Quantum Analysis of AES
Lowering Limit of Quantum Attack Complexity

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

[1] Division of IT Convergence Engineering, Hansung University, Seoul, South Korea

[2] School of Computer Science and Engineering, Nanyang Technological University, Singapore

starj1023@gmail.com, anubhab.baksi@ntu.edu.sg, khj1594012@gmail.com, thdrudwn98@gmail.com
hwajeong84@gmail.com, anupam@ntu.edu.sg

Abstract. Quantum computing is considered among the next big leaps in computer science. While a fully functional quantum computer is still in the future, there is an ever-growing need to evaluate the security of the symmetric key ciphers against a potent quantum adversary. Keeping this in mind, 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 20 implementations per AES variant (thus totaling in 60), by taking the state-of-the-art advancements in the relevant fields into account.

In a nutshell, 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 86 percent. Furthermore, we analyze the Jaques et al.'s Eurocrypt'20 implementations in details, fix the bugs (arising from some problem of the quantum computing tool used and not related to their coding) and report corrected benchmarks. To the best of our finding, our work improves from all the previous works (including the Asiacrypt'22 paper by Huang and Sun and the Asiacrypt'23 paper by Liu et al.) in terms of various quantum circuit complexity metrics (Toffoli depth, full depth, Toffoli/full depth - qubit count product, full depth - gate count product, etc.). Also, our bug-fixing of Jaques et al.'s Eurocrypt'20 implementations seem to improve from the authors' own bug-fixing, thanks to our architecture consideration.

Equipped with the basic AES implementations, we further investigate the prospect of the Grover's search. We also propose three 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 for circuit complexity (like, depth-squared - qubit count product, depth - gate count product).

Keywords: Quantum Implementation · Grover's Search · AES

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 [34], 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 (refer to [25, 45])[3].

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[4] being organized by the US government's National Institute of Standards and Technology (NIST). For example, the Public

---

[3]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.

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

Key Encryption & Key Encapsulation Mechanism (PKE & KEM) finalist CRYSTALS-KYBER [64] and the Digital Signature (DS) finalist CRYSTALS-DILITHIUM [28] use SHA-3 in some form[5]. 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 are defined over secret key ciphers (variants of AES for PKE & KEM, and variants of SHA-3 for DS). As noted in [44, Section 1], this essentially calls for a concrete and precise resource estimates that would be required by an attacker with a quantum computer at disposal.

Therefore, finding the generic quantum security level for a secret key cipher is among the top research directions (see Section 2.3 for related works). One of the main way 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 [33]. 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 [20], thereafter finding the complexity for the Grover's search [33]. 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 [50, 52, 53, 54, 55, 71, 74]). Our objective lies in reducing the cost in various metrics (see Section 2.1 for an overview on the quantum gates); 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

The prerequisite for this work is summarized in Section 2. In particular, the quantum gates are briefly described in Section 2.1, Grover's search in Section 2.2 and previous literary works in Section 2.3.

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.

We observe that the implementation by [44] 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 [77], Asiacrypt'22 [36], Asiacrypt'23 [54] and Indocrypt'22 [39] 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 [77] (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 [44]. We also include the two implementations done in [36] for a quick comparison. In [36], 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, ♦ $\times$ ⊙; FD-M, ✳ $\times$ ⊙; and FD-G, ✳ $\times$ ✳) 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 Section 2.2 and Section 2.4). Our quantum implementations offer the best trade-offs in terms of TD^2-M (♦^{2} $\times$ ⊙) and FD^2-M (✳^{2} $\times$ ⊙) (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):

☆ The *regular* version (originally conceived in [44], but bug-fixed by us) uses the least qubit count and FD-M cost in our work, and reduces Toffoli circuit depth compared to the previous works for all the 3 variants. The MixColumn implementation is taken from [54], which allows zero ancilla/garbage qubit and incurs 92 CNOT

---

[5]Recently, we have also seen ASCON-SIGN [67], which uses hash function to provide quantum-secure signature.

gates. This architecture was proposed by [44], though it has some programming/estimation issues related to non-linear operations, such as S-box, key schedule, and out-of-place MixColumn.

◎ 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) with an improved pipeline architecture. According to [77], this is an important notion of circuit complexity. Similar to the regular version, the MixColumn implementation is taken from [54]. Note that this version was used in [54].

◈ The *shallow/low depth* version (our innovation) looks for reducing the circuit depth by introducing a new low quantum depth implementation of MixColumn (based on the classical implementation from [50], but optimized for quantum). This version can be considered optimal when the parallelization of Grover's search is unavoidable under the constraints of depth (related discussion is given in Section 2.4).

