sufficient security in the post-quantum era. Based on the research available back then (probably the only such work was due to [GLRS16]), NIST estimated used in [NIS16] the following complexities: Level 1: $2^{170}$, Level 3: $2^{233}$, Level 5: $2^{298}$ (on a closer look, however, it seems that complexity estimated in [GLRS16] for Level 1 was close to $2^{169}$). The complexity bounds were calculated as the product of total number of decomposed gates and full depth required for the Grover's key search circuit.

With time, as more research work on the AES family has been reported, the complexity of the security levels (1, 3 and 5) has been gradually reduced. In response to this, recently, NIST has made adjustments to decrease the costs of Grover's key search on the AES family [NIS22]. Currently, NIST has defined new quantum attack costs for AES-128, 192 and 256, based on the reported costs from [JNRV20], which are $2^{157}$, $2^{221}$ and $2^{285}$; respectively. However, these costs are underestimated since there are some programming-related issues in their quantum circuit implementation. In this paper, we analyze these issues from [JNRV20] and demonstrate that the reported costs are closely achievable with our optimized AES quantum circuits.

A comprehensive synopsis of the notable works can be seen in Table 11, where we show the impact of our work on reshaping the security levels. In particular, the following new bounds are achieved (see also Table 10(b)):

- ☞ Level 1: $\mathbf{2^{156.2630}}$; with total Clifford, $T$ and measurement gates = $2^{82.2726}$; full depth = $2^{73.9899}$.
- ☞ Level 3: $\mathbf{2^{221.5801}}$; with total Clifford, $T$ and measurement gates = $2^{115.3346}$; full depth = $2^{106.2461}$.
- ☞ Level 5: $\mathbf{2^{286.0731}}$; with total Clifford, $T$ and measurement gates = $2^{147.5993}$; full depth = $2^{138.4740}$.

Along with this, NIST proposed a parameter called MAXDEPTH to impose a limit on the depth of the circuit. The bounds for MAXDEPTH are not clearly stated; rather it is speculated that the following figures can be taken as good indicators: $2^{40}$, $2^{64}$ and $2^{96}$; judging by the expected computation power of a quantum computer – in a year, or a decade, or a millennium. Keeping that in mind, one would expect the depth of the quantum circuit for the Grover's search is not higher than $2^{96}$ (i.e., the highest bound estimated for MAXDEPTH[14]). However, if it turns out that the depth restriction is not within the stipulated bound, then the following approaches can be undertaken [KHJ18]:

1. *Outer parallelization:* Restrict depth at the $\leq 2^{96}$ at the expense of lower success probability of key recovery.
2. *Inner parallelization:* Split the search space into multiple subspaces with shallow depth, where each circuit measures the secret key with a lower success probability.
3. Cost is calculated as-is without considering MAXDEPTH (see, e.g., [KHJ18, Table 2]). It is worth noting that the previous implementations like [ZWS+20, LPS20, ASAM18, HS22] also did not appear to consider the MAXDEPTH limit.

[14] As the Grover's search makes the circuit depth greater than $2^{k/2}$ for $k$-bit key (the quantum depth for the cipher implementation $\times \lfloor \frac{\pi}{4} 2^{k/2} \rfloor$ required for Grover's iteration), the quantum depth is trivially greater two smaller MAXDEPTH values for AES variants; hence only $2^{96}$ is considered within the context of this paper.

The outer and inner parallelization methods lower the probability of measuring a solution by reducing the number of iterations for the Grover oracle. Outer parallelization halts the Grover iterations at the depth limit, leading to the measurement of suboptimal solutions with lower probabilities. Inner parallelization reduces the number of Grover iterations by reducing the search space, which also lower the probablility of discovering a solution. However, parallelizing the Grover's search is highly inefficient due to the poor performance resulting from the analysis in [Zal99], which indicates that only a $\sqrt{S}$ depth reduction can be achieved with $S$ instances (operating in parallel) of the Grover oracle. Thus, the optimal method is to perform as many iterations of the Grover oracle as possible within a limited depth. According to the analysis in [JNRV20, Section 3.4], to minimize the $TD$-$M$ (Toffoli depth and qubit count product,) and $FD$-$M$ (full depth and qubit count product,) costs under the parallelization of Grover's search, it is necessary to minimize the $TD^2$-$M$ and $FD^2$-$M$ costs. This is because reducing the depth ($TD/FD$) by $\sqrt{S}$ requires $S$ instances of the Grover oracle, leading to a more significant increase in the total number of qubits ($M$) required for parallelization.

Suppose the total depth required for the Grover's search exceeds MAXDEPTH (i.e., $FD >$ MAXDEPTH). To address this depth limit, parallelization is required to reduce $FD$ to match MAXDEPTH. It can be commented that, $FD$ should be reduced by a factor of $\frac{FD}{\text{MAXDEPTH}}$ (which is $\sqrt{S}$). To achieve this reduction, $\frac{FD^2}{\text{MAXDEPTH}^2}$ (which is $S$) Grover instances need to operate in parallel. Consequently, $FD$ is reduced by a factor of $\sqrt{S}$, leading to a reduction to MAXDEPTH. On the other hand, $M$ is increased by a factor of $S$, resulting in $\frac{FD^2}{\text{MAXDEPTH}^2} \cdot M$. Finally, for the parallelization of Grover's search, the $FD$-$M$ cost transforms into $\frac{FD^2 \cdot M}{\text{MAXDEPTH}}$ (i.e., $\frac{FD}{\sqrt{S}} \times (M \times S)$). That is, the focus shifts to the challenge of minimizing the $FD^2$-$M$ metric. In the same context, when considering $TD$-$M$ cost, our goal should be to minimize the $TD^2$-$M$ metric. Therefore, when parallelization due to depth limitation is inevitable, the primary objective should be to minimize the depth.

In terms of the gate count $G$, $S$ Grover instances are executed in paralllel, and the gate count of each instance is decreased by a factor of $\sqrt{S}$. Thus, by the formula $S \cdot \frac{G}{\sqrt{S}}$, the total gate count should be $\frac{G \cdot FD}{\text{MAXDEPTH}}$. This formula mirrors NIST's method for estimating the quantum attack cost for AES [NIS16, NIS22]. This is why we adopt $G \cdot FD$ as a primary metric for estimating attack cost and our paper offers the best performance in terms of this metric (Level 1: $2^{156.2630}$, Level 2: $2^{221.5801}$, and Level 5: $2^{286.0731}$).

As of now, we remark that the depths of our AES quantum circuits are the least when compared to other quantum circuits available in the literature [ZWS+20, LPS20, ASAM18, HS22]. Table 2 displays a quick view where the related works (namely, GLRS [GLRS16] and LPS [LPS20]) are compared with respect to our implementations in terms of full depth. Note that, only AES-128 satisfies the MAXDEPTH criterion (i.e., $\leq 2^{96}$).

One may further note that the depth of quantum attack on AES-128 (i.e., Level 1) is within the permitted MAXDEPTH limit ($2^{73.9207}$ using the AND gate; the same using the Toffoli gate is $2^{73.9899}$). However, the same cannot be stated for AES-192 and -256, since the full depth figures are respectively $2^{106.1763}$ and $2^{138.4082}$ (using the AND gate; the same using the Toffoli gate are $2^{106.2461}$ and $2^{138.4761}$, respectively). In this work, we adopt the 3rd approach for the sake of brevity and report the cost with considering the MAXDEPTH limit.

In future, we can identify the optimal parallelization strategy that strikes a balance between adjusted cost – success probability trade-offs (Section 7). As described, the circuit depth metrics are the primary factor determining performance in general.

**Table 2:** Summary of AES quantum bounds with respect to MAXDEPTH.

| AES | GLRS [GLRS16] | LPS [LPS20] | This work | | | | | | | MAXDEPTH |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | ☆ | ◎ | ◈ | ❄ (Toffoli) | | ❄ (AND) | | ($\leq 2^{96}$) |
| 128 | $2^{81.2141}$ | $2^{79.4751}$ | ❁: $2^{74.6471}$<br>✿: $2^{74.3426}$<br>❀: $2^{74.4573}$ | ❁: $2^{74.1362}$<br>✿: $2^{74.0300}$<br>❀: $2^{73.9627}$ | ❁: $2^{74.0704}$<br>✿: $2^{73.9899}$<br>❀: $\mathbf{2^{73.9207}}$ | ✳✚: $2^{79.6576}$<br>✳+: $2^{79.7011}$ | ✲✚: $2^{75.5008}$<br>✲+: $2^{75.3060}$ | ✳✚: $2^{79.0636}$<br>✳+: $2^{79.1035}$ | ✲✚: $2^{75.4543}$<br>✲+: $2^{74.7847}$ | ✓ |
| 192 | $2^{113.4114}$ | $2^{111.2987}$ | ❁: $2^{106.8906}$<br>✿: $2^{106.5811}$<br>❀: $2^{106.6987}$ | ❁: $2^{106.3907}$<br>✿: $2^{106.2857}$<br>❀: $2^{106.2203}$ | ❁: $2^{106.3254}$<br>✿: $2^{106.2461}$<br>❀: $\mathbf{2^{106.1763}}$ | ✳✚: $2^{111.8395}$<br>✳+: $2^{111.8673}$ | ✲✚: $2^{107.7756}$<br>✲+: $2^{107.5703}$ | ✳✚: $2^{111.2271}$<br>✳+: $2^{111.2702}$ | ✲✚: $2^{107.7258}$<br>✲+: $2^{107.0464}$ | ✗ |
| 256 | $2^{145.6508}$ | $2^{143.6871}$ | ❁: $2^{139.1177}$<br>✿: $2^{138.8049}$<br>❀: $2^{138.9252}$ | ❁: $2^{138.6201}$<br>✿: $2^{138.5150}$<br>❀: $2^{138.4510}$ | ❁: $2^{138.5539}$<br>✿: $2^{138.4740}$<br>❀: $\mathbf{2^{138.4082}}$ | ✳✚: $2^{144.1412}$<br>✳+: $2^{144.1679}$ | ✲✚: $2^{140.0098}$<br>✲+: $2^{139.7964}$ | ✳✚: $2^{143.5283}$<br>✳+: $2^{143.5701}$ | ✲✚: $2^{139.9553}$<br>✲+: $2^{139.2680}$ | ✗ |

☆: Regular version (using AND gate).
◎: Shallow version (using AND gate).
◈: Shallow/low depth version (using AND gate).
❁: S-box with Toffoli depth 4 (low qubit count).
✿: S-box with Toffoli depth 4 (low full depth).
❀: S-box with Toffoli depth 3.
❄: Bug-fixed JNRV [JNRV20] (using S-box from [BP12].)
✳: Bug-fixed depth.
✲: Bug-fixed qubit count.
✚: In-place MixColumn [JNRV20].
+: Maximov's MixColumn [Max19].

# 3 Components for Quantum Circuits for AES

## 3.1 Regular, Shallow and Shallow/Low Depth Versions

Our quantum circuit implementations are divided into regular and shallow versions. The regular version offers high parallelism while taking into account the trade-off of depth-qubit. The regular version has the best performance for qubit count. The shallow version also considers the trade-off of qubit-depth, but further reduces the depth (especially Toffoli depth) by burdening the use of ancilla qubits. The shallow version has the best performance in terms of Toffoli depth, $TD$-$M$ cost (which is the Toffoli depth - qubit count product), $FD$-$M$ cost (which is the full depth - qubit count product), $TD^2$-$M$ cost (which is the Toffoli depth$^2$ - qubit count product), and $FD^2$-$M$ cost (which is the full depth$^2$ - qubit count product). The shallow/low depth version seems to achieve the lowest depth for quantum circuit implementation. The shallow/low version is considered when estimating the quantum attack cost for Grover's key search, and parallelization of the Grover's search is essential due to the depth limit. The shallow/low version has the best performance for full depth ($FD$), $FD$-$G$ cost (which is the metric used by NIST to estimate quantum attack cost).

The regular version focuses on the parallelism within the round. To optimize the depth of the components through parallelization, we utilize additional ancilla sets (except in-place MixColumns). It reduces the depth while conserving the number of qubits by allowing for many ancilla qubits and reusing them in the next round through reverse operation (Figure 4(b)). In this version, while the current round awaits, the previous round goes through the reverse operation. In other words, the next round cannot start until the reverse operation on the current round is complete. Parallelization of modules within a round, such as SubBytes, key schedule, and MixColumns, is achieved, but parallelization between rounds is not attained.

On the other hand, the shallow version manages to parallelize the processing for all the rounds. For this, we present a new idea for pipelining of operation (Figure 7(b)), which reduces the Toffoli depth and full depth from the previous works (as in Figure 7(a)). In this version, the reverse operation of the previous round is run simultaneously with the current round, alternating between the even and the odd rounds (for instance, while the even rounds are at compute operation, the odd rounds are at the un-compute operation). This version uses more qubits, but offers lower depths, because all the rounds of the parallelizable parts of the cipher run simultaneously. As a consequence, it achieves lower circuit depth, as in this case the bottleneck of the depth is that of the SubBytes plus MixColumns in every round (except for the last round where MixColumns depth is

not counted).

That said, one may notice that the depth can be reduced if a different implementation of MixColumn is opted for, though the Toffoli depth is unchanged. In our shallow version, we choose the MixColumn implementation from [SF24], as it offers in-place implementation. As pointed out in Table 4, it is possible to lower the depth at the expense of more qubits, if our MixColumn implementation is chosen instead. Thus, everything else being inherited from the shallow version, the shallow/low depth version achieves lower full depth. One note-worthy point is that the number of ancilla qubits required for the MixColumn implementation in the shallow/low version is irrelevant. This is due to the utilization of idle ancilla qubits after their use in SubBytes (related explanation is given in Section 3.7).

Although minimizing the depth for the Grover's search is more effective, most papers implementing quantum circuits for AES focus on reducing the usage of qubits [GLRS16, LPS20, ZWS+20, ASAM18, WWL22, HS22]. The serial circuit structure (which aims at reducing the number of qubits) significantly increases the circuit depth. As stated already, our quantum circuits for AES attempt to find the lowest depth while maintaining the best possible balance between the number of qubits required with its relation to increment of the circuit depth. Thanks to the careful choices, our AES quantum circuits provide arguably the best trade-offs in terms of $TD$-$M$ cost by varying $TD$ and $M$, where recall that $TD$ is the Toffoli depth and $M$ is the number of qubits. This product is taken as the trade-off indicator for the quantum circuit in [ZWS+20]. Further, as stated in the NIST document [NIS22], Grover's algorithm requires a long-running serial computation, which is difficult to implement in practice. That is, in real-world attack scenarios, it is unavoidable to execute multiple smaller instances of the algorithm in parallel. Indeed, in this scenario (parallel search), the cost of $TD$-$M$ and is redefined as $TD^2$-$M$ (see Appendix 2.4 for the rationale). This product is taken as the trade-off indicator for the quantum circuit in [JNRV20]. From Table 5, it can be observed that our $TD^2$-$M$ is the lowest.

We also use the depth - qubits count product in estimating the $FD$-$M$ cost. This metric is also realistic and is used primarily for evaluation. Our AES quantum circuit achieves the best trade-offs in terms of $FD$-$M$ cost. In the same context as the $TD$-$M$ cost, we achieve the best trade-offs in terms of the $FD^2$-$M$ cost.

## 3.2 Implementation of S-box (SubByte)

The resource estimation in quantum is performed in ProjectQ and according to the method of [AMM+13], one Toffoli gate is decomposed into (8 Clifford gates + 7 $T$ gates) and $T$-depth of 4, and full depth of 8. For the cost comparison and implementation details in Section 3, we use only the Toffoli gate. If we adopt the AND gate instead of the Toffoli gate, an ancilla qubit is required, but it can be saved depending on the overall structure. Thus, the cost of the AND gate version is estimated in Section 6 by replacing the Toffoli gates at the top of the AES quantum circuits implemented with AND gates. Table 3 shows the resources required for the implementations found so far.

The S-box, used in the round function as well as the key schedule (in SubWord, that part is described in Section 3.4), typically occupies the most resources in the quantum circuit. Previous authors like [LPS20, ZWS+20] used the results from [BP10, BP12]. If the Boyar-Peralta's S-box implementations [BP10, BP12], which were originally designed for efficient hardware implementation, are ported to quantum in the naïve way, then the version of [BP12] requires more ancilla qubits (120 ancilla qubits) than the quantum version of [BP10] (107 ancilla qubits), but attains lower depth. JNRV [JNRV20] adopted the implementation of the S-box of [BP12] to the corresponding quantum circuit as-is. The S-box implementation of [BP10] was adopted and improved for quantum implementation in [ZLD+19]. In [LPS20, ZWS+20], Langenberg et al. took the first S-box implementation by Boyar-Peralta from [BP10] and presented the S-box quantum circuit, after converting it to use less qubits. In [JBKK24], the authors found new AES S-box circuits using a

heuristic search framework based on the circuits in [BP10, BP12]. We port [JBKK24, Listing 17] to quantum and estimate the required quantum resources (see Table 3). This is among the best S-box circuit in terms of depth in their work. Huang and Sun reported an improved quantum implementation for the S-box of [JNRV20] in their Asiacrypt'22 paper [HS22]. They presented two quantum implementations of reduced Toffoli depth with new observations of the classical implementation of the AES S-box as given in [BP12]. The first version reduced the Toffoli depth without increasing the number of qubits, while the second version used more qubits to further reduce the Toffoli depth. Finally, the Asiacrypt'23 paper [LPZW23] applied a new method to the S-box from the Asiacrypt'22 paper [HS22].

The S-box implementation in [GLRS16] is based on finite field inversion (using the Itoh–Tsujii's algorithm). In [HB24], a finite field inversion-based S-box implementation was proposed, in which all operations are decomposed into bit-level operations. We port [HB24, Algorithm 6] to quantum and report the required quantum resources in Table 3.

Dansarie [Dan17, Dan21] proposed another method in the context of classical implementation, although it did not seem to receive much attention from the community. It was rather generic, as it can find implementation of an arbitrary 8-bit S-box (i.e., unlike [BP10, BP12], this is not specific to the AES S-box), with respect to a user-provided set of logic gates. With the publicly available source code[15], we generate 5 implementations, in which the number of lines in the C source files is in the ballpark of 400 (it contain AND, OR and NOT gates; and sometime one line contains more than 1 gate). These are not used in this work due to high quantum cost (see Table 3 for the benchmarks).

By prudently using LIGHTER-R [DBSC19][16], the authors of [LGQW23] proposed a new S-box implementation with low qubit count. Subsequently, the authors of [LXX+23] presented another low qubit S-box implementation. Since detailed quantum benchmarks with Clifford + $T$ gates are not provided in [LGQW23, LXX+23][17], only Toffoli depth and qubit count are reported in Table 3.

Two new S-box implementations were presented in [LPZW23]. This, together with their 16 quantum depth MixColumn, allowed the authors of [LPZW23] to reduce the overall cost (compared to an earlier version of this paper), despite using the same shallow architecture as ours.

To push the envelope, we propose new S-box implementations by starting with the implementations reported in [LPZW23]. We manage to reduce the full depth ($82 \rightsquigarrow 67$ and $69 \rightsquigarrow 58$), the CNOT gate count ($400 \rightsquigarrow 394$ and $372 \rightsquigarrow 366$) and the qubit count ($90 \rightsquigarrow 84$) from [LPZW23] (see Table 3 for the comparison). The theory behind our improvement can be found in Appendix D.

We also found two reversible implementations of AES S-box from [LYLL22, Appendices C and D]. However, those were given in raster graphics format and quite difficult to read[18].

One may note that the AES implementations in [ZWS+20] and [SF24] used the S-box$^{-1}$ (from [BP12] and [LPZW23], respectively) in designing a architecture that reduced the number of qubits. However, in our case, we do not use the inverse S-box for the regular, shallow and shallow/low depth versions; this is explained in Section 4.2.

### 3.3 Implementation of SubBytes

Considering the trade-off between the circuit depth and the number of qubits required for an S-box implementation, we treat two cases. The first case is when the ancilla qubits have to be allocated per SubBytes, which is indeed sensitive to the number of qubits. The

[15] https://github.com/dansarie/sboxgates.

[16] It may be possible to use DORCIS [CBC23, BCC+24] (instead of LIGHTER-R) to reduce the cost further.

[17] Given the way it was written, so far we are unable to format it for decomposed benchmark. We shall update the results here should we receive any communication from the authors regarding this.

[18] We contacted the authors for an easily readable format — currently we are awaiting their response.

**Table 3:** Comparison of quantum implementations of AES S-box.

| Method | #CNOT | #1qCliff | $\#T$ | $TD$ | $M$ | Full depth |
|---|---|---|---|---|---|---|
| S-box [GLRS16] | 1818 | 124 | 1792 | 88 | 40 | 951 |
| S-box [BP10] | 358 | 68 | 224 | 8 | 123 | 104 |
| S-box [BP12] ✣ | 392 | 72 | 238 | 6 | 136 | 85 |
| S-box [LPS20] | 628 | 98 | 367 | 40 | 32 | 514 |
| S-box [ZWS+20] | 437 | 72 | 245 | 55 | 22 | 339 |
| S-box [Dan17, Dan21] 391 lines | 1470 | 670 | 1218 | 66 | 399 | 640 |
| S-box [Dan17, Dan21] 406 lines | 1507 | 548 | 1245 | 74 | 414 | 709 |
| S-box [Dan17, Dan21] 413 lines | 1484 | 561 | 1169 | 62 | 421 | 591 |
| S-box [Dan17, Dan21] 409 lines | 1483 | 574 | 1190 | 74 | 416 | 693 |
| S-box [Dan17, Dan21] 400 lines | 2244 | 1006 | 2254 | 111 | 408 | 998 |
| S-box [JBKK24] | 472 | 72 | 238 | 4 | 209 | 69 |
| S-box [HB24] | 446 | 88 | 294 | 8 | 147 | 99 |
| S-box [HS22] | 418 | 72 | 238 | 4 | 136 | 72 |
| S-box [HS22] | 824 | 160 | 546 | 3 | 198 | 69 |
| S-box [LGQW23] | N/A | N/A | N/A | 32 | 20 | N/A |
| S-box [LXX+23] | N/A | N/A | N/A | 24 | 21 | N/A |
| S-box [LXX+23] | N/A | N/A | N/A | 22 | 22 | N/A |
| S-box [LPZW23] | 400 | 72 | 238 | 4 | 76 | 82 |
| S-box [LPZW23] | 372 | 72 | 238 | 4 | 90 | 69 |
| S-box (ours) | 418 | 72 | 238 | 4 | 136 | 61 |
| S-box (ours) ❁ | 394 | 72 | 238 | 4 | 76 | 67 |
| S-box (ours) ✿ | 366 | 72 | 238 | 4 | 84 | 58 |
| S-box (ours) ❀ | 781 | 160 | 546 | 3 | 152 | 56 |

✣: Reused in this work to fix [JNRV20].

❁: Used in this work (Toffoli depth 4 and low qubit count).