Apart from these three architecture (where we implement the quantum circuit from scratch), we also take a look at the implementation by Jaques et al. in Eurocrypt'20 [44] (❄) for AES-128/-192/-256. As noted in Section 5 and Appendix C, that implementation contained bugs[6] (this was confirmed by the authors, most notably in [43] 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 [44] is used (✚) or the Maximov's MixColumn implementation from [58] (✛) is used (both were used in [44]). In order to keep the modification at minimum, we reuse the same design choices made in [44]. For this reason, we reuse the S-box implementation as in [44], which was adopted from [17] (❧).

Orthogonal to these architecture, we also introduce three new S-box implementations (see Table 3). We improve two S-box implementations (from [36, 54]) 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 three S-box implementations, two are used in our architectures, namely regular (✩), shallow (◎) and shallow/low depth (◈). The three newly implemented S-boxes incur the lowest full depth (61, 58, and 56) compared to the other implementations that we found during our literature survey.

Among the three new S-box implementations, two are used in this work (❀, ✿) in order to reduce the cost. Besides, 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 one is revised by us). The shallow version (◎) (which is indeed our innovation) was used by Liu et al. in Asiacrypt'23 [54].

In this work, we present 20 distinct implementations for each variant of AES (each of the following is considered with Toffoli and AND gates):

✩ Regular version.
    ❀ New 3 Toffoli depth S-box, MixColumn from [54].
    ✿ New 4 Toffoli depth S-box. MixColumn from [54].
◎ Shallow version.
    ❀ New 3 Toffoli depth S-box, MixColumn from [54].
    ✿ New 4 Toffoli depth S-box, MixColumn from [54].
◈ Shallow/low depth version.
    ❀ New 3 Toffoli depth S-box, New MixColumn.
    ✿ New 4 Toffoli depth S-box, New MixColumn.
❄ Bug-fixing of JNRV (Eurocrypt'20) [44].
    (a) ✳ Fixed depth, ❧ S-box from [17], ✚ in-place MixColumn [44].
    (b) ✳ Fixed depth, ❧ S-box from [17], ✛ Maximov's MixColumn [58].
    (c) ✶ Fixed qubit count, ❧ S-box from [17], ✚ in-place MixColumn [44].
    (d) ✶ Fixed qubit count, ❧ S-box from [17], ✛ Maximov's MixColumn [58].

As a consequence of our analysis, the state-of-the-art bounds of the quantum security level (Section 2.4) 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 7 for a quick view). Some additional information/discussion can be found in Appendices A (a short discussion on the AES variants), Appendix B summarizes the novelty/new construction presented in this paper, C (detailed discussion on the Eurocrypt'20 [44] bug), D (more details about our depth and qubit reduced S-box/MixColumn implementations), and E (per-round break-up of quantum resource requirement).

---

[6]To be more precise, the bug was caused due to an inherent issue in Q# and not related to their coding.

Table 1: Performance comparison of AES quantum implementations.

AES		Toffoli depth (TD) ♦		Qubit count (M) ✿		$TD\text{-}M$ cost $(TD \times M)$ ♦ × ✿		Full depth (FD) ✳		$FD\text{-}M$ cost $(FD \times M)$ ✳ × ✿	
128	GLRS [32]	12672	(99.68)	984	(−71.3)	12469248	(98.9)	110799	(99.34)	109026216	(97.7)
	LPS [49]	1880	(97.87)	864	(−74.8)	1624320	(91.56)	28927	(97.47)	24992928	(89.97)
	ZWSLW [77]	2016	(98.02)	512	(−85.06)	1032192	(86.72)	.		.	
	HS [36] † 18	820	(95.12)	492	(−85.65)	403440	(66.01)	.		.	
	HS [36] † 9	1558	(97.43)	374	(−89.09)	582692	(76.47)				
	LXXZZ [52]	476	(91.60)	474	(−86.17)	225624	(39.23)	.		.	
	LPZW [54]	40	(0)	3688	(7.05)	147520	(7.05)	840	(12.98)	3097920	(19.11)
	✳ ✳✚	2394	(98.33)	1656	(−51.69)	3964464	(96.54)	33320	(97.81)	55177920	(95.46)
	✳✛	114	(64.91)	5088	(32.63)	580032	(76.36)	1612	(54.65)	8201856	(69.45)
	❀	40◎◈		3428◎		137120◎		731◎		2505868◎	
192	GLRS [32]	11088	(99.57)	1112	(−70.33)	12329856	(98.54)	96956	(99.1)	107815072	(96.96)
	LPS [49]	1640	(97.07)	896	(−76.09)	1469440	(87.76)	25556	(96.58)	22898176	(85.69)
	ZWSLW [77]	2022	(97.63)	640	(−82.92)	1294080	(86.1)	.		.	
	LXXZZ [52]	572	(91.61)	538	(−85.65)	307736	(41.54)	.		.	
	LPZW [54]	48	(0)	3944	(4.97)	189312	(4.97)	1010	(13.47)	3983440	(17.77)
	✳ ✳✚	2682	(98.21)	1976	(−47.28)	5299632	(96.61)	37328	(97.65)	73760128	(95.56)
	✳✛	138	(65.22)	5664	(33.83)	781632	(76.98)	1936	(54.86)	10965504	(70.13)
	❀	48◎◈		3748◎		179904◎		874◎		3275752◎	
256	GLRS [32]	14976	(99.63)	1336	(−66.9)	20007936	(98.87)	130929	(99.22)	174921144	(97.63)
	LPS [49]	2160	(97.41)	1232	(−69.47)	2661120	(91.51)	33525	(96.94)	41302800	(89.98)
	ZWSLW [77]	2292	(97.56)	768	(−80.97)	1760256	(87.16)	.		.	
	LXXZZ [52]	646	(91.33)	602	(−85.08)	388892	(41.88)	.		.	
	LPZW [54]	56	(0)	4456	(9.43)	249536	(9.43)	1176	(12.84)	5240256	(21.06)
	✳ ✳✚	3306	(98.31)	2296	(−43.11)	7590576	(97.02)	46012	(97.77)	105643552	(96.08)
	✳✛	162	(65.43)	6240	(35.32)	1010880	(77.64)	2264	(54.73)	14127360	(70.72)
	❀	56◎◈		4036◎		226016◎		1025◎		4136900◎	