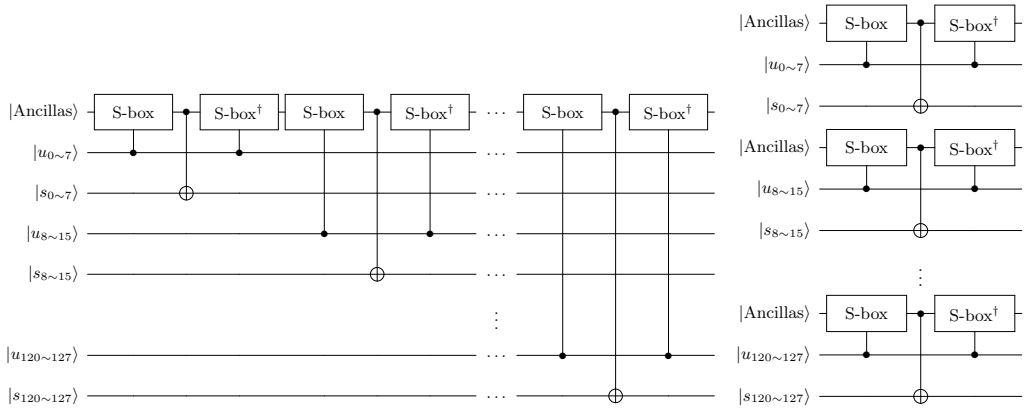
✿: Used in this work (Toffoli depth 4 and low full depth).

❀: Used in this work (Toffoli depth 3).

second case is when the initially allocated ancilla qubits can be reused. Here, there is no need to allocate additional ancilla qubits for the next SubBytes. Therefore, the number of ancilla qubits is maintained, but the depth and number of gates increase due to the reverse operations needed to reuse the ancilla qubits. We choose the second case for our SubBytes implementation, since we expect that the benefit of reducing the number of qubits outweighs the cost saved by not performing additional reverse operations. In this case, only the initial allocation is burdened because the ancilla qubits are reused. In this way, due to the relatively high qubit count but low depth, we increase the initial burden and use fast (low depth) S-boxes for free (without ancilla qubits) until the end.

After we decide upon the implementation of the S-box (SubByte, Section 3.2), this can be used to implement 16 S-boxes (SubBytes). Regarding the implementation of SubBytes in AES, Figure 4(a) shows the method that uses the fewest qubits. In this case, all S-boxes are executed sequentially, which causes a significant increase in depth, as shown in Figure 4(a). On the other hand, we reduce the depth by allocating more ancilla sets initially. The notation S-box$^{\dagger}$ is described in Appendix A.4.

In one round, 16 S-boxes in SubBytes and 4 S-boxes in key schedule, a total of 20 S-boxes are operated, simultaneously. Therefore, we allocate $20 \times 60$ ancilla qubits for S-boxes with Toffoli depth 4 and low qubit count, $20 \times 68$ ancilla qubits for S-boxes with Toffoli depth 4 and low full depth, and $20 \times 136$ ancilla qubits for S-boxes with Toffoli depth 3 to run all S-boxes simultaneously. Figure 4(b) shows 16 S-boxes operation in parallel using multiple ancilla sets. After S-box operations, ancilla qubits are not in a clean




**(a)** Using one ancilla set. **(b)** Using multiple ancilla sets.

**Figure 4:** SubBytes implementation in quantum.

state (i.e., not all ancilla is $|0\rangle$). Initialization with 16 S-boxes$^\dagger$ operation (i.e., returning to $|0\rangle$) is performed in parallel for the next round. Thanks to this, we can reuse the initialized ancilla qubits in the next round of SubBytes. Of course, these reverse operations save qubits, but increase depth. However, if we allocate ancilla qubits each time by skipping reverse operations, it is an abuse of qubits. We consider these trade-offs carefully and the shallow version offsets this depth overhead from reverse operations (this will be described in Section 4.2).

## 3.4 Implementation of Key Schedule (Excluding SubByte)

In the key schedule of AES, SubWord operates on rearranged 32-qubit. Out of the $20 \times (60, 68 \text{ or } 136)$ ancilla qubits previously decided to use (refer to Section 3.3), $4 \times (60, 68 \text{ or } 136)$ ancilla qubits are used to simultaneously operate S-boxes for 32-qubit in the key schedule ($16 \times 60$, $16 \times 68$ or $16 \times 136$ ancilla qubits are used in SubBytes of round). For rearranging the 32 qubits, quantum resources are not used by the logical swap operation as it merely changes the index of the qubits.

In SubBytes, the outputs of S-boxes are stored in new qubits. On the other hand, in the key schedule, no additional qubits are allocated because the outputs of the S-boxes are XORed (using CNOT gates) inside the key. Since SubWord for 32-qubit operates in parallel with SubBytes of round, there is no depth overhead in our AES quantum circuit implementation. This approach was already used in [JNRV20]. XORing the 8-bit round constant (RC) is implemented by performing $X$ gates to $|k_{120\sim127}\rangle$ according to the positions where the bit value of the round constant is 1. Lastly, the CNOT gates inside the key are performed. Figure 5[19] shows the quantum circuit for the AES-128 key schedule (see Appendix A.4 for the description of Rotation$^\dagger$ and SubWord$^\dagger$).

Similar to the round function, the S-boxes in the key schedule also operate in parallel. This means that, the depth is the same as operating the 8-bit S-box once. The implementation details about the S-box is discussed Section 3.2, so it is omitted here for brevity.

Similar to other authors, we adopt the on-the-fly approach where we generate all round keys immediately before use. This contrasts the typical implementation in a classical circuit where the round keys are pre-computed. Our AES quantum implementation executes

[19] In [SF24, Section 6.2], the authors presented the key schedule without ancilla qubits by using the reverse operation, and we adopt it in our implementations.

the key schedule simultaneously with SubBytes in the round function. For the reverse computation (denoted using the † notation), one may refer to Appendix A.4.

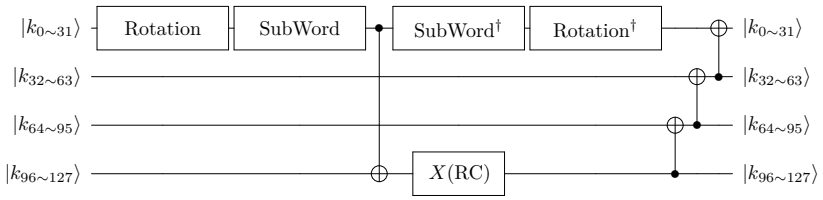



**Figure 5:** AES-128 key schedule in quantum.

In most implementations of AES quantum circuits, the full depth and Toffoli depth of AES-128 are higher [GLRS16, LPS20, JNRV20] or similar [ZWS+20] to those of AES-192. Although AES-128 has fewer rounds, this is due to differences in key schedule. AES-128 requires 16 S-boxes for SubBytes and 4 S-boxes for key schedule in every round. On the other hand, some rounds of AES-192 require only 16 S-boxes for SubBytes, since SubWord in the key schedule are not required. As a result, AES-128 has a higher depth than AES-192.

Another interpretation of this is that there is a depth overhead for key schedule in implementing AES quantum circuits. However, in our AES quantum circuits there is no depth overhead for key schedule (there is overhead for gates and ancilla qubits). Our AES quantum circuit runs the key schedule in complete parallel, so we achieve the same depth as if the key schedule was omitted. As a result, unlike other implementations, the quantum resources required for our AES-128, -192 and -256 quantum circuits are strictly dependent on the number of rounds.

## 3.5 Implementation of AddRoundKey and ShiftRows

The AddRoundKey operation, which XORs a 128-qubit round key, can be implemented simply by using 128 CNOT gates. In the case of ShiftRows, it can be implemented using swap gates, but quantum resources are not used through logical swap that changes the index of qubits. Since no special implementation technique is applied for AddRoundKey and ShiftRows, this approach is used in quantum circuit implementation.

## 3.6 Implementation of MixColumn

In the literature, both the in-place (i.e., of the form $a \leftarrow a \oplus b$ or $b \leftarrow a \oplus b$, and thus require 32 qubits) and out-of-place (i.e., of the form $c \leftarrow a \oplus b$, and requires more than 32 qubits) implementations were used.

As noted in [RBC23], the Gauss-Jordan elimination and the PLU factorization are two legacy algorithms that return in-place implementations. These papers used the PLU factorization in some form: [GLRS16, ZWS+20, ASAM18, JNRV20]. From what we can tell, the Gauss-Jordan elimination was never used as such (although it was used in [XZL+20] as the fallback algorithm for the A$^\star$ search).

Implementations like that of [Max19, LXZZ21, LWF+22], do not work in-place due to the require usage of temporary variables (i.e., ancilla/garbage qubits) and/or depth (due to cleaning up garbage qubits).

**Table 4:** Comparison of quantum implementations of AES MixColumn.

| Method | #CNOT | #qubit ($M$) | Depth |
|---|---|---|---|
| MixColumn (Naïve) GF($2^8$) (Encoding [BP10]) | 184 | 64 | 25 |
| MixColumn (Naïve) GF(2) (Encoding [XZL+20]) | | | 52 |
| MixColumn [GLRS16, ZWS+20]† | 277 | 32 | 39 |
| MixColumn [KLSW17] | 194 | 129 | 15 |
| MixColumn [ASAM18]† | 275 | 32 | 200 |
| MixColumn [Max19]✛ | 188 | 126 | 13 |
| MixColumn [JNRV20]✚† | 277 | 32 | 111 |
| MixColumn [TP19] | 188 | 126 | 17 |
| MixColumn [XZL+20]† | 92 | 32 | 30 |
| MixColumn [ZH22]† | | | 28 |
| MixColumn [LPZW23]† | 98 | 32 | 16 |
| MixColumn [LXZZ21]✳ | 182 | 123 | 16 |
| MixColumn [LWF+22] | 206 | 135 | 13 |
| MixColumn [LZW23] | 204 | 134 | 13 |
| MixColumn [BCC+24]† | 97 | 32 | 17 |
| MixColumn [PD24] | 159 | 76 | 18 |
| MixColumn [LWS+22]† ASIC1, EGT2 | 95 | 32 | 40 |
| MixColumn [LWS+22]† ASIC1, EGT3 | 95 | | 40 |
| MixColumn [LWS+22]† ASIC2, EGT2 | 96 | | 40 |
| MixColumn [LWS+22]† ASIC2, EGT3 | 96 | | 32 |
| MixColumn [BDK+21] 95 $b_1$ | 190 | 127 | 14 |
| MixColumn [BDK+21] 59 $b_2$, 4 depth | 165 | 91 | 15 |
| MixColumn [BDK+21] 59 $b_2$, 5 depth | 162 | 91 | 18 |
| MixColumn [BDK+21] 240.95 (ASIC4) $b_3$ | 290 | 108 | 19 |
| MixColumn [BFI21] 103 XOR, 3 depth | 206 | 135 | 11 |
| MixColumn [BFI21] 95 XOR, 6 depth | 190 | 127 | 15 |
| MixColumn [YWS+24]† 91 CNOT (Classical)☆ ✳ | 91 | 32 | 35 |
| MixColumn [YWS+24]† 98 CNOT (Quantum) | 98 | 32 | 13 |
| MixColumn [SFX23] | 198 | 131 | 14 |
| MixColumn [LSL+19] | 210 | 137 | 11 |
| MixColumn [SF24]◎ † | 131 | 32 | 10 |
| MixColumn (Ours)◈ | 169 | 96 | 8 |

✛✚: Reused in this work to fix [JNRV20] ❄.
☆: Used in regular version.
◎: Used in shallow version.
◈: Used in shallow/low depth version.
✳: Least XOR count in classical circuit.
†: In-place implementation.

In [XZL+20], Xiang et al. presented an implementation using 92 in-place CNOT gates (with classical depth 6). A different implementation costing 92 out-of-place XOR gates (with classical depth 6) was reported earlier by [Max19]. These two were the least cost implementations in classical circuits, until another implementation with 91 out-of-place XOR gates (with classical depth 7) was found by [LXZZ21]. Recently, another paper [YWS+24] tied with 91 in-place CNOT gates. Thus, the research by [YWS+24] improved greatly over all previous in-place implementations.

Another implementation of AES MixColumn was found thanks to [LWF+22], which managed to reduce the classical depth to 3 with 103 out-of-place XOR gates (cf. 103 out-of-place XOR gates incurring 3 classical depth from [BFI21]). This tied with another implementation from [LSL+19], albeit the latter required 105 out-of-place XOR gates. On a different direction, the implementation by [LSL+19] appears to have lower depth than that of [LWF+22] when converted to quantum circuits (despite both having same classical depth).

The authors of [BFI21] presented two implementations, namely (103 XOR, 3 classical

depth) and (95 XOR, 6 classical depth). If taken as-is, the (103 XOR, 3 classical depth) implementation yields 206 CNOT gates, 135 qubits, with 11 quantum depth when ported. Thus, it is in theory possible to slightly improve our shallow/low depth version by switching to this implementation. Further, if the (95 XOR, 6 classical depth) implementation is ported as-is; it incurs 190 CNOT gates with 127 qubits with depth 15; however we could not verify the results (probably due to an encoding issue).

We port the classical implementation of MixColumn in [YWS+24] (91 CNOT) to quantum and use it in our regular version. On the other hand, for the shallow version, we opt for an in-place MixColumn with a quantum depth of 10, presented in [SF24]. The reasoning behind this choice is explained in Sections 4.1 and 4.2.

Apart from the specialized MixColumn implementations just narrated, it is perhaps worth noting that the naïve quantum implementation (i.e., directly porting the binary matrix to quantum circuit, see [RBC23]) was seemingly never studied. With our implementations, one as a $4 \times 4$ matrix over GF($2^8$), and the other as a $32 \times 32$ binary matrix; we notice from Table 4, the CNOT count being the same, that the depth varies — this is likely due to the compiler's inability to optimize for depth[20].

We tested with the SMT/MILP model from [BKD21], whence we inferred that the minimum CNOT count for an in-place implementation of the AES MixColumn matrix is at minimum 17. We also ran the publicly available source-code from [BDK+21], and found a new implementation which took 240.95 GE in ASIC4 (STM 130nm) with the $b_3$ (i.e., out-of-place {XOR, XOR3, XOR4} gates) logic library and improved from the result of 243.00 GE of [LWS+22, Table 3]. The work by [PD24] follows-up from [BDK+21] and proposes a new implementation. Also note that the implementations from [LWS+22] are proposed from the point-of-view of multi-input XOR gates, but those actually work in-place.

### Consideration for Quantum Depth

One may note from Table 4 that the depth for quantum circuit corresponding to the implementation by [XZL+20] is 30, whereas the same for the classical circuit is 6. Although this implementation operates in-place, it still reuses one variable multiple times. In other words, the same variable appears multiple times in the right hand side. For example, one may check that $x_{31}$ appears more than once: $x_{16} \leftarrow x_{16} \oplus x_{31}$ (Line 15), $x_4 \leftarrow x_4 \oplus x_{31}$ (Line 29), $x_0 \leftarrow x_0 \oplus x_{31}$ (Line 56), and so on. This does not account for extra depth in a classical circuit (as multiple fan-outs are allowed). However, in a quantum circuit where there is exactly one fan-out, this situation causes increase of quantum depth.

The work by [ZH22] looked into this reduction of quantum depth problem, and this is the first work to directly address the issue of quantum depth optimization to the best of our finding. In particular, the authors in [ZH22] took the (92 CNOT, 30 quantum depth) implementation from [XZL+20], and managed to reduce the quantum depth to 28 (by keeping the same CNOT count) through shuffling the order of the CNOT gates. Later, a 16 quantum depth in-place implementation was found by [LPZW23] that required 98 CNOT gates; which was improved by the (97 CNOT, 17 quantum depth) implementation

[20] In essence, we do not provide the binary matrix form, rather give instruction to directly implement the GF($2^8$) matrix. Interpreting a $4 \times 4$ matrix over GF($2^8$) results in an implementation with units of 8 qubits, where the inputs ($x_{0\sim31}$) are mapped to outputs ($y_{0\sim7}, y_{8\sim15}, y_{16\sim23}, y_{24\sim31}$) respectively. On the other hand, interpreting a $32 \times 32$ matrix over GF(2) results in an implementation with units of 1 qubit, where the inputs ($x_{0\sim31}$) are mapped to outputs ($y_{0\sim31}$) respectively. Though the explicit form of the binary matrix is not specified; the resulting binary matrix follows in the GF(2) version the same encoding as [XZL+20], while the GF($2^8$) version follows the same encoding as [BP10]. Consequently, the tool used to benchmark the implementations receive different instructions, or same instructions but in different order. This can be compared to the situations where the result from [GLRS16] (respectively, [BFI21]) could not be verified by [JNRV20] (respectively, us), or the reduction of quantum depth in [ZH22] from [XZL+20]). Note that the same thing was addressed, along with the so-called *compiler-friendly* optimization for quantum depth, in [JSB+25, Appendix C].

by [BCC+24]. The least quantum depth of 10 for an in-place implementation was found by [SF24] that required 131 CNOT gates. However, the current record for the least CNOT count × quantum depth for an in-place implementation is held by the (98 CNOT, 13 quantum depth) implementation by [YWS+24], as far as we know.

In this work, we present our out-of-place implementation of MixColumn, by taking inspiration from [LSL+19]. This achieves the least quantum depth (8) in the literature, to the best of our knowledge. This is used in our shallow/low depth architecture. Discussion about the quantum depth reduction and ancilla qubit reduction can be found in Appendix D.

### 3.7 Implementation of MixColumns

For the 128-bit MixColumns operation (i.e., 4 MixColumn operations), the MixColumn implementation can be scaled up directly. As the MixColumn used in the regular and the shallow versions work in-place, we do not have to consider the impact of ancilla qubits. This, however, is more complicated in case of the shallow/low depth version, as described next.

In the shallow/low-depth version, our out-of-place MixColumn implementation requires ancilla qubits. This observation although hints that we need extra qubits (to work as ancilla), here we show how this is not the case. Recall from the implementation of SubBytes (Section 3.2 and Section 3.3) the S-box implementation is also not in-place, requiring ancilla qubits ($20 \times 60$ , $20 \times 68$ or $20 \times 136$). However, when the combined SubBytes and MixColumns is considered, because of efficient resource sharing, the total qubit count does not increase. Those ancilla qubits are initialized as 0 after one SubBytes operation (to use in the next round), meaning that during the MixColumns operations, those qubits are idle. By reusing those idle qubits as ancilla qubits for the MixColumns, only 64 qubits are required to implement our out-of-place MixColumn based on [LSL+19] (32 as input qubits and 32 as output qubits). Thus, even though the MixColumn implementation is not in-place, at the end, we do not need any extra qubit. So, the qubit count does not increase when SubBytes is counted within the scope.

In other words, the total number of qubit requirement is 64 for any implementation in Table 4 (save for the in-place implementations [ASAM18, XZL+20, ZH22] where it is 32) when the non-standalone implementation of MixColumns (in which MixColumn does not operate in-place) is considered.

In [LPZW23], this sharing method was applied to reduce the ancilla qubits for the SubBytes and SubBytes$^\dagger$. That is, in the earlier version of this paper, we combined SubBytes and MixColumns, but in [LPZW23], the authors also combined SubBytes and SubBytes$^\dagger$. Applying the sharing method to SubBytes and SubBytes$^\dagger$ (presented in [LPZW23]) is efficient, so we have adopted it. The details of the sharing method used in [LPZW23] is described in Section 4.2.

## 4 Architecture of AES Quantum Circuits

There are several architecture for designing quantum circuits of AES which differ in how each stores the 128-qubit output generated from SubBytes in each round. In [GLRS16, ASAM18, LPS20], the basic zig-zag architecture (Figure 6(a)) was adopted that uses 4 lines to save qubits by doing reverses on rounds. In [ZWS+20], an improved zig-zag architecture that requires only 2 lines of qubits (Figure 6(b)) was presented. Specifically, $R_i$ and $R_i^\dagger$ indicate one AES round and its uncomputation, respectively, with the output of each round ($R_i$ or $R_i^\dagger$) computed on the other line (i.e., 128 qubits) as indicated by the controlled arrow.

**Table 5:** Comparison of quantum resources required for variants of AES.

| | AES | #CNOT | #NOT | #Toffoli | $TD$ | #qubits | $TD$-$M$ cost ($TD \times M$) | Full depth | $TD^2$-$M$ cost ($TD^2 \times M$) |
|---|---|---|---|---|---|---|---|---|---|
| 128 | GLRS [GLRS16] | 166548 | 1456 | 151552 | 12672 | 984 | 12469248 | 110799 | 158010310656 |
| | ASAM [ASAM18] | 192832 | 1370 | 150528 | N/A | 976 | N/A | N/A | N/A |
| | LPS [LPS20] | 107960 | 1570 | 16940 | 1880 | 864 | 1624320 | 28927 | 3053721600 |
| | ZWSLW [ZWS+20] | 128517 | 4528 | 19788 | 2016 | 512 | 1032192 | N/A | 2080899072 |
| | HS [HS22] † 18 | 126016 | 2528 | 17888 | 820 | 492 | 403440 | N/A | 330820800 |
| | HS [HS22] † 9 | 126016 | 2528 | 17888 | 1558 | 374 | 582692 | N/A | 907834136 |
| | LXXZZ [LXX+23] | 77984 | 2224 | 19608 | 476 | 474 | 225624 | N/A | 107397024 |
| | LPZW [LPZW23] ◎ | 65736 | 800 | 12920 | 40 | 3688 | 147520 | 840 | 5900800 |
| | LPZW [LPZW23] ◈ | 75024 | 800 | 12920 | 40 | 4844 | 193760 | 770 | 7750400 |
| | SF [SF24] | 64750 | 800 | 12920 | 40 | 3268 | 130720 | N/A | 5228800 |
| | ☆ | 73604 | 816 | 12920 | 76 | 2736 | 207936 | 1288 | 15803136 |
| | ◎ ❁ | 75784 | 816 | 12824 | 40 | 3048 | 121920 | 776 | 4876800 |
| | ◈ | 79492 | 816 | 12824 | 40 | 4200 | 168000 | 748 | 6720000 |
| | ☆ | 62784 | 816 | 12560 | 76 | 2896 | 220096 | 1090 | 16727296 |
| | ◎ ✿ | 65092 | 816 | 12416 | 40 | 3268 | 130720 | 686 | 5228800 |
| | ◈ | 68800 | 816 | 12416 | 40 | 4420 | 176800 | 667 | 7072000 |
| | ☆ | 120480 | 816 | 29640 | 57 | 4256 | 242592 | 1069 | 13827744 |
| | ◎ ❀ | 120812 | 816 | 29496 | 30 | 6128 | 183840 | 665 | 5515200 |
| | ◈ | 124520 | 816 | 29496 | 30 | 7280 | 282816 | 647 | 10181376 |
| 192 | GLRS [GLRS16] | 189432 | 1608 | 172032 | 11088 | 1112 | 12329856 | 96956 | 136713443328 |
| | LPS [LPS20] | 125580 | 1692 | 19580 | 1640 | 896 | 1469440 | 25556 | 2409881600 |
| | ZWSLW [ZWS+20] | 152378 | 5128 | 22380 | 2022 | 640 | 1294080 | N/A | 2616629760 |
| | LXXZZ [LXX+23] | 90832 | 2568 | 22800 | 572 | 538 | 307736 | N/A | 176024992 |
| | LPZW [LPZW23] ◎ | 74456 | 896 | 14552 | 48 | 3944 | 189312 | 1010 | 9086976 |
| | LPZW [LPZW23] ◈ | 85808 | 896 | 14552 | 48 | 5356 | 257088 | 924 | 12340224 |
| | ☆ | 83460 | 904 | 14552 | 92 | 3056 | 281152 | 1534 | 25865984 |
| | ◎ ❁ | 86388 | 904 | 14592 | 48 | 3368 | 161664 | 932 | 7759872 |
| | ◈ | 90920 | 904 | 14592 | 48 | 4776 | 229248 | 900 | 11003904 |
| | ☆ | 71272 | 904 | 14144 | 92 | 3216 | 295872 | 1294 | 295872 |
| | ◎ ✿ | 73620 | 904 | 14000 | 48 | 3588 | 172224 | 819 | 8266752 |
| | ◈ | 78152 | 904 | 14000 | 48 | 4996 | 239808 | 797 | 11510784 |
| | ☆ | 136264 | 904 | 33384 | 69 | 4576 | 315744 | 1270 | 21786336 |
| | ◎ ❀ | 136812 | 904 | 33240 | 36 | 6448 | 232128 | 795 | 8356608 |
| | ◈ | 141344 | 904 | 33240 | 36 | 7856 | 282816 | 773 | 10181376 |
| 256 | GLRS [GLRS16] | 233836 | 1943 | 215040 | 14976 | 1336 | 20007936 | 130929 | 299638849536 |
| | LPS [LPS20] | 151011 | 1992 | 23760 | 2160 | 1232 | 2661120 | 33525 | 5748019200 |
| | ZWSLW [ZWS+20] | 177645 | 6103 | 26774 | 2292 | 768 | 1760256 | N/A | 4034506752 |
| | LXXZZ [LXX+23] | 110688 | 3069 | 27816 | 646 | 602 | 388892 | N/A | 251224232 |
| | LPZW [LPZW23] ◎ | 93288 | 1119 | 18360 | 56 | 4456 | 249536 | 1176 | 13974016 |
| | LPZW [LPZW23] ◈ | 106704 | 1119 | 18360 | 56 | 6124 | 342944 | 1074 | 19204864 |
| | ☆ | 102932 | 1111 | 18088 | 108 | 3376 | 364608 | 1798 | 39377664 |
| | ◎ ❁ | 104464 | 1111 | 17992 | 56 | 3688 | 206528 | 1086 | 11565568 |
| | ◈ | 109820 | 1111 | 17992 | 56 | 5352 | 299712 | 1047 | 16783872 |
| | ☆ | 87780 | 1111 | 17576 | 108 | 3536 | 381888 | 1516 | 41243904 |
| | ◎ ✿ | 89544 | 1111 | 17432 | 56 | 3908 | 218848 | 960 | 12255488 |
| | ◈ | 94900 | 1111 | 17432 | 56 | 5572 | 312032 | 934 | 17473792 |
| | ☆ | 168580 | 1111 | 41496 | 81 | 4896 | 396576 | 1488 | 32122656 |
| | ◎ ❀ | 168548 | 1111 | 41316 | 42 | 6768 | 284256 | 933 | 11938752 |
| | ◈ | 173904 | 1111 | 41316 | 42 | 8432 | 354144 | 907 | 14874048 |

†: Choice of $p$.