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

†: Choice of p.

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

❀: Bug-fixed JNRV [44] (using Toffoli gate).

✳: Bug-fixed depth.   ✚: In-place MixColumn [44].
✳: Bug-fixed qubit count.   ✛: Maximov's MixColumn [58].

Our source codes are written in ProjectQ[7], 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[8]. Not directly relevant here, but our idea can be applied to other ciphers also, as it can be seen from [4,39].

Some of the highlights of our work include (see also Appendix B for related information):

(I) The fixed-qubit version for bug-fixing JNRV [44] is introduced. In total, we introduce five new architecture (four of which are entirely our own).

(II) The 16 quantum depth MixColumn implementation reported in [54] is incorporated in our designs. We also introduce our own MixColumn implementation that incurs quantum depth of 8 with fewer gate and qubit counts.

(III) We introduce three S-box implementations, which reduces the full depth and/or qubit count. One of our implementation takes 56 full depth, which is probably the least reported so far.

(IV) In-depth discussion in various places, including discussion/explanation about MAXDEPTH, bug and bug-fixing of JNRV [44], and the importance of depth optimization are included to make the content easier to understand for a non-expert reader.

These cover our new innovation (such as, the fixed-qubit version for bug-fixing JNRV [44]) and recent development (such as, [51, 52, 54, 74]). As far as we can tell, our work presents the state-of-the-art results in the quantum

---

[7]Homepage: https://projectq.ch/.
[8]https://github.com/starj1023/AES_QC.

implementation/analysis of AES-128/-192/-256; improving from the recent works like Asiacrypt'22 [36], Asiacrypt'23 [54] and also the bug-fixing by the Eurocrypt'20 authors themselves [43].

2 Background

2.1 Quantum Gate Basics

Some of the classical gates have quantum counterpart. The X gate performs the classic NOT operation on a single qubit: X $(x) \rightarrow (\sim x)$. The CNOT gate operates on two qubits (i.e., x and y) and performs the classic XOR operation: CNOT $(x, y) \rightarrow (x, x \oplus y)$. In CNOT (x, y), x is the control qubit and y is the target qubit, so x is XORed to y. The Toffoli gate operates on three qubits and performs the classic AND operation: Toffoli $(x, y, z) \rightarrow (x, y, z \oplus (x \cdot y))$. In Toffoli (x, y, z), x and y are the control qubits and z is the target qubit, so the ANDed value of x and y (i.e., $x \cdot y$) is XORed to z.

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-gate 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 [13]. 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 [59].

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.

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 [3, 35, 65]. 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 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 [3].

Additionally, we adopt the quantum AND gates from [44]. 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† gate) is designed according to the measured value of the target qubit of the AND† gate. This AND† gate is counted as (7 Clifford gates + 1 Measurement gate) in resource estimation. Although not adopted in our work, there is another version of the AND gate [31] that does not require an ancilla qubit, but has a T-depth of 2.

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 [33] 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:

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

2. The cipher (Enc) is implemented as a quantum circuit and placed in oracle. In oracle $f(x)$, the plaintext (p) is encrypted with the key in the superposition state. As a result, the ciphertexts for all key values are generated. The sign of the solution key is changed to a negative by comparing it with the known ciphertext. The condition $(f(x) = 1)$ changes the sign to negative and applies to all states. For this phase flip, an n-qubit controlled Z gate is utilized (n is the length of the ciphertext, c).

$$f(x) = \begin{cases} 1 \text{ if } Enc_K(p) = c \\ 0 \text{ if } Enc_K(p) \neq c \end{cases} \tag{2}$$

$$U_f(|\psi\rangle\,|-\rangle) = \frac{1}{2^{k/2}} \sum_{x=0}^{2^k-1} (-1)^{f(x)} |x\rangle\,|-\rangle \tag{3}$$

3. Lastly, the diffusion operator[9] amplifies the amplitude of the negative sign state. Diffusion operator is implemented with the following (H gates layer $\rightarrow$ X gates layer $\rightarrow$ k-qubit controlled Z gate $\rightarrow$ X gates layer $\rightarrow$ H gates layer). In [62], 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 gates layer $\rightarrow$ k-qubit controlled Z gate $\rightarrow$ H gates layer).

The Grover's search executes Equations (2), (3) 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 $\lfloor \frac{\pi}{4}\sqrt{2^k} \rfloor$ [18], 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 1 shows the Grover's oracle of exhaustive key search. Encryption† is defined as the reverse operation of encryption, which reverts to the state before encryption.
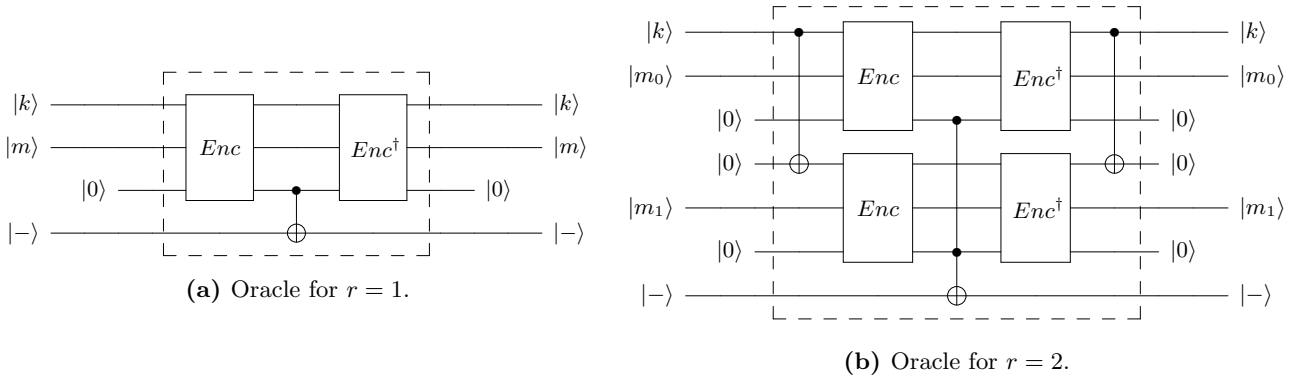
Figure 1: Schematic view for key search using Grover's algorithm.

2.3 Related Work

Quantum analysis of secret-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 [15, 36, 44, 49, 54, 77][10], SIMON [7], SPECK [6, 37], PRESENT and GIFT [40], SHA-2 and SHA-3 [2], FSR-based ciphers [5], ChaCha [12], SM3 [66, 75], RECTANGLE and KNOT [10], DEFAULT [38], ARIA [19], few Korean ciphers [41, 42], SPECK and LowMC [39], CHAM [72], ASCON [61, 63].

However, this is not the only active direction of research; there are other avenues which try to find an efficient quantum attack for a secret-key cipher. One may, for instance, refer to classical attacks that are ported to the quantum realm [30, 46], or specialized quantum attacks like [14, 26, 27]. These avenues, though important, are out-of-scope for our current work.

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 [77] qubit count and performance.

---

[9]Since the diffusion operator is usually generic, it does not require any special technique for implementation.

[10]As far as we can tell, the authors of [15] only made some estimates but did not present any implementation.

➠ They chose an improved S-box implementation atop the Eurocrypt'20 implementation [44] with proposal for a quick fix for the qubit count.