☆: Regular version (using Toffoli gate).
◎: Shallow version (using Toffoli gate).
◈: Shallow/low depth version (using Toffoli gate).
❁: S-box with Toffoli depth 4 (low qubit count).
✿: S-box with Toffoli depth 4 (low full depth).
❀: S-box with Toffoli depth 3.

In [ZWS+20], by using a quantum circuit of S-box$^{-1}$, the authors were able to achieve an improved architecture using fewer qubits. The basic pipeline architecture allocates 128-qubits every round and does not need reverses of rounds. In other words, we can state that the zig-zag architecture requires reverse operations on rounds to save qubits,

significantly increasing depth and gates. The pipeline architecture allocates new qubits per round, but does not require reverse operations, reducing in circuit depth and gate count. It is a trade-off matter, but in a sense, a generic pipeline is probably the most efficient architecture for implementing the AES quantum circuits. We argue that, it is significantly more efficient to allocate new 128-qubits per round than doubling the number of gates and depth by performing reverse operations on the rounds to save qubits.

In our approach, where we have allocated many ancilla qubits already, the overhead of increasing the number of qubits according to the architecture (128 qubits per round) is relatively low. Therefore, for our implementation, rather than reducing the number of qubits with the zig-zag method, a pipeline architecture that can reduce the depth by omitting the reverses is more suitable. Note that the reverse operation for ancilla qubits of SubBytes is still performed, and it is entirely distinct from the reverse operation for the output qubits of the round. Figure 7(a) shows the pipeline architecture of our AES-128 quantum circuit in more detail for the regular version, and Figure 7(b) shows the same for the shallow and shallow/low depth versions. To be more precise, each $R_{1\sim10}$ in Figure 6 represents the full round, but each $R_{1\sim10}$ in Figure 7 does not contain SubBytes. The notation SB (with $\oplus$) represents SubBytes followed by CNOT gates.

## 4.1 Regular Version

The regular version focuses on parallelization in SubBytes, key schedule, and MixColumns. Additional ancilla qubits (for parallelization) are allocated and subsequently reused in the next round through the reverse operation.

In our parallel design, the key schedule operates simultaneously with SubBytes and MixColumn operates simultaneously with SubBytes$^\dagger$. The circuit depth is determined by the number of serial operations of SubBytes and SubBytes$^\dagger$ (the depth of SubBytes$^\dagger$ is larger than that of MixColumn). Thus, we adopt the MixColumn with 91 CNOT gates from [YWS+24] (ported to quantum by us), which achieves a lowest CNOT gate count (see Table 4). In this version, the SubBytes of the current round wait until the SubBytes$^\dagger$ of the previous round is completed.

As shown in Figure 7(a), SubBytes generates 128-qubit output and SubBytes$^\dagger$ cleans the ancilla qubits. In total, SubBytes runs 10 times and SubBytes$^\dagger$ runs 9 times (as it is redundant to clean the last round SubBytes) serially, 19 times in total. Similarly, AES-192 operates 23 times (12 SubBytes plus 11 SubBytes$^\dagger$) and AES-256 operates 27 times (14 SubBytes plus 13 SubBytes$^\dagger$).

In SubBytes, S-boxes operate simultaneously. When S-box with Toffoli depth 4 and low full depth is used, the full depth of SubBytes is 58, which is equal to the depth of a single S-box operation. Finally, our AES quantum circuits provide a depth of 1090 (about $58 \times 19$) for AES-128, 1294 (about $58 \times 23$) for AES-256, and 1516 (about $58 \times 27$) for AES-256 (see Table 5).

It is to be noted that the regular version was originally conceived in [JNRV20], but it contained bugs. In short, ancilla qubits for this architecture were not allocated sufficiently, which is related to a Q# issue (details are described in Section 5). Then we revised their idea and proposed the corrected version in this work. We allocated the correct ancilla qubits, specifically in SubBytes, key schedule, MixColumns etc. to operate this architecture properly (see Section 5). Thus, the role/impact of the original authors [JNRV20] cannot be understated in shaping the regular version, it can still be (partly) counted as our innovation.

## 4.2 Shallow Version and Shallow/Low Depth Version

We propose a shallow version in which all possible parts of AES quantum circuits operate, simultaneously. When S-box with Toffoli depth 4 and low full depth is used, this can be

achieved by using 2 sets of ancilla qubits for SubBytes. The same approach is applied in the key schedule; however, in this section, it is explained only for SubBytes for the sake of brevity. In the shallow version, the first SubBytes in Figure 7(b) uses the first set of ancilla qubits. The second SubBytes uses the second set of ancilla qubits, and at the same time SubBytes$^\dagger$ cleans the first set of ancilla qubits. That is, SubBytes$^\dagger$ operates simultaneously with the SubBytes of the next round. Conceptually, this can be thought as all SubBytes$^\dagger$ in Figure 7(a) are pushed one space to the right. This is possible because SubBytes and SubBytes$^\dagger$ do not share any operations by allocating their own ancilla set. Thanks to this parallel structure (using 2 alternative ancilla sets), the shallow version counts the depth for one round as SubBytes (58) + MixColumns (10), which is the ideal depth. As we evaluate performance in terms of depth rather than gate count $\times$ depth, the reduction of depth by $3 = (13 - 10)$ for MixColumn is more effective than reducing $33 = (131 - 98)$ CNOT gates, which is why we opt for the MixColumn implementation from [SF24] instead of that from [YWS$^+$24] (98 CNOT, 13 quantum depth) for our shallow version. The circuit depth of AES-128 is 686 (about 9 rounds $\times$ 68 + 58), that of AES-192 is 819 (about 11 rounds $\times$ 68 + 58), and the same for AES-256 is 960 (about 13 rounds $\times$ 68 + 58). In the shallow version, up to SubBytes$^\dagger$ operates concurrently within one round, providing maximum parallelism.

Finally, the shallow version offer the least Toffoli depth of the S-box's Toffoli depth $\times$ rounds, Toffoli depth $\times$ qubit count, full depth $\times$ qubit count ), Toffoli depth$^2$ $\times$ qubit count and full depth$^2$ $\times$ qubit count.

Similar to [LPZW23], our shallow and shallow/low depth versions attempt to reduce the ancilla qubits of second set by borrowing the idle ancilla qubits of second set (by reverse operations). Just as the authors in [LPZW23] reduced the number of ancilla qubits in the shallow and shallow/low-depth versions, the second set of ancilla qubits could be reduced by borrowing the first set of ancilla qubits in idle state. The authors of [LPZW23] use the cleaned ancilla qubits immediately after the first SubBytes$^\dagger$ in the second SubBytes. Consequently, the second SubBytes can use the idle ancilla qubits from the first SubBytes$^\dagger$, resulting in a reduction in the number of qubits for the second ancilla set (see Figure 7(c)). As a result, for SubBytes and SubBytes$^\dagger$, instead of requiring 2 sets of ancilla qubits, only $\approx 1$ set (more than 1 but lower than 2) is needed. We have adopted this method, and the required number of qubits has been reduced more than in [LPZW23].

The shallow/low depth version replaces only the MixColumn implementation from the shallow version to our out-of-place MixColumn implementation (which is based on [LSL$^+$19]). In the shallow/low-depth version (utilizing out-of-place MixColumn), implementing the lowest depth MixColumn is optimal while also considering the number of ancilla qubits, as they can be replaced by idle ancilla qubits in SubBytes. The low depth version counts the depth for one round as SubBytes (67, 58 or 56) + MixColumns (8) and $32 \times 4$ output qubits for MixColumns are allocated (see Section 3.7). The low depth version of AES offers the least Toffoli depth, full depth, and full depth $\times$ total gates (a metric to estimate quantum attack cost).

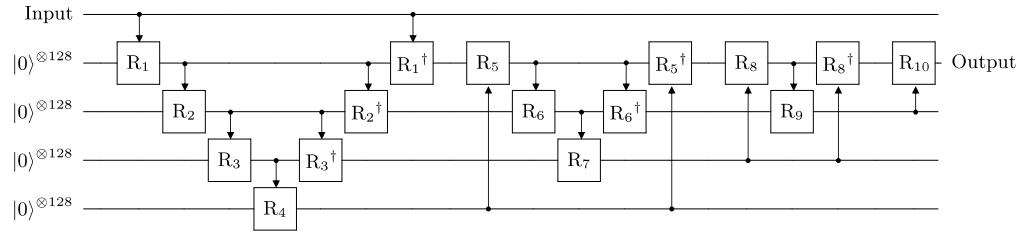
## 4.3 Last Round Optimization

We further reduce the number of qubits by replacing the output qubits in the final round of AES quantum circuits (i.e., 10, 12, and 14 rounds for AES-128, AES-192, and AES-256, respectively) with ancilla qubits used in SubBytes. This optimization is applied to both the shallow and shallow/low-depth versions. Figure 7(c)[21] illustrates the last round optimization by presenting the final 3 rounds of AES-128 using an S-box with a Toffoli depth of 4. The flow of SubBytes and SubBytes$^\dagger$ is presented based on the Toffoli

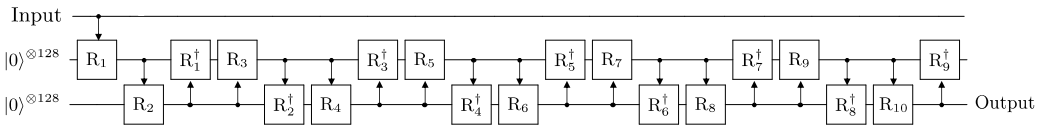
[21] Figure 7(c) includes the sharing method used in [LPZW23], where ancilla qubits are shared between SubBytes and SubBytes$^\dagger$.

depth (i.e., 4 steps), and the ancilla qubits for SubBytes and SubBytes$^\dagger$ are separated (i.e., $anc_{0\sim4}$) according to their use in each Toffoli depth.

Recall that, in the last round, the initialization of ancilla qubits from the prior SubBytes is performed in SubBytes$^\dagger$. The initialized ancilla qubits $(anc_3, anc_2, anc_1)$ are used in the last round, but the fourth one $(anc_0)$ is not needed, as it would only be used in the next round (which does not exist since this is the last round). Thus, we avoid allocating output qubits by replacing them with the ancilla qubits in $anc_0$. With this optimization, we reduce the number of qubits required for the last round output in both the shallow and shallow/low versions. This means that the 128-qubit output for the last round in Figure 7(b) are not actually included in the count.




**(a)** Basic (GLRS, LPS, ASAM)




**(b)** Modified (ZWSLW).

$R_i$: SubBytes → MixColumns → ShiftRows → AddRoundKey for $i = 1, \ldots, 9$.

$R_{10}$: SubBytes → ShiftRows → AddRoundKey.

**Figure 6:** Zig-zag architecture for AES-128 quantum circuit.

# 5 Bug-fixing JNRV (Eurocrypt'20) AES Implementation

In this part, we take a deeper look at the AES implementation and resource estimation by Jaques, Naehrig, Roetteler and Virdia in Eurocrypt'20 [JNRV20]. It is already well-known the resource estimation in their paper was incorrect due to some problem in Q# (unrelated to the coding of [JNRV20]), as already indicated by at least two previous works [ZWS+20, HS22][22] as well as acknowledged by the Eurocrypt'20 authors[23] themselves[24]. Also, one may refer to Appendix C for supplementary discussion on this topic. We fix the Q# bugs and report the corrected benchmarks for the resource requirement of [JNRV20] by porting their codes to ProjectQ.

## 5.1 Issues with Q#

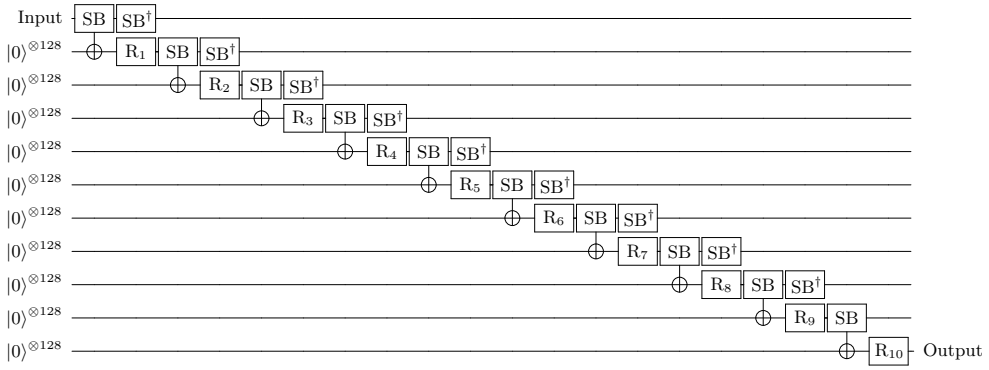
### 5.1.1 Inconsistency and Underestimation of Full Depth

In their AES quantum circuits using Maximov's MixColumn, the AES-192 quantum circuit offers the lowest full depth (see Table 7(b)), although the number of rounds of

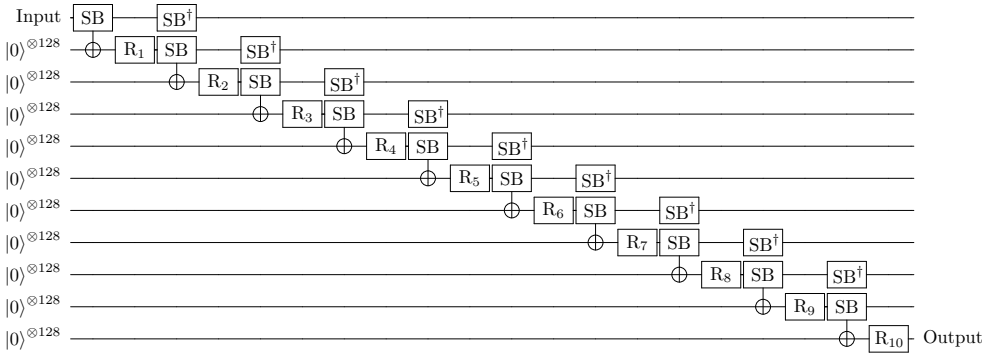
[22] The same bug appeared in context of another cipher, as mentioned in [JBK+22a].

[23] See https://github.com/microsoft/qsharp-runtime/issues/1037 and https://github.com/sam-jaques/grover-blocks/tree/sjaques-version-update#issue-with-estimating-resources.

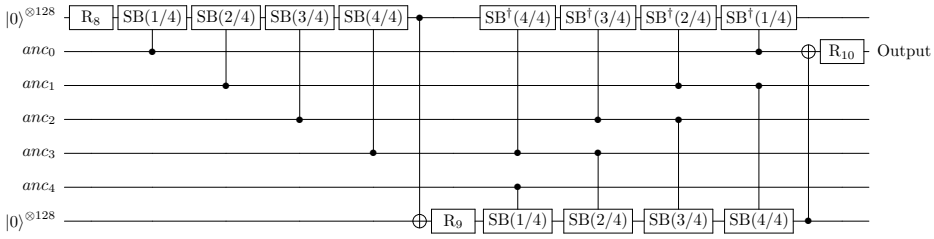
[24] As noted in Section 1, the authors recently updated their own bug-fixing in [JNRV19].




**(a)** Regular version (originally conceived in JNRV but updated by us).




**(b)** Shallow and shallow/low depth versions (ours).




**(c)** Shallow and shallow/low depth versions: Last round optimization (ours).

SB: SubBytes followed by CNOT gates ($\oplus$).

SB$^\dagger$: Clean ancilla qubits used in preceding SubBytes.

R$_i$: MixColumns $\rightarrow$ ShiftRows $\rightarrow$ AddRoundKey for $i = 1, \ldots, 9$; R$_{10}$: ShiftRows $\rightarrow$ AddRoundKey.

**Figure 7:** Pipeline architecture of AES-128.

AES-192 (12 rounds) is higher than that of AES-128 (10 rounds). This case is observed in the zig-zag architecture [GLRS16, LPS20] since the number of key schedules is less in AES-192. However, as a result of analyzing their quantum circuit design (e.g., pipeline and parallel structure) and quantum resources, the full depth should depend on the number of rounds because the key schedule operates in parallel with SubBytes. In other words, in their AES quantum circuits, the full depth should be independent of the number of key schedules. However, their AES-192 quantum circuit has a lower full depth than their AES-128 quantum circuit. Moreover, their AES-256 (14 rounds) quantum circuit has a lower full depth than their AES-128 quantum circuit.

Also, the full depth of their AES-192 and 256 quantum circuits cannot be derived. By analyzing full depth with the quantum resources required for their SubBytes and MixColumn, we believe their report is underestimated. Let us assume the following two things to estimate the full depth for their AES quantum circuit. All S-Boxes of SubBytes operate in parallel (in this case the full depth of SubBytes is 101, see Table 7(a)) and the full depth of round is counted only for SubBytes. Then, about 1212 (12 rounds $\times$ 101) should be the full depth of the AES-192 quantum circuit, and the full depth of the oracle where the AES quantum circuit is operated twice should be about 2424 (12 rounds $\times$ 101 $\times$ 2). Even with these optimistic assumptions, the full depth of the oracle they estimate for AES-192 (1879 in Table 7(b)) cannot be derived. This underestimation also applies to the full depth of the oracle for AES-256, where they estimated 1951 in Table 7(b) $\neq$ about 2828 (14 rounds $\times$ 101 $\times$ 2).

This inconsistency is also observed in AES quantum circuits using in-place MixColumn (full depth is 111, as shown in Table 7(a)). To take one case, the full depth of oracle for AES-256 is 3353 (Table 7(b)). In the AES-256 quantum circuit, MixColumns operates for 13 rounds excluding the last round. Then, even counting only MixColumns, the full depth of oracle for AES-256 is 2886 (13 rounds $\times$ 111 $\times$ 2) even though SubBytes are not counted. If we consider the full depth with SubBytes included (cannot be operated in parallel with MixColumns), the full depth 3353 is lower than expected (i.e., underestimated in [JNRV20]).

## 5.2 Architecture Consideration

If we correct the Eurocrypt'20 implementation [JNRV20] while maintaining depth optimization, the architecture of the bug-fix version is much similar to our regular version. In the regular version, there is a depth overhead associated with the reverse operation for SubBytes and the key schedule (where S-box is used). However, due to the nature of the AND gate, there is less overhead for the reverse operation. Therefore, when applying the AND gate to the bug-fixed version, the depth is significantly lower compared to using the Toffoli gate to fix the bug. Also, the architecture of the bug-fix version uses out-of-place MixColumn implementation by Maximov [Max19], which reduces depth but increases the qubit count. As a result, the bug-fix version using the AND gate and Maximov's MixColumn offers a low full depth, similar to our regular version (but with a higher number of qubits). One thing to note is that the cost of their bug-fixed benchmark in [JNRV19] is higher than that of our bug-fix version.

Conceptually, we can think of the depth optimized bug-fix version as a regular/low version. In contrast, our design philosophy for the regular version is to reduce depth while maintaining a balanced use of qubits. Thus, in the regular version, we adopt an in-place MixColumn implementation [ZH22] rather than an out-of-place MixColumn implementation [LSL$^+$19]. Even so, our regular version, using the improved S-box implementation, provides lower full depth compared to the bug-fix version (regular/low). Furthermore, since our regular version has lower Toffoli depth and qubit count (except for the bug-fix version using in-place MixColumn), we provide improved $TD$-$M$ and $FD$-$M$ costs compared to the bug-fix version.

## 5.3 Corrected Report

To our understanding, some problems arise if the qubits are allocated by the `using` command in Q# (and it affects the non-linear components). However more experiments are to be carried out in order to be completely certain about it.

To patch the bug, we contribute in three major directions:

1. We reflect on the increasing depth in their number of qubits using only one ancilla

set (fixed depth). As shown in Figure 4(a), since the ancilla set is shared, not only SubBytes but also S-boxes of SubWord of the key schedule are operated sequentially. Also, we offer another version (fixed qubit count) that increases the number of qubits while approximating the depth reported in [JNRV20]. As shwon in Figure 4(b), 20 ancilla sets are allocated to operate S-boxes for the SubBytes and the SubWord of the key schedule in parallel. According to the Eurocrypt'22 paper [JNRV20], the authors' design philosophy focuses on optimizing depth.

2. We correct the implementation of MixColumns where the same issue occurs. In Eurocrypt'20 paper [JNRV20], two MixColumn implementations were presented. The in-place method of MixColumn implementation (which uses PLU decomposition, and derived by the authors themselves [JNRV20]) does not cause this issue. On the other hand, similar to S-box, the same issue applies to the MixColumn implementation by Maximov [Max19], which requires ancilla qubits, so this is also solved in the same way as the S-box.

3. We have modified the quantum circuits (SubBytes, key schedule and MixColumns) done by [JNRV20] and re-implemented their algorithm on ProjectQ to bypass the Q# bug. Note that when AND gates are used in large-scale quantum circuits, although resource estimation is possible, checking the test vector becomes infeasible (simulation is impossible). Thus, to verify our bug-fixed implementations, we initially implement quantum circuits using Toffoli gates (using the method from [AMM+13]) instead of directly applying AND gates (which could lead to some coding-related issues) and verify the test vector (Toffoli version). After verification, we cautiously replace Toffoli gates with AND gates in quantum circuits (AND version).

One way to correct the error is to estimate the correct depth by fixing the erroneous parallelism based on the number of qubits reported, which is the fixed depth version. Another way is to increase the number of qubits to satisfy the excessively estimated parallelism, which is the fixed qubit count version. We adopt both approaches and report the modified number of qubits and depth.

In the fixed depth version, when designing quantum circuits using the qubit count reported in [JNRV20], it demonstrates how the depth is increased. Based on these reported qubit counts, it becomes apparent that not all S-boxes and MixColumns (except for in-place) operations can be executed simultaneously. Consequently, the unattainable parallel execution of quantum circuits, as presented in [JNRV20], is rectified by restructuring the operations to be carried out sequentially within the confines of the reported qubit count. As a result of this sequential execution, there is a notable increase in the overall depth of the quantum circuits.

Table 6(a) shows quantum resources for S-box and MixColumns reported in the Eurocrypt'20 paper. Quantum resources in Table 6(a) include cleaning up of used ancilla qubits. Table 6(b) shows quantum resources for AES oracles reported in the Eurocrypt'20 paper. Quantum resources are reported for an oracle rather than a single AES quantum circuit. In the oracle, since the AES quantum circuit operates twice, the estimation of quantum resources for a single AES quantum circuit can be counted in half except for the number of qubits in Table 6(b).

Our results with the bug-fixed Eurocrypt'20 implementation can be found in Tables 7 and 8. Table 7 shows the estimated resources (corrected) for SubBytes, key schedule, MixColumns, and one round where the issue occurs. Tables 7(a) (using Toffoli gate) and 7(c) (using AND gate) correspond to the versions with a fixed depth, while Tables 7(b) (using Toffoli gate) and 7(d) (using AND gate) represent the versions with a fixed qubit count. The change in the corrected depth or qubit count is relatively small for the MixColumns, but significant for the SubBytes. The resources estimated in Table 7 include a reverse operation to clean ancilla qubits. At the end, Table 8 shows the corrected

**Table 6:** Reported benchmarks for JNRV (Eurocrypt'20) implementation of AES.
**(a)** AES-128 gate costs.

| Method | S-box (SubByte) | MixColumn implementation | |
|---|---|---|---|
| | | In-place [JNRV20] | Maximov [Max19] |
| #CNOT | 654 | 1108 | 1248 |
| #1qCliff | 184 | 0 | 0 |
| #$T$ | 136 | 0 | 0 |
| #Measure | 34 | 0 | 0 |
| $T$-depth | 6 | 0 | 0 |
| #qubits ($M$) | 137 | 128 | 318 |
| Full depth | 101 | 111 | 22 |

**(b)** Oracles.

| Method | In-place MixColumn [JNRV20] | | | Maximov's MixColumn [Max19] | | |
|---|---|---|---|---|---|---|
| | AES-128 | AES-192 | AES-256 | AES-128 | AES-192 | AES-256 |
| #CNOT | 292313 | 329697 | 404139 | 294863 | 332665 | 407667 |
| #1qCliff | 84428 | 94316 | 116286 | 84488 | 94092 | 116062 |
| #$T$ | 54908 | 61436 | 75580 | 54908 | 61436 | 75580 |
| #Measure | 13727 | 15359 | 18895 | 13727 | 15359 | 18895 |
| $T$-depth | 121 | 120 | 126 | 121 | 120 | 126 |
| #qubits ($M$) | 1665 | 1985 | 2305 | 2817 | 3393 | 3969 |
| Full depth | 2816 | 2978 | 3353 | 2086 | 1879 | 1951 |

quantum resources for AES quantum circuits, and it is confirmed that the depth or qubit count increases significantly when maintaining the qubit count or depth. Just like Table 7, Tables 8(a) (using Toffoli gate) and 8(c) (using AND gate) correspond to the versions with a fixed depth, while Tables 8(b) (using Toffoli gate) and 8(d) (using AND gate) represent the versions with a fixed qubit count.

# 6 Performance of Quantum Circuits

In this part, we present the performance of our implementations of AES quantum circuits. We use the open-source quantum programming tool ProjectQ to implement and simulate the quantum circuits. An internal library, `ClassicalSimulator`, simulates quantum circuits and verifies test vectors. Quantum resources required to implement quantum circuits are estimated using another library, `ResourceCounter`.

As for the results, Table 5 shows the quantum resources required to implement our AES quantum circuits and previous AES quantum circuits. Although various decompositions exist for the Toffoli gate, Table 5 enables consistent comparison with NCT (NOT, CNOT, Toffoli) level analysis. Table 5 only covers the version using the Toffoli gate, not the version using the AND gate. In [GLRS16, ASAM18], the Itoh–Tsujii based inversion is implemented on a quantum circuit, so many resources are used for SubBytes. In [LPS20, ZWS+20], more efficient quantum circuits are implemented by extending the S-box of [BP10], but the circuit depth is increased due to the serial execution of S-boxes by concentrating on saving qubits. On the other hand, our implementation focuses on minimizing circuit depth while considering the trade-offs for using qubits. In [ZWS+20], the $TD$-$M$ cost metric (where $TD$ is the Toffoli depth, and $M$ is the number of qubits) was used to measure the trade-off of quantum circuits. The $TD$-$M$ cost evaluates the performance of the quantum circuit alone, but in practice, due to depth limitations under the Grover's search, parallelization is necessary. The $TD^2$-$M$ complexity metric in Table 5 demonstrates

**Table 7:** Corrected benchmarks for JNRV (Eurocrypt'20) implementation of AES-128 modules.

**(a)** Fixed depth (using Toffoli gate).

| Method | #CNOT | #1qCliff | #$T$ | $T$-depth | #qubit | Full depth |
|---|---|---|---|---|---|---|
| SubBytes | 12000 | 1220 | 7328 | 768 | 376 | 2672 |
| Key schedule | 3096 | 355 | 1832 | 192 | 248 | 669 |
| MixColumns ✛ | 1248 | 0 | 0 | 0 | 318 | 88 |
| One round† | 16472 | 1507 | 9160 | 960 | 632 | 3417 |

**(b)** Fixed qubit count (using Toffoli gate).

| Method | #CNOT | #1qCliff | #$T$ | $T$-depth | #qubit | Full depth |
|---|---|---|---|---|---|---|
| SubBytes | 12000 | 7328 | 2240 | 48 | 2176 | 167 |
| Key schedule | 3096 | 559 | 1832 | 48 | 608 | 168 |
| MixColumns ✛ | 1248 | 0 | 0 | 0 | 504 | 22 |
| One round† | 16472 | 2799 | 9160 | 48 | 2912 | 178 |

**(c)** Fixed depth (using AND gate).

| Method | #CNOT | #1qCliff | #$T$ | #Measure | $T$-depth | #qubit | Full depth |
|---|---|---|---|---|---|---|---|
| SubBytes | 11136 | 4416 | 2176 | 544 | 96 | 376 | 1744 |
| Key schedule | 2880 | 1103 | 544 | 136 | 24 | 248 | 437 |
| MixColumns ✛ | 1248 | 0 | 0 | 0 | 0 | 318 | 88 |
| One round† | 15392 | 5519 | 2720 | 680 | 120 | 632 | 2260 |

**(d)** Fixed qubit count (using AND gate).

| Method | #CNOT | #1qCliff | #$T$ | #Measure | $T$-depth | #qubit | Full depth |
|---|---|---|---|---|---|---|---|
| SubBytes | 11136 | 4416 | 2176 | 544 | 6 | 2176 | 109 |
| Key schedule | 2880 | 1103 | 544 | 136 | 6 | 608 | 110 |
| MixColumns ✛ | 1248 | 0 | 0 | 0 | 0 | 504 | 22 |
| One round† | 15392 | 5519 | 2720 | 680 | 6 | 2912 | 123 |

✛: Maximov's MixColumn [Max19].
†: One typical round (that includes MixColumn).

that in the trade-off of parallelization under Grover's search, the depth metric becomes significantly more important (this is discussed in more detail in Appendix 2.4). In this work, all AES quantum circuits with reduced depth and quantum gates using a reasonable number of qubits offer the best trade-off.

In [JNRV20], the quantum resources required to implement the circuits for AES were also estimated. However, it seems there were some issues with Q#'s `ResourcesEstimator`[25] used in their work, specially in implementing quantum circuits for SubBytes. Therefore, the results from [JNRV20] are not used here. In the NCT-level analysis (as shown in Table 5), quantum resources are estimated without decomposing Toffoli gates. For a more detailed analysis, we further estimate the resources by decomposing Toffoli and AND gates. Similar to [ZWS+20, Table 10], Table 9 shows the detailed quantum resources by decomposing Toffoli gates (Table 9(a)) and AND gates taken from [JNRV19] (Table 9(b)) for the AES quantum circuits implemented in this work. The Toffoli gate is decomposed into (8 Clifford gates + 7 $T$ gates), and $T$-depth 4, and full depth 8 following to one of the methods (described in Section 3.2) in [AMM+13]. The AND gate requires 1 ancilla qubit and is decomposed into (11 Clifford gates + 4 $T$ gates), and $T$ depth 1, and full depth 8; and the AND$^\dagger$ gate (Figure 10(c)) is decomposed into (5 Clifford gates + 1 Measurement gate), and incurs full depth of 4.

[25] https://github.com/microsoft/qsharp-runtime/issues/192.

**Table 8:** Corrected benchmarks for JNRV (Eurocrypt'20) implementation of AES variants.

**(a)** Fixed depth (using Toffoli gate).

| AES | | #CNOT | #1qCliff | #$T$ | $T$-depth | #qubit | Full depth |
|---|---|---|---|---|---|---|---|
| 128 | | 161982 | 14400 | 91380 | 9576 | 1656 | 33320 |
| 192 | ✚ | 182774 | 16128 | 102372 | 10728 | 1976 | 37328 |
| 256 | | 224214 | 19871 | 126188 | 13224 | 2296 | 46012 |
| 128 | | 163242 | 14994 | 91380 | 9576 | 2808 | 33914 |
| 192 | ✢ | 184314 | 16854 | 102372 | 10728 | 3384 | 38054 |
| 256 | | 226034 | 20729 | 126188 | 13224 | 3960 | 46870 |

**(b)** Fixed qubit count (using Toffoli gate).

| AES | | #CNOT | #1qCliff | #$T$ | $T$-depth | #qubit | Full depth |
|---|---|---|---|---|---|---|---|
| 128 | | 155180 | 14400 | 87200 | 456 | 3936 | 1845 |
| 192 | ✚ | 175972 | 16128 | 98192 | 552 | 4256 | 2232 |
| 256 | | 217412 | 19871 | 122008 | 648 | 4576 | 2625 |
| 128 | | 156440 | 16776 | 87200 | 456 | 5088 | 1612 |
| 192 | ✢ | 177512 | 19032 | 98192 | 552 | 5664 | 1936 |
| 256 | | 219232 | 23303 | 122008 | 648 | 6240 | 2264 |

To replace the AES quantum circuits that use the Toffoli gate with the AND gate, a number of ancilla qubits equal to the maximum number of AND gates operating in parallel is required. The number of ancilla qubits needed for AND gates can be minimized by utilizing idle ancilla qubits that are already allocated for SubBytes or MixColumns. However, since our implementations do not have enough ancilla qubits in an idle state (already fully utilized), we allocate ancilla qubits for AND gates equal to the maximum number operating in parallel. As a result, for the AND gate version using the S-box with Toffoli depth 4, 360 ancilla qubits are needed for replacement; while for the version using the S-box with Toffoli depth 3, ancilla qubits are allocated for replacement.

# 7 Performance of Quantum Key Search

In this part, the corresponding costs for applying Grover's search algorithm to exhaustive key search are estimated based on the proposed quantum circuits for the three variants of AES. We estimate the cost of oracle, which accounts for the largest portion of Grover's search algorithm. The overhead for diffusion operator is negligible compared to oracle and is not difficult to implement. For this reason, it is common to estimate the cost for oracle excluding the diffusion operator [GLRS16, LPS20]. In the oracle, the target cipher's quantum circuit encrypts a known plaintext with the key in the superposition state. The generated ciphertext in the superposition state is compared with the known ciphertext and a reverse operation is performed for Grover's iterations. For comparison, an $n$-multi controlled NOT gate is used to check that the generated ciphertext ($n$-qubit) is a known ciphertext. In Grassl et al. [GLRS16] and Langenberg et al.'s AES paper [LPS20], the authors added $(32n - 84)$ $T$ gates to their estimate for the $n$-multi controlled NOT gate [WR14]. If we estimate the cost of a 128-multi control NOT gate, only 4012 $(= 128 \times 32 - 84)$ $T$ gates increase. However, the total number of gates to operate our AES-128 circuit (regular version using S-box with Toffoli depth 4) in the oracle is already 532960 (the number of $T$ gates is 180880). That is, there is no significant change in the number of gates. In contrast, the $T$-depth overhead is relatively high. However, the increase in depth was also ignored in [GLRS16, LPS20]. Also in [JNRV20], the estimation of the $n$-multi controlled NOT gate was totally ignored. So, for the $n$-multi controlled NOT

**(c)** Fixed depth (using AND gate).

| AES | | #CNOT | #1qCliff | #$T$ | #Measure | $T$-depth | #qubit | Full depth |
|---|---|---|---|---|---|---|---|---|
| 128 | | 151540 | 55200 | 27200 | 6800 | 1200 | 1656 | 21801 |
| 192 | ✚ | 171036 | 61824 | 30464 | 7616 | 1440 | 1976 | 24417 |
| 256 | | 209668 | 76175 | 37536 | 9384 | 1680 | 2296 | 30085 |
| 128 | | 152800 | 55200 | 27200 | 6800 | 1200 | 2808 | 22413 |
| 192 | ✛ | 172576 | 61824 | 30464 | 7616 | 1440 | 3384 | 25165 |
| 256 | | 211488 | 76175 | 37536 | 9384 | 1680 | 3960 | 30969 |

**(d)** Fixed qubit count (using AND gate).

| AES | | #CNOT | #1qCliff | #$T$ | #Measure | $T$-depth | #qubit | Full depth |
|---|---|---|---|---|---|---|---|---|
| 128 | | 151540 | 55200 | 27200 | 6800 | 60 | 3936 | 1786 |
| 192 | ✚ | 171036 | 61824 | 30464 | 7616 | 72 | 4256 | 2156 |
| 256 | | 209668 | 76175 | 37536 | 9384 | 84 | 4576 | 2528 |
| 128 | | 152800 | 55200 | 27200 | 6800 | 60 | 5088 | 1123 |
| 192 | ✛ | 172576 | 61824 | 30464 | 7616 | 72 | 5664 | 1346 |
| 256 | | 211488 | 76175 | 37536 | 9384 | 84 | 6240 | 1570 |

✚: In-place MixColumn [JNRV20]. | ✛: Maximov's MixColumn [Max19].

gate, we estimate the number of $T$ gates to be $(32n - 84)$ according to the decomposition method in [WR14] and $T$-depth is maintained.

To recover the secret key (which is not a spurious key) uniquely in the quantum exhaustive key search, Grassl et al. in [GLRS16] estimated the attack cost for $r$ known (plaintext, ciphertext) pairs ($r = 3$, $r = 4$ and $r = 5$, respectively). Later in [LPS20], Langenberg et al. explained that $r = \lceil k/n \rceil$ (key size/block size) is sufficient to successfully recover a unique key. The authors in [JNRV20] also estimated the cost for the same $r$ (plaintext, ciphertext) pairs in [LPS20] through detailed computations. Following this approach, we also estimate the cost of recovering a unique key for $r = \lceil k/n \rceil$ (plaintext, ciphertext) pairs. When $r = 1$, the quantum circuit of the target block cipher is serially executed twice in oracle. Thus, the cost of the oracle is twice that required to implement a quantum circuit, excluding qubits. When $r \geq 2$, $r$ target block quantum circuits are executed twice in parallel, and the following should be considered in cost estimation. Although $r \geq 2$ plaintexts are used, only one input key is used, so the cost for key schedule should be estimated only once. Finally, the complexity of quantum exhaustive key search for the target block cipher is roughly the cost of oracle $\times \lfloor \frac{\pi}{4}\sqrt{2^k} \rfloor$ (where $k$ is the key size). The complexity figures are estimated at the (Clifford + T) level and computed as the number of total decomposed gates $\times$ full depth.

We show the cost of quantum key search by the Grover's algorithm for AES-128, AES-192, AES-256; with the two S-boxes (i.e., with Toffoli depth of 4 and 3) in Table 10(a) (using Toffoli gate) and Table 10(b) (using AND gate). Based on Table 10, we can determine the optimal strategy for implementing the Grover's search algorithm for each AES variant while adhering to the depth constraint. For AES-128 (full depth $\leq 2^{96}$), parallelization is not essential since it does not fall under the MAXDEPTH limit. Thus, without considering parallelization, the shallow/low depth version using S-box with Toffoli depth 4 (low full depth) has the lowest attack complexity ($FD$-$G$). However, when considering the more realistic metric of $FD$-$M$ cost, the shallow version using S-box with Toffoli depth 4 (low full depth) shows the highest efficiency. If the $T$-depth metric for error correction takes priority (i.e., $Td$-$M$ cost), then the shallow version using S-box with Toffoli depth 4 (low qubit count and using Toffoli gate) and the regular version using S-box with Toffoli depth 4 (low qubit count and using AND gate) are the optimal choices (although it is not shown in Tables 10(a) and 10(b), it can be found in Tables 9(a) and 9(b)). In

**Table 9:** Quantum circuit resources required for variants of AES (this work).
**(a)** Using Toffoli gate.

| AES | | #CNOT | #1qCliff | $\#T$ | $T$-depth ($Td$) | #qubit ($M$) | Full depth ($FD$) | $Td$-$M$ cost ($Td \times M$) | $FD$-$M$ cost ($FD \times M$) |
|---|---|---|---|---|---|---|---|---|---|
| 128 | ☆ | 148244 | 14416 | 87560 | 304 | 2736 | 1288 | 831744 | 3523968 |
| | ◎ ❁ | 150064 | 19264 | 87104 | 160 | 3048 | 776 | 487680 | 2365248 |
| | ◈ | 153772 | 19264 | 87104 | 160 | 4200 | 748 | 672000 | 3141600 |
| | ☆ | 138144 | 15496 | 87920 | 304 | 2896 | 1090 | 880384 | 3156640 |
| | ◎ ✿ | 139588 | 19168 | 86912 | 160 | 3268 | 686 | 522880 | 2241848 |
| | ◈ | 143296 | 19168 | 86912 | 160 | 4420 | 667 | 707200 | 2948140 |
| | ☆ | 298320 | 40496 | 207480 | 228 | 4256 | 1069 | 970368 | 4549664 |
| | ◎ ❀ | 297788 | 39168 | 206472 | 120 | 6128 | 665 | 735360 | 4075120 |
| | ◈ | 301496 | 39168 | 206472 | 120 | 7280 | 647 | 873600 | 4710160 |
| 192 | ☆ | 167508 | 16136 | 98600 | 368 | 3056 | 1534 | 1124608 | 4687904 |
| | ◎ ❁ | 170804 | 21568 | 99008 | 192 | 3368 | 932 | 646656 | 3138976 |
| | ◈ | 175336 | 21568 | 99008 | 192 | 4776 | 900 | 916992 | 4298400 |
| | ☆ | 156136 | 173496 | 99008 | 368 | 3216 | 1294 | 1183488 | 4161504 |
| | ◎ ✿ | 157620 | 21272 | 98000 | 192 | 3588 | 819 | 688896 | 2938572 |
| | ◈ | 162152 | 21272 | 98000 | 192 | 4996 | 797 | 959232 | 3981812 |
| | ☆ | 336568 | 43192 | 233688 | 276 | 4576 | 1270 | 1262976 | 5811520 |
| | ◎ ❀ | 336252 | 43864 | 232680 | 144 | 6448 | 795 | 928512 | 5126160 |
| | ◈ | 340784 | 43864 | 232680 | 144 | 7856 | 773 | 1131264 | 6072688 |
| 256 | ☆ | 207364 | 19879 | 122520 | 432 | 3376 | 1798 | 1458432 | 6070048 |
| | ◎ ❁ | 208320 | 26703 | 121944 | 224 | 3688 | 1086 | 826112 | 4005168 |
| | ◈ | 213676 | 240379 | 121944 | 224 | 5352 | 1047 | 1198848 | 5603544 |
| | ☆ | 193236 | 21415 | 123032 | 432 | 3536 | 1516 | 1527552 | 5360576 |
| | ◎ ✿ | 193936 | 26607 | 122024 | 224 | 3908 | 960 | 875392 | 3751680 |
| | ◈ | 199292 | 26607 | 122024 | 224 | 5572 | 934 | 1248128 | 5204248 |
| | ☆ | 417556 | 53383 | 290472 | 324 | 4896 | 1488 | 1586304 | 7285248 |
| | ◎ ❀ | 416444 | 53983 | 289212 | 168 | 6768 | 933 | 1137024 | 6314544 |
| | ◈ | 421800 | 53983 | 289212 | 168 | 8432 | 907 | 1416576 | 7647824 |

☆: Regular version. ❁: S-box with Toffoli depth 4 (low qubit count).
◎: Shallow version. ✿: S-box with Toffoli depth 4 (low full depth).
◈: Shallow/low depth version. ❀: S-box with Toffoli depth 3.

contrast to AES-128, AES-192 and AES-256 require parallelization of the Grover' search due to the MAXDEPTH limitation. As specified in Appendix 2.4, parallelizing Grover's search is highly inefficient, and in such cases, we should minimize $FD^2$-$M$ and $Td^2$-$M$ costs (i.e., Cost under MAXDEPTH in Table 10). Therefore, under the MAXDEPTH limit, the shallow version using S-box with Toffoli depth 4 (low full depth) is the most efficient in terms of full depth metric ($FD^2$-$M$). If we consider the $T$-depth ($Td^2$-$M$), then the shallow version using S-box with Toffoli depth 4 (low qubit count and using Toffoli gate) and the regular version using S-box with Toffoli depth 4 (low qubit count and using AND gate) are the optimal choices.