We think there is some scope for improvement on the patch done by [36] on the Eurocrypt'20 implementation (based on the Q# code[11]). Also, the number of qubits was estimated manually in [36, Table 7] in the bug-fix of [44]. 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 [71]), 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 [36], we note the following points. Our approaches are mostly disjoint from that of [36] (note that we use the 16 quantum depth MixColumn implementation from [54], as opposed to the 30 quantum depth implementation from [71]); and when their S-box implementation is used in our implementation, our result outperforms theirs (thus we have the best-known implementation so far). 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 [36]. Further, we cover optimized quantum implementations of AES-192 and AES-256 as well.

Reflection on LPZW (Liu, Preneel, Zhao and Wang in Asiacrypt'23). The recent Asiacrypt'23 paper by Liu et al. [54] directly adopted the shallow (◎) version introduced in this paper. The other three contributions in [54] are as summarized as:

➠ They proposed a new S-box implementation (see Table 3 for its benchmark).
➠ They reduced the number of qubits by sharing idle ancilla qubits in the shallow (◎), similar to what we reduced ancilla qubits for MixColumns in the shallow/low depth (◈) version.
➠ They managed to reduce the MixColumn implementation by [71] from 30 to 16 (see Table 4).

In this regard, we further improve their results by including their 16 quantum depth MixColumn implementation and reducing the full depth & qubit count of their S-box implementations into our repertoire.

Ancilla qubit reduction by LPZW. In [54], 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 [54], this efficient sharing technique was also applied between SubBytes and SubBytes† on the shallow version. Consequently, they reduced the number of ancilla qubits by sharing the idle state ancilla qubits of SubBytes and SubBytes†.

2.4 NIST Security Levels

The following security levels were defined by NIST [59] 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.

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 [61, 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 [32]), NIST estimated used in [59] 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 [32] 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 the passage of time, as more research works on the AES family have been being reported, the complexity for the security levels (1, 3 and 5) have 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 [60]. Presently, NIST has defined new quantum attack costs for AES-128, 192, and 256, based on the reported costs from [44], 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

---

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

implementation. In this paper, We analyze these issues from [44] 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 from Table 11, where we show the impact on our work in reshaping the security levels. In particular, the following new bounds are achieved (see also Table 10(b)):

☞ Level 1: $2^{156.2654}$; with total Clifford, T and measurement gates ($\ast + \divideontimes + \star$) = $2^{82.2762}$; full depth ($\ast$) = $2^{73.9884}$.

☞ Level 3: $2^{221.5869}$; with total Clifford, T and measurement gates ($\ast + \divideontimes + \star$) = $2^{115.3414}$; full depth ($\ast$) = $2^{106.2461}$.

☞ Level 5: $2^{286.0800}$; with total Clifford, T and measurement gates ($\ast + \divideontimes + \star$) = $2^{147.6050}$; full depth ($\ast$) = $2^{138.4751}$.

Along with this, NIST proposed a parameter called MAXDEPTH to impose a limit on circuit depth. 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[12]). However, if it turns out that the depth restriction is not within the stipulated bound, then the following approaches can be undertaken [47]:

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., [47, Table 2]). It is worth noting that the previous implementations like [1, 36, 49, 77] also did not appear to consider the MAXDEPTH limit.

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 [73], 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 [44, Section 3.4], to minimize the TD-M (Toffoli depth and qubit count product, $\blacklozenge \times \circ$) and FD-M (full depth and qubit count product, $\ast \times \circ$) costs under the parallelization of Grover's search, it is necessary to minimize the TD^2-M ($\blacklozenge^2 \times \circ$) and FD^2-M ($\ast^2 \times \circ$) 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 of 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 sholud be $\frac{G \cdot FD}{\text{MAXDEPTH}}$. This formular mirrors NIST's method for estimating the quantum attack cost for AES [59, 60]. 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: $2^{156.2654}$, Level 2: $2^{221.5869}$, and Level 5: $2^{286.0800}$).

As of now, we remark that the depths ($\ast$ and $\blacklozenge$) of our AES quantum circuits are the lowest when compared to other quantum circuits available in the literature [1, 36, 49, 77]. Table 2 displays a quick view where the related works (namely, GLRS [32] and LPS [49]) are compared with respect to our implementations in terms of full depth ($\ast$). Note that, only AES-128 satisfies the MAXDEPTH criterion (i.e., $\leq 2^{96}$).

---

[12]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.