Additionally, a quick comparison of NIST's security level (under the Grover's search) of our work together with the previous works is given in Table 11. As it can be seen, when compared with the current state-of-the-art security bounds, we reduce the quantum complexity for running the Grover's search on the AES family, thereby setting up a new benchmark for the NIST security levels. The complexity is calculated in terms of the

**(b)** Using AND gate.

| AES | | #CNOT | #1qCliff | #$T$ | #Measure | $T$-depth ($Td$) | #qubit ($M$) | Full depth ($FD$) | $Td$-$M$ cost ($Td \times M$) | $FD$-$M$ cost ($FD \times M$) |
|---|---|---|---|---|---|---|---|---|---|---|
| 128 | ☆ | 134124 | 43896 | 27200 | 6120 | 40 | 2968 | 1021 | 118720 | 3030328 |
| | ◎ ❁ | 136208 | 43608 | 27200 | 6024 | 40 | 3408 | 716 | 136320 | 2440128 |
| | ◈ | 139916 | 43608 | 27200 | 6024 | 40 | 4688 | 684 | 187520 | 3206592 |
| | ☆ | 120944 | 40296 | 27200 | 5760 | 40 | 3160 | 826 | 126400 | 2610160 |
| | ◎ ✿ | 123156 | 39936 | 27200 | 5580 | 40 | 3700 | 666 | 148000 | 2464200 |
| | ◈ | 126864 | 39936 | 27200 | 5580 | 40 | 4852 | 647 | 194080 | 3139244 |
| | ☆ | 259320 | 94056 | 62400 | 14040 | 30 | 4864 | 895 | 145920 | 4353280 |
| | ◎ ❀ | 258900 | 93456 | 62400 | 13860 | 30 | 6864 | 635 | 205920 | 4358640 |
| | ◈ | 262608 | 93456 | 62400 | 13860 | 30 | 8016 | 617 | 240480 | 4945872 |
| 192 | ☆ | 151324 | 49456 | 30464 | 6936 | 48 | 3288 | 1209 | 157824 | 3975192 |
| | ◎ ❁ | 154292 | 49672 | 30464 | 6976 | 48 | 3728 | 855 | 178944 | 3187440 |
| | ◈ | 158824 | 49672 | 30464 | 6976 | 48 | 5264 | 817 | 252672 | 4300688 |
| | ☆ | 136488 | 45376 | 30464 | 6528 | 48 | 3480 | 975 | 167040 | 3393000 |
| | ◎ ✿ | 139100 | 45088 | 30464 | 6384 | 48 | 4020 | 795 | 192960 | 3195900 |
| | ◈ | 143632 | 45088 | 30464 | 6384 | 48 | 5428 | 773 | 260544 | 4195844 |
| | ☆ | 291952 | 105952 | 69888 | 15912 | 36 | 5184 | 895 | 186624 | 5484672 |
| | ◎ ❀ | 292228 | 105472 | 69888 | 15768 | 36 | 7184 | 759 | 258624 | 5452656 |
| | ◈ | 296760 | 105472 | 69888 | 15768 | 36 | 8592 | 737 | 309312 | 6332304 |
| - | ☆ | 186708 | 61519 | 37536 | 8704 | 56 | 3608 | 1415 | 202048 | 5105320 |
| | ◎ ❁ | 170320 | 61231 | 37536 | 8608 | 56 | 4048 | 1002 | 226688 | 4056096 |
| | ◈ | 193404 | 61231 | 37536 | 8608 | 56 | 5712 | 957 | 319872 | 5466384 |
| 256 | ☆ | 168284 | 56399 | 37536 | 8192 | 56 | 3800 | 1139 | 212800 | 4328200 |
| | ◎ ✿ | 170320 | 56111 | 37536 | 8048 | 56 | 4340 | 932 | 243040 | 4044880 |
| | ◈ | 175676 | 56111 | 37536 | 8048 | 56 | 6004 | 906 | 336224 | 5439624 |
| | ☆ | 360772 | 131743 | 86112 | 19968 | 42 | 5504 | 1238 | 231168 | 6813952 |
| | ◎ ❀ | 360400 | 131143 | 86112 | 19788 | 42 | 7504 | 891 | 315168 | 6686064 |
| | ◈ | 365756 | 131143 | 86112 | 19788 | 42 | 9168 | 865 | 385056 | 7930320 |

☆: Regular version. ❁: S-box with Toffoli depth 4 (low qubit count).
◎: Shallow version. ✿: S-box with Toffoli depth 4 (low full depth).
◈: Shallow/low depth version. ❀: S-box with Toffoli depth 3.

product of decomposed (Clifford and $T$) gate count and full depth. Also, the MAXDEPTH constraint (see Appendix 2.4) is not considered in the computation. For instance, the figure of $2^{156.2630}$ corresponding to the shallow/low version for AES-128 in Table 11 is computed as the product of the total number of decomposed gates and the full depth for $2^{64}$ (i.e., square-root bound of the exhaustive case) searches (required to run Grover's search). If the MAXDEPTH constraint is to be considered, one has scale down the complexity figures by dividing by the MAXDEPTH constant.

# 8 Conclusion

In this work, we collate multiple research contributions, including the up-to-date optimizations on the building blocks of the ciphers in one place; whence significantly reducing the quantum circuit complexity for the AES family of block ciphers. Apart from that, we take a deeper look into the overall architecture design as well as efficient S-box and MixColumn implementations.

Among other results, we show the least Toffoli depth and full depth implementations of all variants of AES (more than 97% and 95% improvement from [ZWS+20] and [HS22] respectively). At the same time, we improve the Toffoli depth - qubit count product by more than 87% and 69% and more than 99% and 98% in the Toffoli depth squared ×

**Table 10:** Quantum resources required for Grover's search on AES (this work) and key costs for parallelization.

**(a)** Using Toffoli gate.

| AES | | $r$ | #qubit ($M$) | Total gates ($G$) | Full depth ($FD$) | $FD$-$G$ cost ($FD \times G$) | $FD$-$M$ cost ($FD \times M$) | Cost under MAXDEPTH | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | $FD^2$-$M$ | $Td^2$-$M$ |
| 128 | ☆ | 1 | 2737 | $1.511 \cdot 2^{82}$ | $1.976 \cdot 2^{74}$ | $1.493 \cdot 2^{157}$ | $1.320 \cdot 2^{86}$ | $1.303 \cdot 2^{161}$ | $1.159 \cdot 2^{157}$ |
| | ◎ ❁ | | 3049 | $1.549 \cdot 2^{82}$ | $1.190 \cdot 2^{74}$ | $1.843 \cdot 2^{156}$ | $1.772 \cdot 2^{85}$ | $1.054 \cdot 2^{160}$ | **$1.431 \cdot 2^{155}$** |
| | ◈ | | 4201 | $1.571 \cdot 2^{82}$ | $1.147 \cdot 2^{74}$ | $1.802 \cdot 2^{156}$ | $1.176 \cdot 2^{86}$ | $1.350 \cdot 2^{160}$ | $1.971 \cdot 2^{155}$ |
| | ☆ | | 2897 | $1.459 \cdot 2^{82}$ | $1.672 \cdot 2^{74}$ | $1.220 \cdot 2^{157}$ | $1.182 \cdot 2^{86}$ | $1.976 \cdot 2^{160}$ | $1.227 \cdot 2^{157}$ |
| | ◎ ✿ | | 3629 | $1.484 \cdot 2^{82}$ | $1.053 \cdot 2^{74}$ | $1.562 \cdot 2^{156}$ | **$1.680 \cdot 2^{85}$** | **$1.768 \cdot 2^{159}$** | $1.534 \cdot 2^{155}$ |
| | ◈ | | 4421 | $1.506 \cdot 2^{82}$ | $1.023 \cdot 2^{74}$ | **$1.542 \cdot 2^{156}$** | $1.104 \cdot 2^{86}$ | $1.130 \cdot 2^{160}$ | $1.037 \cdot 2^{156}$ |
| | ☆ | | 4257 | $1.643 \cdot 2^{83}$ | $1.640 \cdot 2^{74}$ | $1.347 \cdot 2^{158}$ | $1.704 \cdot 2^{86}$ | $1.397 \cdot 2^{161}$ | $1.016 \cdot 2^{157}$ |
| | ◎ ❀ | | 6129 | $1.634 \cdot 2^{83}$ | $1.021 \cdot 2^{74}$ | $1.668 \cdot 2^{157}$ | $1.527 \cdot 2^{86}$ | $1.558 \cdot 2^{160}$ | $1.613 \cdot 2^{155}$ |
| | ◈ | | 7281 | $1.645 \cdot 2^{83}$ | $1.986 \cdot 2^{73}$ | $1.634 \cdot 2^{157}$ | $1.765 \cdot 2^{86}$ | $1.753 \cdot 2^{160}$ | $1.917 \cdot 2^{155}$ |
| 192 | ☆ | 2 | 5681 | $1.578 \cdot 2^{115}$ | $1.177 \cdot 2^{107}$ | $1.857 \cdot 2^{222}$ | $1.632 \cdot 2^{119}$ | $1.920 \cdot 2^{226}$ | $1.767 \cdot 2^{222}$ |
| | ◎ ❁ | | 6217 | $1.619 \cdot 2^{115}$ | $1.429 \cdot 2^{106}$ | $1.156 \cdot 2^{222}$ | $1.084 \cdot 2^{119}$ | $1.549 \cdot 2^{225}$ | **$1.049 \cdot 2^{221}$** |
| | ◈ | | 9033 | $1.646 \cdot 2^{115}$ | $1.380 \cdot 2^{106}$ | $1.135 \cdot 2^{222}$ | $1.521 \cdot 2^{119}$ | $1.049 \cdot 2^{226}$ | $1.524 \cdot 2^{221}$ |
| | ☆ | | 5969 | $1.526 \cdot 2^{115}$ | $1.985 \cdot 2^{106}$ | $1.514 \cdot 2^{222}$ | $1.446 \cdot 2^{119}$ | $1.435 \cdot 2^{226}$ | $1.857 \cdot 2^{222}$ |
| | ◎ ✿ | | 6613 | $1.546 \cdot 2^{115}$ | $1.256 \cdot 2^{106}$ | $1.941 \cdot 2^{221}$ | **$1.013 \cdot 2^{119}$** | **$1.273 \cdot 2^{225}$** | $1.115 \cdot 2^{221}$ |
| | ◈ | | 9429 | $1.573 \cdot 2^{115}$ | $1.222 \cdot 2^{106}$ | **$1.922 \cdot 2^{221}$** | $1.406 \cdot 2^{119}$ | $1.718 \cdot 2^{225}$ | $1.591 \cdot 2^{221}$ |
| | ☆ | | 8417 | $1.36 \cdot 2^{117}$ | $1.948 \cdot 2^{106}$ | $1.324 \cdot 2^{224}$ | $1.000 \cdot 2^{120}$ | $1.949 \cdot 2^{226}$ | $1.469 \cdot 2^{222}$ |
| | ◎ ❀ | | 11761 | $1.709 \cdot 2^{116}$ | $1.219 \cdot 2^{106}$ | $1.041 \cdot 2^{223}$ | $1.750 \cdot 2^{119}$ | $1.066 \cdot 2^{226}$ | $1.118 \cdot 2^{221}$ |
| | ◈ | | 14577 | $1.722 \cdot 2^{116}$ | $1.186 \cdot 2^{106}$ | $1.021 \cdot 2^{223}$ | $1.055 \cdot 2^{120}$ | $1.251 \cdot 2^{226}$ | $1.386 \cdot 2^{221}$ |
| 256 | ☆ | 2 | 6257 | $1.905 \cdot 2^{147}$ | $1.379 \cdot 2^{139}$ | $1.313 \cdot 2^{287}$ | $1.053 \cdot 2^{152}$ | $1.452 \cdot 2^{291}$ | $1.339 \cdot 2^{287}$ |
| | ◎ ❁ | | 6881 | $1.944 \cdot 2^{147}$ | $1.666 \cdot 2^{138}$ | $1.619 \cdot 2^{286}$ | $1.399 \cdot 2^{151}$ | $1.165 \cdot 2^{290}$ | **$1.579 \cdot 2^{285}$** |
| | ◈ | | 10209 | $1.976 \cdot 2^{147}$ | $1.606 \cdot 2^{138}$ | $1.586 \cdot 2^{286}$ | $1.000 \cdot 2^{152}$ | $1.607 \cdot 2^{290}$ | $1.171 \cdot 2^{286}$ |
| | ☆ | | 6545 | $1.838 \cdot 2^{147}$ | $1.163 \cdot 2^{139}$ | $1.068 \cdot 2^{287}$ | $1.858 \cdot 2^{151}$ | $1.08 \cdot 2^{291}$ | $1.401 \cdot 2^{287}$ |
| | ◎ ✿ | | 7189 | $1.866 \cdot 2^{147}$ | $1.472 \cdot 2^{138}$ | $1.373 \cdot 2^{286}$ | **$1.291 \cdot 2^{151}$** | **$1.901 \cdot 2^{289}$** | $1.649 \cdot 2^{285}$ |
| | ◈ | | 10517 | $1.546 \cdot 2^{148}$ | $1.432 \cdot 2^{138}$ | **$1.358 \cdot 2^{286}$** | $1.838 \cdot 2^{151}$ | $1.316 \cdot 2^{290}$ | $1.206 \cdot 2^{286}$ |
| | ☆ | | 8993 | $1.035 \cdot 2^{149}$ | $1.141 \cdot 2^{139}$ | $1.180 \cdot 2^{288}$ | $1.252 \cdot 2^{152}$ | $1.429 \cdot 2^{291}$ | $1.080 \cdot 2^{287}$ |
| | ◎ ❀ | | 12337 | $1.033 \cdot 2^{149}$ | $1.431 \cdot 2^{138}$ | $1.478 \cdot 2^{287}$ | $1.077 \cdot 2^{152}$ | $1.541 \cdot 2^{290}$ | $1.589 \cdot 2^{285}$ |
| | ◈ | | 15665 | $1.041 \cdot 2^{149}$ | $1.391 \cdot 2^{138}$ | $1.448 \cdot 2^{287}$ | $1.329 \cdot 2^{152}$ | $1.849 \cdot 2^{290}$ | $1.009 \cdot 2^{286}$ |

☆: Regular version. ❁: S-box with Toffoli depth 4 (low qubit count).
◎: Shallow version. ✿: S-box with Toffoli depth 4 (low full depth).
◈: Shallow/low depth version. ❀: S-box with Toffoli depth 3.

qubit count compared to the respective papers. In total, we present 26 implementations per variant of AES (including bug-fixing of [JNRV20]), each incorporating a special design idea/optimization. We show improvement over the recent papers, including Asiacrypt'22 [HS22], Asiacrypt'23 [LPZW23] and Asiacrypt'24 [SF24]. From what we can tell, this work shows the most advanced results in quantum analysis of AES.

Most recent papers about AES quantum implementations focus on reducing the number of qubits [GLRS16, LPS20, ZWS+20, ASAM18, WWL22]. In our work, one of the major ways we lower the depth metrics is by allowing a relatively higher number of qubits, so that the product terms (i.e., when the number of qubits is multiplied with the circuit depth metrics or decomposed gate counts) becomes smaller. Having a lower circuit depth also makes it easier to maximize the number of iterations (required to run the Grover's search algorithm) and thus is a crucial factor in reducing the cost of evaluating the overall quantum search complexity for exhaustive key search a cipher.

**(b)** Using AND gate.

| AES | | $r$ | #qubit ($M$) | Total gates ($G$) | Full depth ($FD$) | $FD$-$G$ cost ($FD \times G$) | $FD$-$M$ cost ($FD \times M$) | Cost under MAXDEPTH | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | $FD^2$-$M$ | $Td^2$-$M$ |
| 128 | ☆ | 1 | 2969 | $1.278 \cdot 2^{82}$ | $1.566 \cdot 2^{74}$ | $1.001 \cdot 2^{157}$ | $1.135 \cdot 2^{86}$ | $1.778 \cdot 2^{160}$ | $1.360 \cdot 2^{151}$ |
| | ◎ ❁ | | 3409 | $1.289 \cdot 2^{82}$ | $1.099 \cdot 2^{74}$ | $1.416 \cdot 2^{156}$ | **$1.828 \cdot 2^{85}$** | $1.004 \cdot 2^{160}$ | $1.562 \cdot 2^{151}$ |
| | ◈ | | 4689 | $1.311 \cdot 2^{82}$ | $1.050 \cdot 2^{74}$ | $1.376 \cdot 2^{156}$ | $1.201 \cdot 2^{86}$ | $1.261 \cdot 2^{160}$ | $1.074 \cdot 2^{152}$ |
| | ☆ | | 3161 | $1.176 \cdot 2^{82}$ | $1.268 \cdot 2^{74}$ | $1.490 \cdot 2^{156}$ | $1.956 \cdot 2^{85}$ | $1.239 \cdot 2^{160}$ | $1.448 \cdot 2^{151}$ |
| | ◎ ✿ | | 3701 | $1.186 \cdot 2^{82}$ | $1.021 \cdot 2^{74}$ | $1.211 \cdot 2^{156}$ | $1.845 \cdot 2^{85}$ | **$1.885 \cdot 2^{159}$** | $1.695 \cdot 2^{151}$ |
| | ◈ | | 4853 | $1.208 \cdot 2^{82}$ | $1.986 \cdot 2^{73}$ | **$1.200 \cdot 2^{156}$** | $1.176 \cdot 2^{86}$ | $1.168 \cdot 2^{160}$ | $1.111 \cdot 2^{152}$ |
| | ☆ | | 4865 | $1.294 \cdot 2^{83}$ | $1.373 \cdot 2^{74}$ | $1.776 \cdot 2^{157}$ | $1.630 \cdot 2^{86}$ | $1.119 \cdot 2^{161}$ | **$1.281 \cdot 2^{151}$** |
| | ◎ ❀ | | 6865 | $1.290 \cdot 2^{83}$ | $1.949 \cdot 2^{73}$ | $1.257 \cdot 2^{157}$ | $1.633 \cdot 2^{86}$ | $1.591 \cdot 2^{160}$ | $1.807 \cdot 2^{151}$ |
| | ◈ | | 8017 | $1.301 \cdot 2^{83}$ | $1.893 \cdot 2^{73}$ | $1.231 \cdot 2^{157}$ | $1.852 \cdot 2^{86}$ | $1.752 \cdot 2^{160}$ | $1.055 \cdot 2^{152}$ |
| 192 | ☆ | 2 | 6073 | $1.332 \cdot 2^{115}$ | $1.854 \cdot 2^{106}$ | $1.234 \cdot 2^{222}$ | $1.374 \cdot 2^{119}$ | $1.274 \cdot 2^{226}$ | $1.018 \cdot 2^{217}$ |
| | ◎ ❁ | | 6865 | $1.347 \cdot 2^{115}$ | $1.311 \cdot 2^{106}$ | $1.765 \cdot 2^{221}$ | **$1.098 \cdot 2^{119}$** | $1.440 \cdot 2^{225}$ | $1.150 \cdot 2^{217}$ |
| | ◈ | | 9937 | $1.374 \cdot 2^{115}$ | $1.253 \cdot 2^{106}$ | $1.721 \cdot 2^{221}$ | $1.519 \cdot 2^{119}$ | $1.904 \cdot 2^{225}$ | $1.665 \cdot 2^{217}$ |
| | ☆ | | 6425 | $1.224 \cdot 2^{115}$ | $1.496 \cdot 2^{106}$ | $1.831 \cdot 2^{221}$ | $1.173 \cdot 2^{119}$ | $1.755 \cdot 2^{225}$ | $1.077 \cdot 2^{217}$ |
| | ◎ ✿ | | 7397 | $1.234 \cdot 2^{115}$ | $1.219 \cdot 2^{106}$ | $1.504 \cdot 2^{221}$ | $1.100 \cdot 2^{119}$ | **$1.341 \cdot 2^{225}$** | $1.240 \cdot 2^{217}$ |
| | ◈ | | 10213 | $1.261 \cdot 2^{115}$ | $1.186 \cdot 2^{106}$ | **$1.495 \cdot 2^{221}$** | $1.478 \cdot 2^{119}$ | $1.753 \cdot 2^{225}$ | $1.712 \cdot 2^{217}$ |
| | ☆ | | 9489 | $1.349 \cdot 2^{116}$ | $1.623 \cdot 2^{106}$ | $1.094 \cdot 2^{223}$ | $1.879 \cdot 2^{119}$ | $1.525 \cdot 2^{226}$ | **$1.773 \cdot 2^{216}$** |
| | ◎ ❀ | | 13089 | $1.348 \cdot 2^{116}$ | $1.165 \cdot 2^{106}$ | $1.570 \cdot 2^{222}$ | $1.861 \cdot 2^{119}$ | $1.084 \cdot 2^{226}$ | $1.223 \cdot 2^{217}$ |
| | ◈ | | 15905 | $1.361 \cdot 2^{116}$ | $1.130 \cdot 2^{106}$ | $1.537 \cdot 2^{222}$ | $1.096 \cdot 2^{120}$ | $1.239 \cdot 2^{226}$ | $1.486 \cdot 2^{217}$ |
| 256 | ☆ | 2 | 6649 | $1.605 \cdot 2^{147}$ | $1.085 \cdot 2^{139}$ | $1.741 \cdot 2^{286}$ | $1.761 \cdot 2^{151}$ | $1.910 \cdot 2^{290}$ | $1.499 \cdot 2^{281}$ |
| | ◎ ❁ | | 7441 | $1.504 \cdot 2^{147}$ | $1.537 \cdot 2^{138}$ | $1.155 \cdot 2^{286}$ | $1.396 \cdot 2^{151}$ | $1.072 \cdot 2^{290}$ | $1.678 \cdot 2^{281}$ |
| | ◈ | | 10769 | $1.643 \cdot 2^{147}$ | $1.468 \cdot 2^{138}$ | $1.205 \cdot 2^{286}$ | $1.929 \cdot 2^{151}$ | $1.416 \cdot 2^{290}$ | $1.214 \cdot 2^{282}$ |
| | ☆ | | 7001 | $1.474 \cdot 2^{147}$ | $1.747 \cdot 2^{138}$ | $1.287 \cdot 2^{286}$ | $1.493 \cdot 2^{151}$ | $1.304 \cdot 2^{290}$ | $1.579 \cdot 2^{281}$ |
| | ◎ ✿ | | 7973 | $1.483 \cdot 2^{147}$ | $1.429 \cdot 2^{138}$ | $1.059 \cdot 2^{286}$ | **$1.390 \cdot 2^{151}$** | **$1.987 \cdot 2^{289}$** | $1.798 \cdot 2^{281}$ |
| | ◈ | | 11301 | $1.515 \cdot 2^{147}$ | $1.389 \cdot 2^{138}$ | **$1.052 \cdot 2^{286}$** | $1.916 \cdot 2^{151}$ | $1.330 \cdot 2^{290}$ | $1.274 \cdot 2^{282}$ |
| | ☆ | | 10065 | $1.628 \cdot 2^{148}$ | $1.899 \cdot 2^{138}$ | $1.545 \cdot 2^{287}$ | $1.166 \cdot 2^{152}$ | $1.107 \cdot 2^{291}$ | **$1.267 \cdot 2^{281}$** |
| | ◎ ❀ | | 13665 | $1.625 \cdot 2^{148}$ | $1.367 \cdot 2^{138}$ | $1.110 \cdot 2^{287}$ | $1.140 \cdot 2^{152}$ | $1.558 \cdot 2^{290}$ | $1.720 \cdot 2^{281}$ |
| | ◈ | | 16993 | $1.641 \cdot 2^{148}$ | $1.327 \cdot 2^{138}$ | $1.088 \cdot 2^{287}$ | $1.376 \cdot 2^{152}$ | $1.826 \cdot 2^{290}$ | $1.069 \cdot 2^{282}$ |

☆: Regular version. ❁: S-box with Toffoli depth 4 (low qubit count).
◎: Shallow version. ✿: S-box with Toffoli depth 4 (low full depth).
◈: Shallow/low depth version. ❀: S-box with Toffoli depth 3.

The optimization of cipher building blocks can be considered among the top priorities of future research works. As far as we can tell, there is a vacant niche for a tool that can efficiently find such implementation for $8 \times 8$ S-boxes. Besides, the idea in [DP21] can be used on top of our implementations to further reduce the cost for AES-192 and AES-256 (i.e., when $r > 1$); this is kept as a follow-up work. Similarly, other decomposition of the Toffoli gate (e.g., as in [Sel13]) can also be considered in the future scope.

Other than that, one may also consider implementation of combined components (as one $128 \times 128$ binary matrix), such as; 4 combined MixColumn's ($128 \times 128$ binary matrix), 4 combined MixColumn's with ShiftRow and T-table. At the same time, we expect a follow-up work with a focus on reorganizing the linear operations can be carried out in order to reduce the full depth for a linear layer (binary non-singular matrix) as well as an S-box.