One may further note that the depth of quantum attack on AES-128 (i.e., Level 1) is within the permitted MAXDEPTH limit ($2^{73.9884}$ using the AND gate; the same using the Toffoli gate is $2^{74.0314}$). However, the same cannot be stated for AES-192 and -256, since the full depth figures are respectively $2^{106.2461}$ and $2^{138.4751}$ (using the AND gate; the same using the Toffoli gate are $2^{106.2892}$ and $2^{138.5190}$, 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 [32]	LPS [49]	This work							MAXDEPTH ($\leq 2^{96}$)
			☆	◎	◈	✽ (Toffoli)		✽ (AND)		
128	$2^{81.2141}$	$2^{79.4751}$	✿: $2^{74.3414}$	✿: $2^{74.1243}$	✿: $2^{73.9884}$	✽✚: $2^{79.6576}$	✽✚: $2^{75.5008}$	✽✚: $2^{79.0636}$	✽✚: $2^{75.4543}$	✓
			❋: $2^{74.4563}$	❋: $2^{74.0607}$	❋: $\mathbf{2^{73.9207}}$	✽✛: $2^{79.7011}$	✽✛: $2^{75.3060}$	✽✛: $2^{79.1035}$	✽✛: $2^{74.7847}$	
192	$2^{113.4114}$	$2^{111.2987}$	✿: $2^{106.7172}$	✿: $2^{106.3829}$	✿: $2^{106.2461}$	✽✚: $2^{111.8395}$	✽✚: $2^{107.7756}$	✽✚: $2^{111.2271}$	✽✚: $2^{107.7258}$	✗
			❋: $2^{106.6978}$	❋: $2^{106.3196}$	❋: $\mathbf{2^{106.1763}}$	✽✛: $2^{111.8673}$	✽✛: $2^{107.5703}$	✽✛: $2^{111.2702}$	✽✛: $2^{107.0464}$	
256	$2^{145.6508}$	$2^{143.6871}$	✿: $2^{138.8007}$	✿: $2^{138.6126}$	✿: $2^{138.4751}$	✽✚: $2^{144.1412}$	✽✚: $2^{140.0098}$	✽✚: $2^{143.5283}$	✽✚: $2^{139.9553}$	✗
			❋: $2^{138.9245}$	❋: $2^{138.5509}$	❋: $\mathbf{2^{138.4071}}$	✽✛: $2^{144.1679}$	✽✛: $2^{139.7964}$	✽✛: $2^{143.5701}$	✽✛: $2^{139.2680}$	

☆: Regular version (using AND gate).
◎: Shallow version (using AND gate).
◈: Shallow/low depth version (using AND gate).
✿: S-box with Toffoli depth 4.
❋: S-box with Toffoli depth 3.
✽: Bug-fixed JNRV [44] (using S-box from [17] ✿.)
✽: Bug-fixed depth. | ✽: Bug-fixed qubit count.
✚: In-place MixColumn [44]. | ✛: Maximov's MixColumn [58].

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 $FD\text{-}M$ cost (which is the full depth - qubit count product, ✽ × ●). 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 (◆), $TD\text{-}M$ cost (which is the Toffoli depth - qubit count product, ◆ × ●), and $TD^2\text{-}M$ cost (which is the Toffoli depth^2 - qubit count product, ◆^{2} × ●). The shallow/low depth version seems to achieve the lowest depth for quantum circuit implementation. The shallow/low version is the optimal choice 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, ✽), $FD\text{-}G$ cost, (which is the metric used by NIST to estimate quantum attack cost, ✽ × ✽) and $FD^2\text{-}M$ cost (which is the full depth^2 - qubit count product, ✽^{2} × ●).

All the three versions (☆, ◎, ◈) employ the reverse operation (i.e., un-compute) to initialize/clean ancilla qubits. Notably, if we allocate new qubits every time they are needed, both gate count and circuit depth can be reduced. However, our design philosophy revolves around using reverse operations and attempting to develop optimal architectures for AES quantum circuits, emphasizing the importance of maintaining the best balance between depth and qubit count while aiming for the lowest possible depth.

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 2(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. Simply put, 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 5(b)), which reduces the Toffoli depth and full depth from the previous works (as in Figure 5(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 (♦ and ❋), 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 [54], 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 the MixColumn implementation from [50] 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 [1, 32, 36, 49, 69, 77]. 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 second 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 [77]. TD-M cost of our AES quantum circuits is slightly larger than AES quantum circuits from [52] (which is the best). However, as stated in the NIST document [60], 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 Section 2.4 for the rationale). This product is taken as the trade-off indicator for the quantum circuit in [44]. From Table 5, it can be observed that our TD-M is slightly larger than those in [52], while TD^2-M is significantly smaller than that in [52].

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 second best trade-offs in terms of FD-M cost. However, 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)

Table 3 shows the resources required for the implementations found by Boyar-Peralta [16, 17] and the resources for the S-boxes used by the previous authors [49, 77], among others. Resource estimation is performed in ProjectQ and according to the method of [3], 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 implemented AES quantum circuits with AND gates.

Note that the S-box implementation in [32] is based on a field inversion technique, while the rest are based on some version of the Boyar-Peralta's algorithm [16, 17]. Apart from these, another method which is a courtesy of Dansarie [21, 22] exists. This is rather generic, as it can find implementation of an arbitrary 8-bit S-box (i.e., unlike [16, 17], 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[13] we checked the implementation of the AES S-box. In total, we found 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).

If the Boyar-Peralta's S-box implementations [16, 17] are directly ported to quantum, then the version of [17] requires more ancilla qubits (120 ancilla qubits) than the quantum version of [16] (107 ancilla qubits), but attains lower depth. JNRV [44] adopted the implementation of the S-box of [17] to the corresponding quantum circuit as-is. In [49, 77], the authors took the first S-box implementation by Boyar-Peralta from [16] and presented the S-box quantum circuit with a reduced number of qubits.

Huang and Sun reported an improved quantum implementation for the S-box of [44] in their Asiacrypt'22 paper [36]. They presented two quantum implementations of reduced Toffoli depth with new observations of the classical implementation of the AES S-box as given in [17]. 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.

---

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

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

Method	#CNOT ❄	#1qCliff ❅	#T ✛	TD ◆	M ◎	Full depth ❄
S-box [32]	1818	124	1792	88	40	951
S-box [16]	358	68	224	8	123	104
S-box [17] ✤	392	72	238	6	136	85
S-box [49]	628	98	367	40	32	514
S-box [77]	437	72	245	55	22	339
S-box [21, 22] 391 lines	1470	670	1218	66	399	640
406 lines	1507	548	1245	74	414	709
413 lines	1484	561	1169	62	421	591
409 lines	1483	574	1190	74	416	693
400 lines	2244	1006	2254	111	408	998
S-box [36]	418	72	238	4	136	72
	824	160	546	3	198	69
S-box [51]	·	·	·	32	20	·
S-box [52]	·	·	·	24	21	·
	·	·	·	22	22	·
S-box [54]	372	72	238	4	90	69
S-box	418	72	238	4	136	61
✿	366	72	238	4	84	58
❀	781	160	546	3	152	56

✤: Reused in this work to fix [44] ❅.
✿: Used in this work (Toffoli depth 4).
❀: Used in this work (Toffoli depth 3).

However, we propose new S-box implementations in this paper in order to push the state-of-the-art even further (we manage to reduce the full depth from theirs).

By prudently using LIGHTER-R [23], the authors of [51] proposed a new S-box implementation for AES that requires low qubit count. Subsequently, the authors of [52] presented another low qubit S-box implementation. Since detailed quantum benchmarks with Clifford + T gates are not provided in [51, 52][14], only Toffoli depth and qubit count are reported in Table 3.

Most recently, a new S-box implementation was presented in [54]. This, together with their 16 quantum depth MixColumn, allowed the authors of [54] to reduce the overall cost (compared to an earlier version of this paper), despite using the same shallow (◎) architecture as ours. Starting with their S-box implementation, we reduce the full depth as well as ancilla qubit count from [54] (shown in Table 3). The theory behind our improvement can be found in Appendix D.

We found two reversible implementations of the AES S-box from [57, Appendices C and D]. However, those are given in raster graphics format and quite difficult to read[15].

One may note that the AES implementation in [77] required the implementation of the inverse S-box. In our case, however, 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 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 relatively high qubit count

---

[14]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.

[15]We contacted the authors for quickly readable format — currently we are awaiting their response.

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 2(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 2(a). On the other hand, we reduce the depth by allocating more ancilla sets initially. The notation S-box† is described in Appendix A.3.

(a) Using one ancilla set.