All in all, our paper manages to adopt practically all relevant research works that have been carried out so far in the literature, including the recent works like [HS22, LPZW23, LGQW23, LXX+23, ZH22]. This enables us to pick the suitable candidates for each

**Table 11:** Comparison of NIST security levels based on AES variants.

| Level (AES) | N$^+$'16[NIS16] (G$^+$ [GLRS16]) | L$^+$ [LPS20] | N$^+$'22 [NIS22] | This work | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | ☆ | ◎ | ◈ | ❄ ✳ (Tof) | ❄ ✲ (Tof) | ❄ ✳ (AND) | ❄ ✲ (AND) |
| 1 (128) | $2^{170}$ ($2^{168.6683}$) | $2^{162.6093}$ | $2^{157}$ | ❁: $2^{157.0014}$<br>✿: $2^{156.5753}$<br>❀: $2^{157.8286}$ | ❁: $2^{156.5018}$<br>✿: $2^{156.2762}$<br>❀: $2^{157.3300}$ | ❁: $2^{156.4605}$<br>✿: $2^{156.2630}$<br>❀: $2^{157.2998}$ | ✚: $2^{162.3577}$<br>✛: $2^{162.5641}$ | ✚: $2^{158.1337}$<br>✛: $2^{157.9591}$ | ✚: $2^{161.6042}$<br>✛: $2^{161.6510}$ | ✚: $2^{157.9949}$<br>✛: $2^{157.3327}$ |
| 3 (192) | $2^{233}$ ($2^{233.4645}$) | $2^{227.6491}$ | $2^{221}$ | ❁: $2^{222.3033}$<br>✿: $2^{221.8726}$<br>❀: $2^{223.1296}$ | ❁: $2^{221.8197}$<br>✿: $2^{221.5880}$<br>❀: $2^{222.6508}$ | ❁: $2^{221.7832}$<br>✿: $2^{221.5801}$<br>❀: $2^{222.6201}$ | ✚: $2^{227.5867}$<br>✛: $2^{227.6260}$ | ✚: $2^{223.4821}$<br>✛: $2^{223.3002}$ | ✚: $2^{226.9368}$<br>✛: $2^{226.9885}$ | ✚: $2^{223.4355}$<br>✛: $2^{222.7643}$ |
| 5 (256) | $2^{298}$ ($2^{298.3467}$) | $2^{292.3100}$ | $2^{285}$ | ❁: $2^{286.7999}$<br>✿: $2^{286.3640}$<br>❀: $2^{287.6276}$ | ❁: $2^{286.2079}$<br>✿: $2^{286.0827}$<br>❀: $2^{287.1506}$ | ❁: $2^{286.2690}$<br>✿: $2^{286.0731}$<br>❀: $2^{287.1217}$ | ✚: $2^{292.1520}$<br>✛: $2^{292.1900}$ | ✚: $2^{287.9871}$<br>$2^{287.7966}$ | ✚: $2^{291.5326}$<br>✛: $2^{291.5822}$ | ✚: $2^{287.9595}$<br>✛: $2^{287.2801}$ |

☆: Regular version (using AND gate). ❁: S-box with Toffoli depth 4 (low qubit count).
◎: Shallow version (using AND gate). ✿: S-box with Toffoli depth 4 (low full depth).
◈: Shallow/low depth version (using AND gate). ❀: S-box with Toffoli depth 3.
❄: Bug-fixed JNRV [JNRV20] (using S-box from [BP12]).
✳: Bug-fixed depth. ✲: Bug-fixed qubit count.
✚: In-place MixColumn [JNRV20]. ✛: Maximov's MixColumn [Max19].

building block. On top of that, we focus on finding the optimum architecture, whence we propose/categorize 3 versions. Indeed, the shallow architecture proposed by us was used in [LPZW23, SF24]. We present four new implementations for the AES S-box and one new implementation for the AES MixColumn. From what we can tell, our paper currently holds the best-known quantum attack on the 3 variants of AES. Apart from that, we go into adequate details about the theory (probably for the first time); e.g., on NIST security levels [NIS16, NIS22], the bug and bug-fix of JNRV (Eurocrypt'20) [JNRV20], or the depth/quabit reduction (Appendix D); which we expect to be useful to future researchers. Besides, the official NIST implementation[26] of AES-128 takes AND-depth of 60, but our implementations improve upon that. As it can be seen from Table 1, that we achieve the AND-depth (i.e., Toffoli depth) of 40,

Our paper bases its analysis on a quantum simulator, bypassing hardware constraints such as the nearest-neighbor rule. However, real quantum devices may enforce or relax this rule, affecting algorithm implementation and attack complexity. Beyond qubit connectivity constraints, quantum hardware introduces noise and errors, requiring fault-tolerant techniques like quantum error correction and mitigation. Building on our quantum implementations of AES, a deeper analysis of their physical impact in real-world systems would provide key insights into feasibility and performance

# References

[ADMG+17] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In Roberto Avanzi and Howard Heys, editors, *Selected Areas in Cryptography – SAC 2016*, pages 317–337, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-69453-5_18.

[AMM+13] Matthew Amy, Dmitri Maslov, Michele Mosca, Martin Roetteler, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, Jun 2013. URL: http://dx.doi.org/10.1109/TCAD.2013.2244643, doi:10.1109/tcad.2013.2244643.

[26] Homepage: https://csrc.nist.gov/Projects/circuit-complexity/list-of-circuits. Code: https://github.com/usnistgov/Circuits/blob/master/data/slp/aes/aes128-enc.slp.

[ASAM18] Mishal Almazrooie, Azman Samsudin, Rosni Abdullah, and Kussay N. Mutter. Quantum reversible circuit of AES-128. *Quantum Information Processing*, 17(5):1–30, may 2018. doi:10.1007/s11128-018-1864-3.

[Bak21] Anubhab Baksi. *Classical and Physical Security of Symmetric Key Cryptographic Algorithms*. PhD thesis, School of Computer Science & Engineering, Nanyang Technological University, Singapore, 2021. URL: https://dr.ntu.edu.sg/handle/10356/152003.

[BBHT98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998. doi:10.1002/3527603093.ch10.

[BC17] Debjyoti Bhattacharjee and Anupam Chattopadhyay. Depth-optimal quantum circuit placement for arbitrary topologies, 2017. URL: https://arxiv.org/abs/1703.08540, doi:10.48550/arXiv.1703.08540.

[BCC+24] Anubhab Baksi, Sumanta Chakraborty, Anupam Chattopadhyay, Matthew Chun, SK Hafizul Islam, Kyungbae Jang, Hyunji Kim, Yujin Oh, Soham Roy, Hwajeong Seo, and Siyi Wang. Quantum implementation of linear and non-linear layers. *IEEE International System-on-Chip Conference (SOCC)*, 2024. URL: https://ieeexplore.ieee.org/document/10737862/, doi:10.1109/SOCC62300.2024.10737862.

[BDK+21] Anubhab Baksi, Vishnu Asutosh Dasu, Banashri Karmakar, Anupam Chattopadhyay, and Takanori Isobe. Three input exclusive-or gate support for boyar-peralta's algorithm. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology - INDOCRYPT 2021, Jaipur, India, December 12-15, 2021, Proceedings*, volume 13143 of *Lecture Notes in Computer Science*, pages 141–158. Springer, 2021. doi:10.1007/978-3-030-92518-5_7.

[BFI21] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. Further results on efficient implementations of block cipher linear layers. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 104(1):213–225, 2021. doi:10.1587/transfun.2020CIP0013.

[BJ24] Anubhab Baksi and Kyungbae Jang. *Implementation and Analysis of Ciphers in Quantum Computing*. Springer, 2024. doi:10.1007/978-981-97-0025-7.

[BJS+21] Anubhab Baksi, Kyungbae Jang, Gyeongju Song, Hwajeong Seo, and Zejun Xiang. Quantum implementation and resource estimates for rectangle and knot. *Quantum Information Processing*, 20(12), dec 2021. doi:10.1007/s11128-021-03307-6.

[BKD21] Anubhab Baksi, Banashri Karmakar, and Vishnu Asutosh Dasu. Poster: Optimizing device implementation of linear layers with automated tools. In *Applied Cryptography and Network Security Workshops*, pages 500–504, Cham, 2021. Springer International Publishing. URL: https://ieeexplore.ieee.org/document/10737862/, doi:10.1007/978-3-030-81645-2_30.

[BNPS19] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. Quantum security analysis of AES. *IACR Transactions on Symmetric Cryptology*, 2019(2):55–93, Jun. 2019. URL: https://tosc.iacr.org/index.php/ToSC/article/view/8314, doi:10.13154/tosc.v2019.i2.55-93.

[BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *Experimental Algorithms*, pages 178–189, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-13193-6_16.

[BP12] Joan Boyar and René Peralta. A small depth-16 circuit for the aes s-box. In *IFIP International Information Security Conference*, pages 287–298. Springer, 2012. doi:10.1007/978-3-642-30436-1_24.

[CBC23] Matthew Chun, Anubhab Baksi, and Anupam Chattopadhyay. DORCIS: depth optimized quantum implementation of substitution boxes. *IACR Cryptol. ePrint Arch.*, page 286, 2023. URL: https://eprint.iacr.org/2023/286.

[CS20] Amit Kumar Chauhan and Somitra Kumar Sanadhya. Quantum resource estimates of grover's key search on aria. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 238–258. Springer, 2020. doi:10.1007/978-3-030-66626-2_13.

[Dan17] Marcus Dansarie. *Cryptanalysis of the SoDark family of cipher algorithms*. PhD thesis, Naval Postgraduate School, Dudley Knox Library, 2017. URL: https://calhoun.nps.edu/handle/10945/56118.

[Dan21] Marcus Dansarie. sboxgates: A program for finding low gate count implementations of S-boxes. *Journal of Open Source Software*, 6(62):2946, 2021. doi:10.21105/joss.02946.

[DBSC19] Vishnu Asutosh Dasu, Anubhab Baksi, Sumanta Sarkar, and Anupam Chattopadhyay. LIGHTER-R: optimized reversible circuit implementation for sboxes. In *32nd IEEE International System-on-Chip Conference, SOCC 2019, Singapore, September 3-6, 2019*, pages 260–265, 2019. doi:10.1109/SOCC46988.2019.1570548320.

[DP21] James H. Davenport and Benjamin Pring. Improvements to quantum search techniques for block-ciphers, with applications to aes. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography*, pages 360–384, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-81652-0_14.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. doi:10.1007/978-3-662-04722-4.

[GH19] Emily Grumbling and Mark Horowitz. *Quantum Computing: Progress and Prospects*. The National Academies Press, Washington DC, 2019. URL: https://www.nap.edu/catalog/25196/quantum-computing-progress-and-prospects.

[Gid18] Craig Gidney. Halving the cost of quantum addition. *Quantum*, 2:74, June 2018. doi:10.22331/q-2018-06-18-74.

[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*, pages 29–43, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-29360-8_3.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996. doi:10.1145/237814.237866.

[HB24] Vedad Hadžic and Roderick Bloem. Efficient and composable masked aes s-box designs using optimized inverters. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):656–683, Dec. 2024. URL: https://tches.iacr.org/index.php/TCHES/article/view/11942, doi:10.46586/tches.v2025.i1.656-683.

[HLZ+17] Yong He, Ming-Xing Luo, E Zhang, Hong-Ke Wang, and Xiao-Feng Wang. Decompositions of n-qubit toffoli gates with linear circuit complexity. *International Journal of Theoretical Physics*, 56(7):2350–2361, 2017. doi:10.1007/s10773-017-3389-4.

[HS22] Zhenyu Huang and Siwei Sun. Synthesizing quantum circuits of AES with lower t-depth and less qubits. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part III*, volume 13793 of *Lecture Notes in Computer Science*, pages 614–644. Springer, 2022. doi:10.1007/978-3-031-22969-5_21.

[JBB+23] Kyungbae Jang, Anubhab Baksi, Jakub Breier, Hwajeong Seo, and Anupam Chattopadhyay. Quantum implementation and analysis of default. *Cryptography and Communications*, pages 1–17, 2023. doi:10.1007/s12095-023-00666-y.

[JBK+22a] Kyungbae Jang, Anubhab Baksi, Hyunji Kim, Hwajeong Seo, and Anupam Chattopadhyay. Improved quantum analysis of SPECK and LowMC. In Takanori Isobe and Santanu Sarkar, editors, *Progress in Cryptology - INDOCRYPT 2022, Kolkata, India, December 11-14, 2022, Proceedings*, volume 13774 of *Lecture Notes in Computer Science*, pages 517–540. Springer, 2022. doi:10.1007/978-3-031-22912-1_23.

[JBK+22b] Kyungbae Jang, Anubhab Baksi, Hyunji Kim, Gyeongju Song, Hwajeong Seo, and Anupam Chattopadhyay. Quantum analysis of AES. Cryptology ePrint Archive, Paper 2022/683, 2022. URL: https://eprint.iacr.org/2022/683.

[JBKK24] Yongjin Jeon, Seungjun Baek, Giyoon Kim, and Jongsung Kim. A framework for generating s-box circuits with boyar–peralta algorithm-based heuristics, and its applications to aes, snow3g, and saturnin. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):586–631, Dec. 2024. URL: https://tches.iacr.org/index.php/TCHES/article/view/11940, doi:10.46586/tches.v2025.i1.586-631.

[JCK+20] Kyoungbae Jang, Seungju Choi, Hyeokdong Kwon, Hyunji Kim, Jaehoon Park, and Hwajeong Seo. Grover on Korean block ciphers. *Applied Sciences*, 10(18), 2020. doi:10.3390/app10186407.

[JLO+25] Kyungbae Jang, Sejin Lim, Yujin Oh, Hyunjun Kim, Anubhab Baksi, Sumanta Chakraborty, and Hwajeong Seo. Quantum implementation and analysis of sha-2 and sha-3. *IEEE Transactions on Emerging Topics in Computing*, pages 1–15, 2025. doi:10.1109/TETC.2025.3546648.

[JNRV19] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on aes and lowmc. Cryptology ePrint Archive, Paper 2019/1146, 2019. URL: https://eprint.iacr.org/2019/1146.

[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, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 280–310. Springer, 2020. doi:10.1007/978-3-030-45724-2_10.

[JSB+25] Kyungbae Jang, Vikas Srivastava, Anubhab Baksi, Santanu Sarkar, and Hwajeong Seo. New quantum cryptanalysis of binary elliptic curves (extended version). Cryptology ePrint Archive, Paper 2025/017, 2025. URL: https://eprint.iacr.org/2025/017.

[JSK+21a] Kyungbae Jang, Gyeongju Song, Hyunjun Kim, Hyeokdong Kwon, Hyunji Kim, and Hwajeong Seo. Efficient implementation of PRESENT and GIFT on quantum computers. *Applied Sciences*, 11(11), 2021. doi:10.3390/app11114776.

[JSK+21b] Kyungbae Jang, Gyeongju Song, Hyeokdong Kwon, Siwoo Uhm, Hyunji Kim, Wai-Kong Lee, and Hwajeong Seo. Grover on PIPO. *Electronics*, 10(10):1194, 2021. doi:10.3390/electronics10101194.

[JSK+22] Kyungbae Jang, Gyeongju Song, Hyunjun Kim, Hyeokdong Kwon, Hyunji Kim, and Hwajeong Seo. Parallel quantum addition for korean block ciphers. *Quantum Information Processing*, 21(11):373, 2022. doi:10.1007/s11128-022-03714-3.

[KHJ18] Panjin Kim, Daewan Han, and Kyung Chul Jeong. Time–space complexity of quantum search algorithms in symmetric cryptanalysis: applying to aes and sha-2. *Quantum Information Processing*, 17(12):1–39, 2018. https://doi.org/10.1007/s11128-018-2107-3. doi:10.1007/s11128-018-2107-3.

[KJB+24] Hyunji Kim, Kyungbae Jang, Anubhab Baksi, Sumanta Chakraborty, and Hwajeong Seo. Concrete quantum cryptanalysis of shortest vector problem. Cryptology ePrint Archive, Paper 2024/712, 2024. URL: https://eprint.iacr.org/2024/712.

[KLSW17] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter linear straight-line programs for mds matrices. *IACR Transactions on Symmetric Cryptology*, 2017(4):188–211, Dec. 2017. URL: https://tosc.iacr.org/index.php/ToSC/article/view/813, doi:10.13154/tosc.v2017.i4.188-211.

[LGQW23] Zhenqiang Li, Fei Gao, Sujuan Qin, and Qiaoyan Wen. New record in the number of qubits for a quantum implementation of aes, 2023. doi:10.3389/fphy.2023.1171753.

[LPS20] Brandon Langenberg, Hai Pham, and Rainer Steinwandt. Reducing the cost of implementing the advanced encryption standard as a quantum circuit. *IEEE Transactions on Quantum Engineering*, 1:1–12, 01 2020. doi:10.1109/TQE.2020.2965697.

[LPZW23] Qun Liu, Bart Preneel, Zheng Zhao, and Meiqin Wang. Improved quantum circuits for aes: Reducing the depth and the number of qubits. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 67–98. Springer, 2023. doi:10.1007/978-981-99-8727-6_3.

[LSL+19] Shun Li, Siwei Sun, Chaoyun Li, Zihao Wei, and Lei Hu. Constructing low-latency involutory mds matrices with lightweight circuits. *IACR Transactions on Symmetric Cryptology*, 2019(1):84–117, Mar. 2019. URL: https://tosc.iacr.org/index.php/ToSC/article/view/7398, doi:10.13154/tosc.v2019.i1.84-117.

[LWF+22] Qun Liu, Weijia Wang, Yanhong Fan, Lixuan Wu, Ling Sun, and Meiqin Wang. Towards low-latency implementation of linear layers. *IACR Transactions on Symmetric Cryptology*, 2022(1):158–182, Mar. 2022. URL: https://tosc.iacr.org/index.php/ToSC/article/view/9532, doi:10.46586/tosc.v2022.i1.158-182.

[LWS+22] Qun Liu, Weijia Wang, Ling Sun, Yanhong Fan, Lixuan Wu, and Meiqin Wang. More inputs makes difference: Implementations of linear layers using gates with more than two inputs. *IACR Transactions on Symmetric Cryptology*, 2022(2):351–378, Jun. 2022. URL: https://tosc.iacr.org/index.php/ToSC/article/view/9724, doi:10.46586/tosc.v2022.i2.351-378.

[LXX+23] Da Lin, Zejun Xiang, Runqing Xu, Shasha Zhang, and Xiangyong Zeng. Optimized quantum implementation of aes. *Quantum Information Processing*, 22(9):352, 2023. doi:10.1007/s11128-023-04043-9.

[LXZZ21] Da Lin, Zejun Xiang, Xiangyong Zeng, and Shasha Zhang. A framework to optimize implementations of matrices. In Kenneth G. Paterson, editor, *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12704 of *Lecture Notes in Computer Science*, pages 609–632. Springer, 2021. doi:10.1007/978-3-030-75539-3_25.

[LYLL22] Qing-bin Luo, Guo-wu Yang, Xiao-yu Li, and Qiang Li. Quantum reversible circuits for GF($2^8$)multiplicative inverse. *EPJ Quantum Technology*, 2022. URL: https://link.springer.com/content/pdf/10.1140/epjqt/s40507-022-00144-z.pdf, doi:10.1140/epjqt/s40507-022-00144-z.

[LZW23] Qun Liu, Zheng Zhao, and Meiqin Wang. Improved heuristics for low-latency implementations of linear layers. In *Cryptographers' Track at the RSA Conference*, pages 524–550. Springer, 2023. doi:10.1007/978-3-031-30872-7_20.

[MAR+24] Surajit Mandal, Ravi Anand, Mostafizar Rahman, Santanu Sarkar, and Takanori Isobe. Implementing grover's on aes-based aead schemes. *Scientific Reports*, 14, 09 2024. doi:10.1038/s41598-024-69188-8.

[Max19] Alexander Maximov. AES MixColumn with 92 XOR gates. Cryptology ePrint Archive, Paper 2019/833, 2019. URL: https://eprint.iacr.org/2019/833.

[NIS16] NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016. URL: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[NIS22] NIST. Call for additional digital signature schemes for the post-quantum cryptography standardization process, 2022. URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf.

[OJBS23] Yujin Oh, Kyungbae Jang, Anubhab Baksi, and Hwajeong Seo. Depth-optimized implementation of ASCON quantum circuit. Cryptology ePrint Archive, Paper 2023/1030, 2023. URL: https://eprint.iacr.org/2023/1030.

[OJS25] Yujin Oh, Kyungbae Jang, and Hwajeong Seo. Quantum security evaluation of ASCON. Cryptology ePrint Archive, Paper 2025/260, 2025. URL: https://eprint.iacr.org/2025/260.

[PD24] Meltem Kurt Pehlivanoğlu and Mehmet Ali Demir. Optimizing implementations of linear layers using two and higher input xor gates. *PeerJ Computer Science*, 10:e1820, 2024. doi:10.7717/peerj-cs.1820.

[Per19] Simone Perriello. *Design and development of a quantum circuit to solve the Information Set Decoding problem*. PhD thesis, Politecnico di Milano, Scuola di Ingegneria Industriale e dell'Informazione, 2019. URL: https://hdl.handle.net/10589/147855.

[RBC23] Soham Roy, Anubhab Baksi, and Anupam Chattopadhyay. Quantum implementation of ascon linear layer. NIST Lightweight Cryptography Workshop, 2023. URL: https://csrc.nist.gov/csrc/media/Events/2023/lightweight-cryptography-workshop-2023/documents/accepted-papers/06-quantum-implementation-ascon-linear-layer.pdf.

[Sel13] Peter Selinger. Quantum circuits of t-depth one. *Physical Review A*, 87(4):042302, 2013. doi:10.1103/PhysRevA.87.042302.

[SF24] Haotian Shi and Xiutao Feng. Quantum circuits of aes with a low-depth linear layer and a new structure. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 358–395. Springer, 2024. doi:10.1007/978-981-96-0944-4_12.

[SFX23] Haotian Shi, Xiutao Feng, and Shengyuan Xu. A framework with improved heuristics to optimize low-latency implementations of linear layers. *IACR Transactions on Symmetric Cryptology*, 2023(4):489–510, Dec. 2023. URL: https://tosc.iacr.org/index.php/ToSC/article/view/11297, doi:10.46586/tosc.v2023.i4.489-510.

[SJK+21] Gyeongju Song, Kyungbae Jang, Hyunji Kim, Wai-Kong Lee, Zhi Hu, and Hwajeong Seo. Grover on sm3. In *International Conference on Information Security and Cryptology*, pages 421–433. Springer, 2021. doi:10.1007/978-3-031-08896-4_22.

[TP19] Quan Quan Tan and Thomas Peyrin. Improved heuristics for short linear programs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):203–230, Nov. 2019. URL: https://tches.iacr.org/index.php/TCHES/article/view/8398, doi:10.13154/tches.v2020.i1.203-230.

[WJB+24] Siyi Wang, Kyungbae Jang, Anubhab Baksi, Sumanta Chakraborty, Bryan Lee, Anupam Chattopadhyay, and Hwajeong Seo. New results in quantum analysis of LED: Featuring one and two oracle attacks. Cryptology ePrint

Archive, Paper 2024/1982, 2024. URL: https://eprint.iacr.org/2024/1982.

[WR14] Nathan Wiebe and Martin Roetteler. Quantum arithmetic and numerical analysis using repeat-until-success circuits, 2014. URL: https://arxiv.org/abs/1406.2040, doi:10.48550/arXiv.1406.2040.

[WWL22] Ze-Guo Wang, Shi-Jie Wei, and Gui-Lu Long. A quantum circuit design of aes requiring fewer quantum qubits and gate operations. *Frontiers of Physics*, 17(4):1–7, 2022. doi:10.1007/s11467-021-1141-2.

[XZL+20] Zejun Xiang, Xiangyoung Zeng, Da Lin, Zhenzhen Bao, and Shasha Zhang. Optimizing implementations of linear layers. *IACR Transactions on Symmetric Cryptology*, 2020(2):120–145, Jul. 2020. URL: https://tosc.iacr.org/index.php/ToSC/article/view/8671, doi:10.13154/tosc.v2020.i2.120-145.

[YJBS23] Yujin Yang, Kyungbae Jang, Anubhab Baksi, and Hwajeong Seo. Optimized implementation and analysis of cham in quantum computing. *Applied Sciences*, 13(8), 2023. doi:10.3390/app13085156.

[YWS+24] Yufei Yuan, Wenling Wu, Tairong Shi, Lei Zhang, and Yu Zhang. A framework to improve the implementations of linear layers. *IACR Transactions on Symmetric Cryptology*, 2024(2):322–347, Jun. 2024. URL: https://tosc.iacr.org/index.php/ToSC/article/view/11633, doi:10.46586/tosc.v2024.i2.322-347.

[Zal99] Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746, 1999. https://doi.org/10.1103/physreva.60.2746. doi:10.1103/PhysRevA.60.2746.

[ZH22] Chengkai Zhu and Zhenyu Huang. Optimizing the depth of quantum implementations of linear layers. In *Inscrypt 2022, Beijing, China, December 11-13, 2022*, volume 13837 of *Lecture Notes in Computer Science*, pages 129–147. Springer, 2022. doi:10.1007/978-3-031-26553-2_7.

[ZLD+19] Jian Zou, Yongyang Liu, Chen Dong, Wenling Wu, and Le Dong. Observations on the quantum circuit of the SBox of AES. Cryptology ePrint Archive, Paper 2019/1245, 2019. URL: https://eprint.iacr.org/2019/1245.

[ZLW+22] Jian Zou, Liji Li, Zihao Wei, Yiyuan Luo, Qian Liu, and Wenling Wu. New quantum circuit implementations of sm4 and sm3. *Quantum Information Processing*, 21(5):1–38, 2022. doi:10.1007/s11128-022-03518-5.

[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, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-64834-3_24.

# A Concise Description of AES Variants

The Advanced Encryption Standard (AES) [DR02] is an SPN block cipher family with a block of 128 bits. The state of AES is arranged as a $4 \times 4$ matrix of bytes. AES contains three specific variants denoted as AES-128, AES-192 and AES-256 according to the key size. Schematic diagrams of AES-128 round function and key schedule can be seen in Figure 8.

## A.1 Round Function

The round function of AES consists of AddRoundKey ∘ MixColumns ∘ ShiftRows ∘ SubBytes, except for the last round which misses the MixColumns operation.

#### SubBytes

This operation substitutes each element by a predefined $8 \times 8$ S-box.

#### ShiftRows

This operation cyclically rotates the $r^{\text{th}}$ row of state to the left by $i$ places; for $i = 0, 1, 2, 3$.

#### MixColumns

The MixColumn operation pre-multiplies each of the state column with the right circulant matrix $(\texttt{02}, \texttt{03}, \texttt{01}, \texttt{01})$, over $\mathrm{GF}(2^8)[x]$ with modulus $x^8 + x^4 + x^3 + x + 1$. Since the MixColumn operates on the state based on an entire column, it can also be represented as a matrix over $\mathbb{F}_2$ with dimension $32 \times 32$ (one may refer to [Bak21, Chapter 2.4.1] for a representation as a binary matrix).

#### AddRoundKey

The sub-key of each round is generated by the Key Expansion algorithm. Each call of AddRoundKey XORs the 128-bit sub-key to the state.

The encryption procedure for different instances of AES family are somewhat similar, except the number of round varies. For AES-128, AES-192 and AES-256, the round numbers are 10, 12, 14 respectively and all round functions are identical except that there is no MixColumns operation in the last round. Note that there is an extra key addition before the first round (also known as whitening).

## A.2 Key Schedule

Similar to the state, the master key of AES is allocated to a $4 \times l$ grid of byte in order, where $l = 4, 6$ or 8 for AES-128, AES-192 and AES-256, respectively. Generally, the generation of the round sub-keys are based on *word* (the entire column in the grid) with the operations RotWord (cyclically rotating the bytes in a word to the left by one byte), SubWord (operating the SubBytes of round function on each bytes in a word) and the XOR of Rcon$[r]$ (the $r^{\text{th}}$ 32-bit round constant).

The master key is loaded to the grid $W_0, W_1, \cdots, W_i$; where $i$ is 3, 5 and 7 for AES-128, AES-192 and AES-256 respectively. In order to guarantee the encryption, 40, 46 and 52 words need to be provided by key expansion for those three AES instances, respectively.

For AES-128, the word $W_i$ is generated by

$$W_i = \begin{cases} W_{i-4} \oplus \mathrm{SubWord}(\mathrm{RotWord}(W_{i-1})) \oplus \mathrm{Rcon}[i/4], & \text{if } i \equiv 0 \bmod 4, \\ W_{i-4} \oplus W_{i-1}, & \text{otherwise,} \end{cases}$$

where $i = 4, 5, \cdots, 43$.

For AES-192, the word $W_i$ is generated by

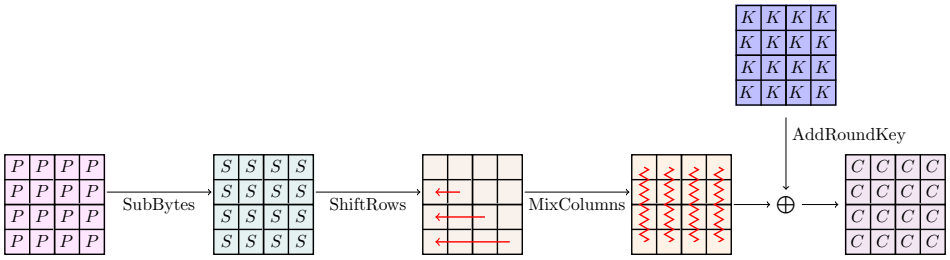
$$W_i = \begin{cases} W_{i-6} \oplus \mathrm{SubWord}(\mathrm{RotWord}(W_{i-1})) \oplus \mathrm{Rcon}[i/6], & \text{if } i \equiv 0 \bmod 6, \\ W_{i-6} \oplus W_{i-1}, & \text{otherwise,} \end{cases}$$

where $i = 6, 7, \cdots, 51$.

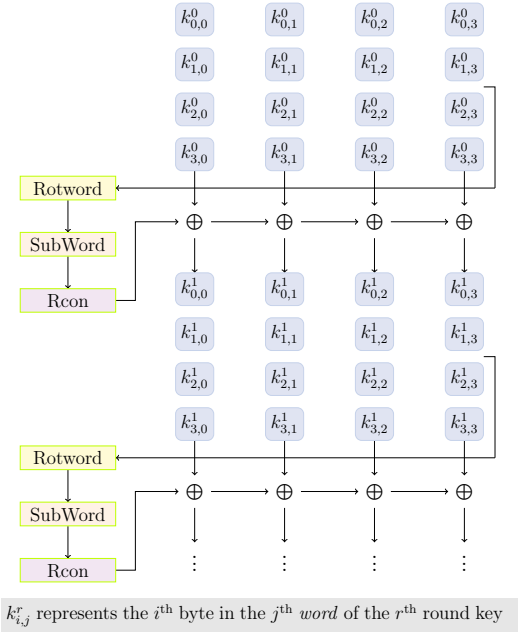
For AES-256, the word $W_i$ is generated by

$$W_i = \begin{cases} W_{i-8} \oplus \text{SubWord}(\text{RotWord}(W_{i-1})) \oplus \text{Rcon}[i/8], & \text{if } i \equiv 0 \bmod 8, \\ W_{i-8} \oplus \text{SubWord}(W_{i-1}), & \text{if } i \equiv 4 \bmod 8, \\ W_{i-8} \oplus W_{i-1}, & \text{otherwise,} \end{cases}$$

where $i = 8, 9, \cdots, 59$.




**(a)** Round function (except last round which skips MixColumns).




**(b)** Key schedule.

**Figure 8:** Schematic of AES encryption.

## A.3 Notations Related to Singular and Plural Forms

The AES state is represented as a $4 \times 4$ matrix and the operation on one column of the matrix is denoted here as MixColumn. As described earlier, MixColumn corresponds to a matrix multiplication over $\text{GF}(2^8)$, which can equivalently be expressed as multiplication by a matrix of dimension $32 \times 32$ over $\mathbb{F}_2$. In the AES round function, the MixColumns operates on the whole block by applying MixColumn to every four bytes in the state (i.e., one column in the $4 \times 4$ matrix). Thus, one MixColumns operation is equivalent to $4\times$ MixColumn operations on different columns in the matrix. Denoting the binary matrix corresponding to MixColumn as $M$ with size $32 \times 32$, MixColumns can be represented as the diagonal matrix $(M, M, M, M)$ of dimension $128 \times 128$ over $\mathbb{F}_2$.

The bytes in each row of the matrix will be cyclically shifted to the left in each round and the shift operation on the bytes in one row is denoted here as ShiftRow, in the step of ShiftRows, the ShiftRow will be operated on all the rows in the matrix and shift the bytes in the $i^{\text{th}}$ row to the left by $i$ bytes, where $i = 1, 2, 3$. Thus, one ShiftRows operation is equivalent to 4× ShiftRow operations on different rows in the $4 \times 4$ matrix with the shift parameter varies from 0 to 3.

The SubBytes in the round function updates every byte in the $4 \times 4$ matrix in the same way. The process of applying the S-box to one byte in the AES state is denoted here as SubByte. In each round, the SubBytes updates all the bytes in the $4 \times 4$ matrix by replacing each byte by another one according to the predefined nonlinear map. Thus, one SubBytes operation is equivalent to 16 SubByte operations on the bytes of the $4 \times 4$ matrix.

### A.4 Notations Related to Reverse Computation

#### S-box and S-box$^\dagger$ in Quantum

S-box in quantum denotes before storing values from ancilla qubits to output qubits. We denote the reverse operation of S-box as S-box$^\dagger$ and uses input qubits to clean up ancilla qubits.

#### SubBytes and SubBytes$^\dagger$ in Quantum

SubBytes of AES in quantum denotes parallel operation for 16 S-boxes. We denote the reverse operation of SubBytes as SubBytes$^\dagger$ and cleans up all used ancilla qubits in 16 S-boxes.

#### Rotation and Rotation$^\dagger$ in Quantum

Rotation of AES in quantum denotes the same RotWord. The reverse operation of Rotation is denoted as Rotation$^\dagger$.

#### SubWord and SubWord$^\dagger$ in Quantum

SubWord of AES in quantum denotes parallel operation for 4 S-boxes. We denote the reverse operation of SubWord as SubWord$^\dagger$ (and clean up all used ancilla qubits in 4 S-boxes).

## B New Construction and Novelty

We basically collect practically all the relevant research works and collate in one place. Our coverage of the literature includes papers as recent as [ZH22, LXX+23, LGQW23, LPZW23, SF24]. On top of that, we would like to note the following points about the novelty/new building blocks introduced in this paper:

1. We propose three four implementations of the S-box (see Table 3 for a summary, and Appendix D for details) that reduce the full depth and/or ancilla qubits from that of [HS22, LPZW23]. Indeed, we currently hold the record for the least full depth implementation of the AES S-box (56, reduced from 69 as reported in [HS22]), to the best of our knowledge. Of the four new S-box implementations, three are used in this work.

2. We propose our new out-of-place implementation of MixColumn that takes only 8 quantum depth (the idea is described in Appendix D), and use it in our shallow/low depth implementation. From what we find, this is the least quantum depth for the AES MixColumn implementation reported so far. Further, the required number of ancilla qubits is only 32.

3. We optimize the depth of the components by using more ancilla sets (except in-place MixColumns) through parallelization. We reduce the depth while conserving the number of qubits by allowing for many ancilla qubits and reusing them in the next round through reverse operations.

4. We present a new idea for pipelining of operation (Figure 7(b)), which reduces the T-depth and full depth from the previous works (as in Figure 7(a)). This involves combining the previous round's reverse operation with the current round's operation by using two alternate ancilla sets.

5. We present the last-round optimization technique (Figure 7(c)), which replaces the output qubits in the final round with ancilla qubits used in SubBytes. This further reduces the 128 output qubits required for the final round in both the shallow and shallow/low depth versions.

6. We propose five new architecture in this paper:

   (a) The regular architecture was originally conceived by Jaques et al. in Eurocrypt'20 [JNRV20]. However, their proposal contained bug (related to non-linear operations). We analysis the bug in detail (in Section 5 and Appendix C) and present the corrected regular version after patching the bug.

   (b) The shallow and shallow/low depth architecture are our innovation and proposed for the first time in literature in this work (note that the authors of Asiacrypt'23 [LPZW23] adopted our shallow architecture). The shallow/low depth version has the advantage that the ancilla qubits for MixColumn can be taken for free (in the regular version, used in [JNRV20], ancilla qubits are not free when the Q# bug is patched). Similarly, for SubBytes$^\dagger$ in the shallow and shallow/low-depth architecture, many of the ancilla qubits can be saved by taking the idle ancilla qubits in SubBytes (as presented in [LPZW23]).

   (c) The bug-fixing of [JNRV20] involves two more architecture (see Section 5.2 as well as Appendix C for more details), which we call fixed depth and fixed qubit versions. Our bug-fixed benchmark of [JNRV20] improves from the authors' own bug-fixed benchmark presented recently in [JNRV19].

We present 78 implementations for the AES variants in this work:

(i) **Ours:** 3 architecture × 3 S-box implementations × 3 AES variants × 2 gates (Toffoli and AND).

(ii) **Bug-fixed JNRV (Eurocrypt'20):** 3 AES variants × 2 MixColumn implementations × 2 architecture (fixed depth and fixed qubit count ) × 2 gates (Toffoli and AND).

A bird's-eye view can be seen from Figure 9, where we show how our work contributes in lowering the quantum circuit complexity (in terms of qubit count and full depth) compared to GLRS [GLRS16] and LPS [LPS20].

The concepts presented in our work (which has been publicly available as [JBK+22b]) has been used in many other works such as [JBK+22a, MAR+24, OJBS23, JLO+25, JSB+25, KJB+24, WJB+24]; not to mention two recent works in Asiacrypt'23 [LPZW23]

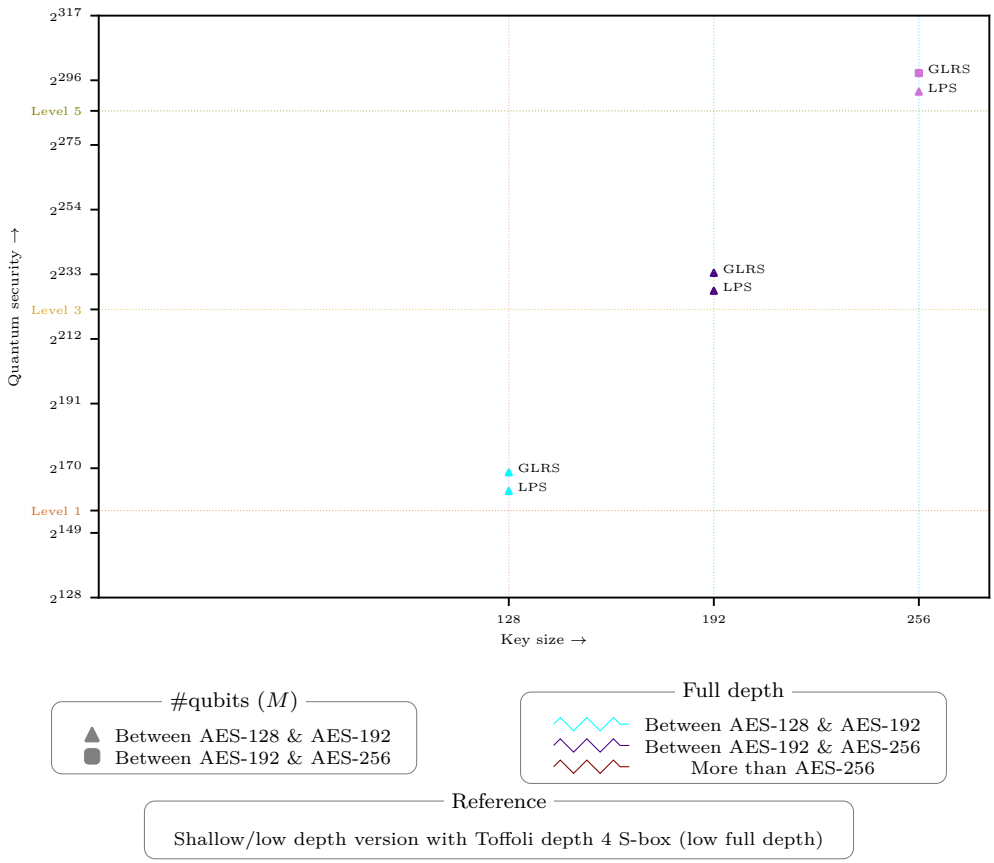



**Figure 9:** Comparison of quantum circuit complexities for AES variants.

and Asiacrypt'24 [SF24] were (partially) inspired from earlier versions of our work (this was acknowledged by the authors in the respective papers). Apart from being a survey/systematization-of-knowledge, our paper covers in-depth discussion prerequisite topics. We wrap-up with the comment by the first author of [LPZW23] made in a private email communication:

> "I would like to express our gratitude for your paper... In this paper, we gained valuable insights into the new AES construction structure. ... Thank you once again for your valuable contributions to the field."

# C Discussion about Q# Bug in JNRV (Eurocrypt'20)

Continuing from Section 5, we detail more about the Q# bug which affected the Eurocrypt'20 implementation [JNRV20]. We encountered two issues. First (non-parallelizable) and second (issue with AND gate) problems analyzed in Section C.1 can be solved by adjusting the number of qubits. If many ancilla qubits are used, over-parallelized depth may be possible. However, the third problem (inconsistency and underestimation of full depth) in that Section 5.1.1 cannot be solved that way. A well-observed case of this error is the depth of AES-256 using in-place MC reported in JNRV [JNRV20]. Only 234 should be derived as depth for SubBytes $\times$ 14 rounds. This depth margin, therefore, cannot be derived even with excessive parallelization.

The `using` command automatically disposes when the function ends. If ancilla qubits to implement AES S-box are allocated with the `using` command, the consistency between depth and qubits is lost. When 16 S-boxes are executed in SubBytes, the ancilla qubits allocated by the `using` are counted only for the first S-box and not after. Also counts the depth for executing 16 S-boxes simultaneously. In order to derive the correct result, the number of qubits or depth must be increased. Q#'s `ResourcesEstimator` tries to find its own lower bound for depth and qubit. That is, to achieve the qubits of the lower bound, the depth may have to be increased, and to achieve the depth of the lower bound, the qubits may have to be increased.

Another problem is inconsistencies between quantum resources. We observe underestimation when cross-checking the full depth of oracles, S-box and MixColumn they report. We could not pinpoint the exact cause, but we suspect the problems were caused by the `using` command and the AND gate. As noted, these problems effectively construct quantum circuits that are impossible.

The Q# compiler finds non-trivial parallelism in the circuit, but according to our examples, this parallelism is excessive in the Eurocrypt'20 paper [JNRV20]. In our case also, the estimated depth of the circuit is slightly reduced, rather than being exactly equal to the product of the round number and the depth (which would indicate trivial parallelism). MixColumns requires the result from SubBytes (i.e., it operates sequentially like this: SubBytes $\rightarrow$ MixColumns $\rightarrow$ SubBytes $\rightarrow$ MixColumns), so it cannot be estimated in parallel. There is a small degree of overlap between the MixColumn operation in the current round and the SubBytes operation in the following round. However, as demonstrated by our example, this overlap is excessive. The reported depth still seems impossible because the depth of each round has to be counted independently (only slight reduction possible with trivial parallelization).

A well-observed case of this error is the depth of AES-256 using in-place MixColumn reported in [JNRV20]. The full depth of their AES-256 (in-place MixColumn) oracle is 3353. Then about 1677 (half) would be the full depth of the AES-256 circuit. However, the full depth of the in-place MixColumn is 111, so 13 rounds (excluding the last round) $\times$ 111, the full depth is already 1443. Then only 234 ($= 1677 - 1443$) should be derived as depth for SubBytes $\times$ 14 rounds. Therefore, the full depth derived from Sbox in each round should be only about 17 ($= 234 \div 14$), which cannot be derived even with excessive parallelization or omitting cleaning of ancilla qubits.

Additionally, if the full depth is estimated assuming all parallelization with bugs, the full depth for the AES variants should depend on the number of rounds. However, the full depth of AES-192, -256 (Maximov's MixColumn [Max19]) reported in JNRV [JNRV20] is even lower than AES-128. The lower depth of AES-192 is due to fewer key schedules (corresponding to the zig-zag structure). However, if complete parallelism is assumed, depth should depend on the number of rounds, since key schedule works in parallel with rounds (like ours).

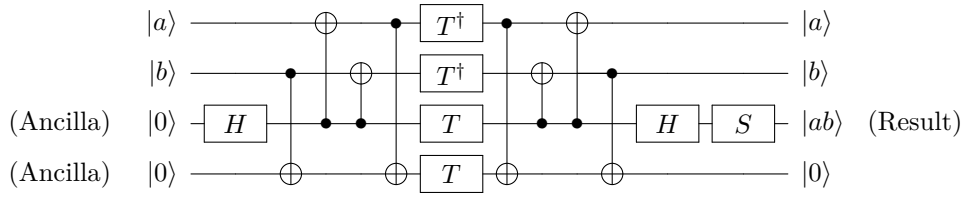
## C.1 Non-parallelizable SubBytes

In the Eurocrypt'20 implementation, the S-box of [BP10] is adopted and ported to the corresponding quantum circuit. The quantum resources required for the S-box quantum circuit reported in this paper [JNRV20, Table 1] are only correct for the stand-alone S-box (except for T-depth, this is described in Section C.2). However, in the case of SubBytes operating with 16 S-boxes, incorrect quantum resources are reported. This is a major cause of their resource estimation issues.

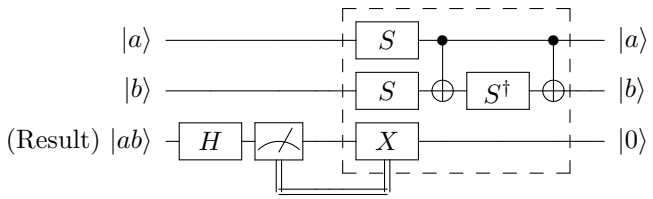
According to the reported number of required qubits, only one ancilla set is used in their SubBytes implementation. In other words, 16 S-boxes share one ancilla set. Thus, the arrangement of qubits in their SubBytes quantum circuit is the serial structure of Figure 4(b). Since 16 S-boxes generate each output using one ancilla set, all S-boxes in a

limited space (one ancilla set) must be operated sequentially. However, in their report, the depth of the SubBytes is the same as the depth for a standalone S-box (meaning all S-boxes operate in parallel). That is, it is an impossible quantum circuit structure and the lower-bound depth is reported. The same error is seen again in the SubWord sub-routine of the key schedule.

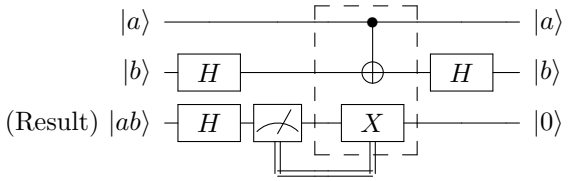
## C.2 Issue with AND Gate




**(a)** AND gate (Eurocrypt'20).




**(b)** AND† gate (Eurocrypt'20).




**(c)** AND† gate (ePrint'23 update of Eurocrypt'20).

**Figure 10:** Quantum AND and AND† gates (JNRV).

This issue was also found in their usage of AND gates. Suppose that 5 Toffoli gates are operated in parallel during the S-box process. Toffoli gates (the method used in [AMM+13]) operate in parallel without any additional work, providing one Toffoli depth and full depth for one Toffoli gate. On the other hand, in the AND gate of Figure 10(a), an ancilla qubit (bottom line in Figure 10(a)) is used. Thus, if replaced with AND gates, 5 ancilla qubits for 5 AND gates must be allocated for parallel operation. Note that, the ancilla qubit of the AND gate is initialized to 0 after operation and can be reused in the next AND gate, but a sequential operation is forced.

In summary, in their S-box (out of 137 qubits, 136 qubits for the S-box and 1 qubit for the AND gate application), only one ancilla qubit is used for one AND gate. However, quantum resources for parallel operations are reported. Technically speaking, the ancilla qubits required for the AND gates can be replaced with idle state qubits in the S-box operation, but this was not considered in their implementation. In our bug-fixed versions, this technique (utilizing idle state qubits) is applied.

In [JNRV19] (ePrint'23 update), the Eurocrypt'20 authors themselves fixed the [JNRV20] (Eurocrypt'20) bug, and introduced a more efficient AND† gate (Figure 10(b)). However, in this update, the AND† gate from [JNRV20] (Figure 10(c)) itself was used to fix the bug of [JNRV20] (Eurocrypt'20).

# D Further Improvement of Quantum Circuits

## D.1 Full Depth Reduction of S-box and MixColumn

Quantum programming tools, in general, attempt to synchronize quantum gates to produce optimized results. In our view, most quantum tools excel in achieving synchronization with Toffoli gates. However, we observe that the tools do not always find the optimal depth for the linear quantum gates (CNOT and X gates). Therefore, we deduce that, in order to achieve the least quantum depth, it may be necessary to directly manipulate the sequence of the linear quantum gates for more efficient parallelization. The concept of reordering to reduce depth itself is not new, e.g., a deterministic greedy method was employed in [ZH22] to reduce the quantum depth of AES MixColumn (which is linear) to 28.

In this case, our target is an S-box (which contains non-linear operations), still we show how it is possible to reduce the full depth by reordering the linear operations only. As a result, we propose a reordering method that employs a randomized approach to optimize the quantum circuit of S-box.

In short, we search all possible linear gate operations that can be reordered (so that the output remains unchanged) and then reorder those gate operations (by picking a sequence randomly) which achieves the lowest full depth. Thanks to this approach, we reduce the depth step by step through multiple reordering of gates.

The overall process is explained through a toy example in Code 1 in ProjectQ compatible format (we also conduct the same experiment with IBM's Qiskit with consistent result). In this example, we want to implement the affine (vectorial) Boolean function $f(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}) = x_0, x_1, x_2, x_3, x_0 \oplus x_1 \oplus x_2 \oplus x_4, x_0 \oplus x_1 \oplus x_2 \oplus x_5, x_0 \oplus x_1 \oplus x_2 \oplus x_6 \oplus 1, x_0 \oplus x_1 \oplus x_2 \oplus x_7, x_2 \oplus x_8 \oplus 1, x_9, x_{10} \oplus 1, x_0 \oplus x_3 \oplus x_{11} \oplus 1$. In input qubits are stored in array $a = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11})$ and the output is stored in array $a$ by in-place operations. This is similar to the way the Asiacrypt'23 [LPZW23] implementation of the S-box is given in [LPZW23, Table 2] (where they had three types of qubits; $u_{0\sim7}$ as the input qubits, $q_{0\sim73}$ as the ancilla qubits, and $s_{0\sim7}$ as the output qubits), except we do not have any ancilla qubit and output qubit in this example.

As it can be seen, Code 1(a) $\rightsquigarrow$ Code 1(b) $\rightsquigarrow$ Code 1(c) $\rightsquigarrow$ Code 1(d) show the progression of depth reduction ($9 \rightsquigarrow 8 \rightsquigarrow 7 \rightsquigarrow 6$) with our approach. CNOT gates and X gates on qubit arrays $a$ and $b$ are operated in Code 1(a). Since there are only CNOT $(a, b)$and X $(a)$ operations, which are of the form ($a = a$, $b = b \oplus a$; $a = a \oplus 1$), all gate CNOT and X operations can be reordered among themselves (including X gates on qubit array $a$). The initial full depth of Code 1(a) is 9, but after three rounds of reordering, the final full depth is reduced to 6 (Code 1(d)).

For the first reordering, we move Lines 3 and 4 (Code 1(a)) to Lines 19 and 20 (Code 1(b)). This does not change the overall output, but the depth is reduced by 1. From this, we guess if same operand (`a[0]` in this case) called continuously, it is operated in sequential even though this gate operations can be operated parallel with subsequent (follow-up) gate operations. However, with our reordering, "`CNOT | (a[0], a[11])`" is synchronized with subsequent gate operations, thanks to being pushed back (resulting in a reduction of depth by 1).

In the second reordering, the depth is reduced by 1 by moving the X gate operations (Lines 22, 23, 24 and 25 in Code 1(b)) to the top (see Code 1(c); 3, 4, 5 and 6).

In the third reordering step, similar to the first reordering (Code 1(b)), Line 21 ("`CNOT | (a[2], a[8])`"), successive calls to `a[2]`) is moved to Line 12 (Code 1(d)). As a result, "`CNOT | (a[2], a[8])`" is synchronized well with the preceding gate operations (i.e., Lines 8, 9, 10 and 11 in Code 1(d)).

With three rounds of randomized reordering, the full depth, which was initially 9, is finally reduced to 6. With this randomized greedy approach, we reduced the full depth of

**Code 1:** Full depth reduction through reordering (toy example)

**(a)** Initial target (9 depth)

```
def Reorder_0 (a): # Full depth : 9

    CNOT | (a[0], a[11])
    CNOT | (a[3], a[11])

    CNOT | (a[0], a[4])
    CNOT | (a[0], a[5])
    CNOT | (a[0], a[6])
    CNOT | (a[0], a[7])

    CNOT | (a[1], a[4])
    CNOT | (a[1], a[5])
    CNOT | (a[1], a[6])
    CNOT | (a[1], a[7])

    CNOT | (a[2], a[4])
    CNOT | (a[2], a[5])
    CNOT | (a[2], a[6])
    CNOT | (a[2], a[7])
    CNOT | (a[2], a[8])

    X | a[6]
    X | a[8]
    X | a[10]
    X | a[11]
```

**(b)** First reordering (8 depth)

```
def Reorder_1 (a): # Full depth : 8

    CNOT | (a[0], a[4])
    CNOT | (a[0], a[5])
    CNOT | (a[0], a[6])
    CNOT | (a[0], a[7])

    CNOT | (a[1], a[4])
    CNOT | (a[1], a[5])
    CNOT | (a[1], a[6])
    CNOT | (a[1], a[7])

    CNOT | (a[2], a[4])
    CNOT | (a[2], a[5])
    CNOT | (a[2], a[6])
    CNOT | (a[2], a[7])
    CNOT | (a[2], a[8])

    CNOT | (a[0], a[11]) # Reordered 1
    CNOT | (a[3], a[11]) # Reordered 1

    X | a[6]
    X | a[8]
    X | a[10]
    X | a[11]
```

**(c)** Second reordering (7 depth)

```
def Reorder_2 (a): # Full depth : 7

    X | a[6] # Reordered 2
    X | a[8] # Reordered 2
    X | a[10] # Reordered 2
    X | a[11] # Reordered 2

    CNOT | (a[0], a[4])
    CNOT | (a[0], a[5])
    CNOT | (a[0], a[6])
    CNOT | (a[0], a[7])

    CNOT | (a[1], a[4])
    CNOT | (a[1], a[5])
    CNOT | (a[1], a[6])
    CNOT | (a[1], a[7])

    CNOT | (a[2], a[4])
    CNOT | (a[2], a[5])
    CNOT | (a[2], a[6])
    CNOT | (a[2], a[8])
    CNOT | (a[2], a[7])

    CNOT | (a[0], a[11]) # Reordered 1
    CNOT | (a[3], a[11]) # Reordered 1
```

**(d)** Third reordering (6 depth)

```
def Reorder_3 (a): # Full depth : 6

    X | a[6]  # Reordered 2
    X | a[8]  # Reordered 2
    X | a[10]  # Reordered 2
    X | a[11]  # Reordered 2

    CNOT | (a[0], a[4])
    CNOT | (a[0], a[5])
    CNOT | (a[0], a[6])
    CNOT | (a[0], a[7])
    CNOT | (a[2], a[8]) # Reordered 3

    CNOT | (a[1], a[4])
    CNOT | (a[1], a[5])
    CNOT | (a[1], a[6])
    CNOT | (a[1], a[7])

    CNOT | (a[2], a[4])
    CNOT | (a[2], a[5])
    CNOT | (a[2], a[6])
    CNOT | (a[2], a[7])

    CNOT | (a[0], a[11])  # Reordered 1
    CNOT | (a[3], a[11])  # Reordered 1
```

S-boxes as presented in [HS22] by reordering partial linear operations within S-box using our approach.

Overall, our idea is generic, it can be applied to any linear layer as well as linear part of an S-box implementation. We apply this method to the out-of-place MixColumn from [LSL+19] and achieve the depth reduction ($11 \rightsquigarrow 8$).

### D.2 Ancilla Qubit Reduction of S-box and MixColumn

We focus on identifying skippable operations when classical implementations are ported to quantum. These operations may be necessary in the classical domain but incur unnecessary overhead in the quantum domain. In particular, in classical implementations, many intermediate values are stored in new variables before being passed to other variables. However, some of the storing steps (storing values in new variables) can be skipped if they do not change the result or do not increase the circuit depth.

With this insight, by altering the computation flow, we eliminate specific operations using new variable in MixColumn and S-box implementations. In simple terms, we eliminate all possible temporary variables that can be skipped. For a simple example, we change XOR ($a[0]$, $temp$) $\rightarrow$ XOR ($temp, y[0]$) directly to XOR ($a[0]$, $y[0]$), resulting in the removal of the $temp$ value. That is, we no longer use a new variable (i.e., $temp$), and two gate operations are reduced to one (in this example).

Thanks to this customization, we could reduce the number of ancilla qubits corresponding to the removed variables. Additionally, the number of CNOT gates is reduced because we save XOR costs by avoiding the need to store values in new variables.

## E Further Results

Similar to [ZWS+20, Table 6], we present the per-round benchmarks for our implementations of the AES family in Table 12, using the S-box implementation with Toffoli depth 4 (low qubit count), Toffoli depth 4 (low full depth) and Toffoli depth 3 in Tables 12(a), 12(b) and 12(c); respectively.

**Table 12:** Quantum resources required per round for variants of AES (this work).
**(a)** Using S-box with Toffoli depth 4 (low qubit count).

| AES | Round | #CNOT ☆ | #CNOT ◎ | #CNOT ◈ | #NOT ☆◎◈ | #Toffoli ☆ | #Toffoli ◎◈ | $TD$ ☆ | $TD$ ◎◈ |
|---|---|---|---|---|---|---|---|---|---|
| 128 | 1≀ | 7836 | 4720 | 5132 | 81 | 1360 | 680 | 8 | 4 |
| | 2 | 7708 | 8276 | 8688 | 81 | 1360 | 1360 | 8 | 4 |
| | 3 | 7708 | 8276 | 8688 | 81 | 1360 | 1360 | 8 | 4 |
| | 4 | 7708 | 8276 | 8688 | 81 | 1360 | 1360 | 8 | 4 |
| | 5 | 7708 | 8276 | 8688 | 81 | 1360 | 1360 | 8 | 4 |
| | 6 | 7708 | 8276 | 8688 | 81 | 1360 | 1360 | 8 | 4 |
| | 7 | 7708 | 8276 | 8688 | 81 | 1360 | 1360 | 8 | 4 |
| | 8 | 7708 | 8276 | 8688 | 81 | 1360 | 1360 | 8 | 4 |
| | 9 | 7708 | 8276 | 8688 | 84 | 1360 | 1360 | 8 | 4 |
| | 10 | 4068 | 7192 | 7192 | 84 | 680 | 1264 | 4 | 4 |
| 192 | 1≀ | 7900 | 8180 | 8592 | 81 | 1360 | 1360 | 8 | 4 |
| | 2 | 7772 | 8212 | 8624 | 81 | 1360 | 1360 | 8 | 4 |
| | 3 | 6188 | 6508 | 6920 | 64 | 1088 | 1088 | 8 | 4 |
| | 4 | 7772 | 8180 | 8592 | 81 | 1360 | 1360 | 8 | 4 |
| | 5 | 7772 | 8212 | 8624 | 81 | 1360 | 1360 | 8 | 4 |
| | 6 | 6188 | 6508 | 6920 | 64 | 1088 | 1088 | 8 | 4 |
| | 7 | 7772 | 8180 | 8592 | 81 | 1360 | 1360 | 8 | 4 |
| | 8 | 7772 | 8212 | 8624 | 64 | 1360 | 1360 | 8 | 4 |
| | 9 | 6188 | 6508 | 6920 | 64 | 1088 | 1088 | 8 | 4 |
| | 10 | 7772 | 8180 | 8592 | 81 | 1360 | 1360 | 8 | 4 |
| | 11 | 7152 | 7620 | 8032 | 81 | 1224 | 1264 | 8 | 4 |
| | 12 | 3296 | 3296 | 3296 | 64 | 544 | 544 | 4 | 4 |
| 256 | 1≀ | 6316 | 3820 | 4232 | 64 | 1088 | 544 | 8 | 4 |
| | 2 | 7708 | 7344 | 7756 | 81 | 1360 | 1224 | 8 | 4 |
| | 3 | 7708 | 8076 | 8488 | 80 | 1360 | 1360 | 8 | 4 |
| | 4 | 7708 | 8084 | 8496 | 81 | 1360 | 1360 | 8 | 4 |
| | 5 | 7708 | 8076 | 8488 | 80 | 1360 | 1360 | 8 | 4 |
| | 6 | 7708 | 8084 | 8496 | 81 | 1360 | 1360 | 8 | 4 |
| | 7 | 7708 | 8076 | 8488 | 80 | 1360 | 1360 | 8 | 4 |
| | 8 | 7708 | 8084 | 8496 | 81 | 1360 | 1360 | 8 | 4 |
| | 9 | 7708 | 8076 | 8488 | 80 | 1360 | 1360 | 8 | 4 |
| | 10 | 7708 | 8084 | 8496 | 81 | 1360 | 1360 | 8 | 4 |
| | 11 | 7708 | 8076 | 8488 | 80 | 1360 | 1360 | 8 | 4 |
| | 12 | 7708 | 8084 | 8496 | 81 | 1360 | 1360 | 8 | 4 |
| | 13 | 7708 | 8076 | 8488 | 80 | 1360 | 1360 | 8 | 4 |
| | 14 | 4068 | 7000 | 7000 | 81 | 680 | 1264 | 4 | 4 |

≀: Including initial key XOR.

☆: Regular version.
◎: Shallow version.
◈: Shallow/low depth version.

**(b)** Using S-box with Toffoli depth 4 (low full depth).

| AES | | #CNOT | | | #NOT | #Toffoli | | $TD$ | |
|---|---|---|---|---|---|---|---|---|---|
| | Round | ☆ | ◎ | ◈ | ☆◎◈ | ☆ | ◎◈ | ☆ | ◎◈ |
| 128 | 1≀ | 6696 | 4160 | 4572 | 81 | 1320 | 680 | 8 | 4 |
| | 2 | 6568 | 6848 | 7260 | 81 | 1320 | 1320 | 8 | 4 |
| | 3 | 6568 | 6848 | 7260 | 81 | 1320 | 1320 | 8 | 4 |
| | 4 | 6568 | 6848 | 7260 | 81 | 1320 | 1320 | 8 | 4 |
| | 5 | 6568 | 6848 | 7260 | 81 | 1320 | 1320 | 8 | 4 |
| | 6 | 6568 | 6848 | 7260 | 81 | 1320 | 1320 | 8 | 4 |
| | 7 | 6568 | 6848 | 7260 | 81 | 1320 | 1320 | 8 | 4 |
| | 8 | 6568 | 6848 | 7260 | 81 | 1320 | 1320 | 8 | 4 |
| | 9 | 6568 | 6848 | 7260 | 84 | 1320 | 1320 | 8 | 4 |
| | 10 | 3508 | 5892 | 5892 | 84 | 680 | 1176 | 4 | 4 |
| 192 | 1≀ | 6760 | 6516 | 7452 | 81 | 1320 | 1320 | 8 | 4 |
| | 2 | 6632 | 6452 | 7388 | 81 | 1320 | 1320 | 8 | 4 |
| | 3 | 5276 | 4944 | 5880 | 64 | 1056 | 1056 | 8 | 4 |
| | 4 | 6632 | 6388 | 7324 | 81 | 1320 | 1320 | 8 | 4 |
| | 5 | 6632 | 6452 | 7388 | 81 | 1320 | 1320 | 8 | 4 |
| | 6 | 5276 | 4944 | 5880 | 64 | 1056 | 1056 | 8 | 4 |
| | 7 | 6631 | 6388 | 7324 | 81 | 1320 | 1320 | 8 | 4 |
| | 8 | 6632 | 6452 | 7388 | 64 | 1320 | 1320 | 8 | 4 |
| | 9 | 5276 | 4944 | 5880 | 64 | 1056 | 1056 | 8 | 4 |
| | 10 | 6632 | 6388 | 7324 | 81 | 1320 | 1320 | 8 | 4 |
| | 11 | 6128 | 4944 | 5880 | 81 | 1192 | 1048 | 8 | 4 |
| | 12 | 2848 | 2752 | 2752 | 64 | 544 | 544 | 4 | 4 |
| 256 | 1≀ | 5276 | 3244 | 3656 | 64 | 1056 | 544 | 8 | 4 |
| | 2 | 6568 | 6192 | 3604 | 81 | 1320 | 1224 | 8 | 4 |
| | 3 | 6568 | 6784 | 7196 | 80 | 1320 | 1360 | 8 | 4 |
| | 4 | 6568 | 6784 | 7196 | 81 | 1320 | 1360 | 8 | 4 |
| | 5 | 6568 | 6784 | 7196 | 80 | 1320 | 1360 | 8 | 4 |
| | 6 | 6568 | 6784 | 7196 | 81 | 1320 | 1360 | 8 | 4 |
| | 7 | 6568 | 6784 | 7196 | 80 | 1320 | 1360 | 8 | 4 |
| | 8 | 6568 | 6784 | 7196 | 81 | 1320 | 1360 | 8 | 4 |
| | 9 | 6568 | 6784 | 7196 | 80 | 1320 | 1360 | 8 | 4 |
| | 10 | 6568 | 6784 | 7196 | 81 | 1320 | 1360 | 8 | 4 |
| | 11 | 6568 | 6784 | 7196 | 80 | 1320 | 1360 | 8 | 4 |
| | 12 | 6568 | 6784 | 7196 | 81 | 1320 | 1360 | 8 | 4 |
| | 13 | 6568 | 6784 | 7196 | 80 | 1320 | 1360 | 8 | 4 |
| | 14 | 3508 | 5828 | 5828 | 81 | 680 | 1176 | 4 | 4 |

≀: Including initial key XOR.

☆: Regular version.
◎: Shallow version.
◈: Shallow/low depth version.

**(c)** Using S-box with Toffoli depth 3.

| AES | Round | #CNOT ☆ | #CNOT ◎ | #CNOT ◈ | #NOT ☆◎◈ | #Toffoli ☆ | #Toffoli ◎◈ | $TD$ ☆ | $TD$ ◎◈ |
|---|---|---|---|---|---|---|---|---|---|
| 128 | 1≀ | 12776 | 7136 | 7548 | 81 | 3120 | 1560 | 6 | 3 |
| | 2 | 12648 | 12936 | 13348 | 81 | 3120 | 3120 | 6 | 3 |
| | 3 | 12648 | 12936 | 13348 | 81 | 3120 | 3120 | 6 | 3 |
| | 4 | 12648 | 12936 | 13348 | 81 | 3120 | 3120 | 6 | 3 |
| | 5 | 12648 | 12936 | 13348 | 81 | 3120 | 3120 | 6 | 3 |
| | 6 | 12648 | 12936 | 13348 | 81 | 3120 | 3120 | 6 | 3 |
| | 7 | 12648 | 12936 | 13348 | 81 | 3120 | 3120 | 6 | 3 |
| | 8 | 12648 | 12936 | 13348 | 81 | 3120 | 3120 | 6 | 3 |
| | 9 | 12648 | 12936 | 13348 | 84 | 3120 | 3120 | 6 | 3 |
| | 10 | 6484 | 7408 | 7408 | 84 | 1560 | 1560 | 3 | 3 |
| 192 | 1≀ | 12968 | 13032 | 13444 | 81 | 3120 | 3120 | 6 | 3 |
| | 2 | 12840 | 12936 | 13348 | 81 | 3120 | 3120 | 6 | 3 |
| | 3 | 10012 | 10332 | 10744 | 64 | 2496 | 2496 | 6 | 3 |
| | 4 | 12840 | 12904 | 13316 | 81 | 3120 | 3120 | 6 | 3 |
| | 5 | 12840 | 12936 | 13348 | 81 | 3120 | 3120 | 6 | 3 |
| | 6 | 10012 | 10332 | 10744 | 64 | 2496 | 2496 | 6 | 3 |
| | 7 | 12840 | 12904 | 13316 | 81 | 3120 | 3120 | 6 | 3 |
| | 8 | 12840 | 12936 | 13348 | 64 | 3120 | 3120 | 6 | 3 |
| | 9 | 10012 | 10332 | 10744 | 64 | 2496 | 2496 | 6 | 3 |
| | 10 | 12840 | 12904 | 13316 | 81 | 3120 | 3120 | 6 | 3 |
| | 11 | 12840 | 10128 | 10540 | 81 | 3120 | 1560 | 6 | 3 |
| | 12 | 5008 | 5136 | 5548 | 64 | 1248 | 1248 | 3 | 3 |
| 256 | 1≀ | 10268 | 5788 | 6200 | 64 | 2496 | 1248 | 6 | 3 |
| | 2 | 12648 | 11648 | 12060 | 81 | 3120 | 2808 | 6 | 3 |
| | 3 | 12648 | 11648 | 12060 | 80 | 3120 | 3120 | 6 | 3 |
| | 4 | 12648 | 11648 | 12060 | 81 | 3120 | 3120 | 6 | 3 |
| | 5 | 12648 | 11648 | 12060 | 80 | 3120 | 3120 | 6 | 3 |
| | 6 | 12648 | 11648 | 12060 | 81 | 3120 | 3120 | 6 | 3 |
| | 7 | 12648 | 11648 | 12060 | 80 | 3120 | 3120 | 6 | 3 |
| | 8 | 12648 | 11648 | 12060 | 81 | 3120 | 3120 | 6 | 3 |
| | 9 | 12648 | 11648 | 12060 | 80 | 3120 | 3120 | 6 | 3 |
| | 10 | 12648 | 11648 | 12060 | 81 | 3120 | 3120 | 6 | 3 |
| | 11 | 12648 | 11648 | 12060 | 80 | 3120 | 3120 | 6 | 3 |
| | 12 | 12648 | 11648 | 12060 | 81 | 3120 | 3120 | 6 | 3 |
| | 13 | 12648 | 11648 | 12060 | 80 | 3120 | 3120 | 6 | 3 |
| | 14 | 6484 | 10224 | 10224 | 81 | 1560 | 1560 | 3 | 3 |

≀: Including initial key XOR.

☆: Regular version.

◎: Shallow version.

◈: Shallow/low depth version.