(b) Using multiple ancilla sets.
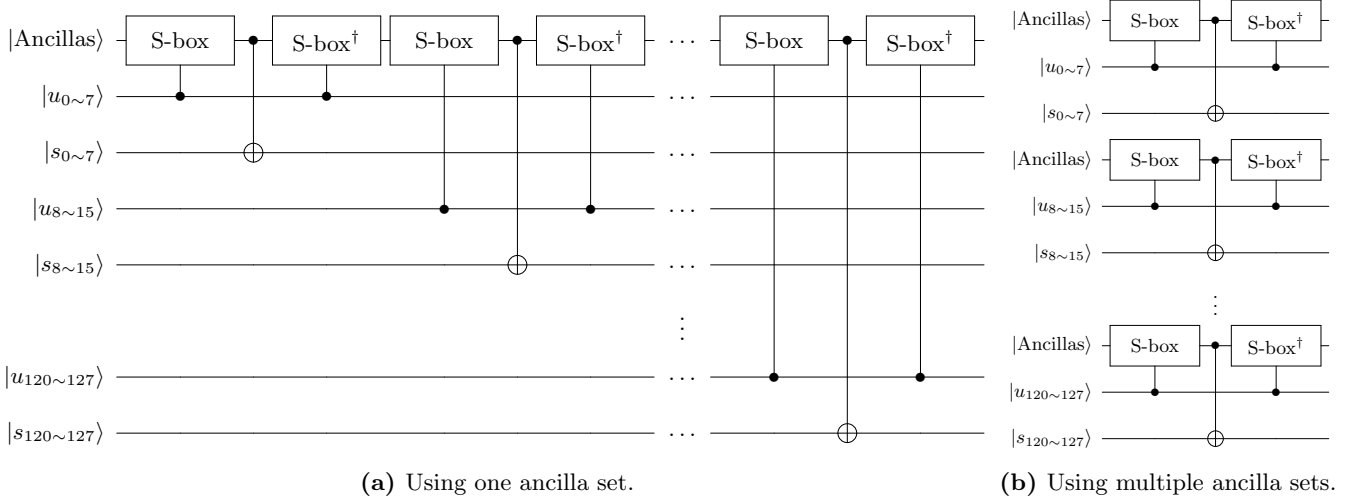
Figure 2: SubBytes implementation in quantum.

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 20×68 ancilla qubits for S-boxes with Toffoli depth 4 (✿) and 20×136 ancilla qubits for S-boxes with Toffoli depth 3 (❀) to run all S-boxes simultaneously. Figure 2(b) shows 16 S-boxes operation in parallel using multiple ancilla sets. After S-box operations, ancilla qubits are not in a clean state (i.e., not all ancilla is 0). Initialization with 16 S-boxes† operation (i.e., returning to 0) 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

In the key schedule of AES, SubWord operates on rearranged 32-qubit. Out of the $20 \times (68$ or $136)$ ancilla qubits previously decided to use (refer to Section 3.3), $4 \times (68$ or $136)$ ancilla qubits are used to simultaneously operate S-boxes for 32-qubit in the key schedule (16×68 or 16×136 ancilla qubits are used in SubBytes of round). For rearranging the 32 qubits, quantum resources are not used by using logical swap that only 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 is already utilized in [44]. 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 3 shows the quantum circuit for the AES-128 key schedule (see Appendix A.3 for description of Rotation† and SubWord†).

All the S-boxes in key schedule and round function are designed to operate in parallel. That is, the depth is the same as operating an 8-bit S-box once. Quantum implementation for S-box is required for key schedule and SubBytes, and S-box occupies the most resources in AES quantum circuit. In [32], GLRS used the Itoh–Tsujii inversion to implement S-box of AES, which requires a lot of quantum resources.

The hardware design for AES has been adopted to implement an efficient S-box quantum circuit, namely those were proposed by Boyar-Peralta in [16, 17] were frequently used. In [49], Langenberg et al. adopted the S-box

implementation of [16] and converted it to suit their purpose of reducing qubits. The S-box implementation of [16] was adopted and improved in [76]. ZWSLW [77] used the S-box^{-1} (from [17]) implementation in designing a new architecture for AES that reduced number of qubits. For the key schedule, an on-the-fly approach is adopted, and our AES quantum circuit implementation executes the key schedule simultaneously with SubBytes in the round function.
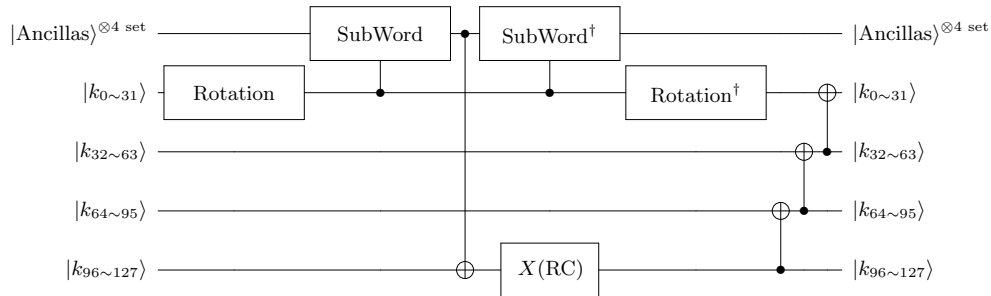
Figure 3: AES-128 key schedule in quantum.

In most implementations of AES quantum circuits, the full depth and Toffoli depth of AES-128 are higher [32,44,49] or similar [77] 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 [71], Xiang et al. presented a novel heuristic search algorithm to optimize the implementation of linear layers based on factorization of binary matrices. When applied to the MixColumn of AES, their algorithm resulted in an implementation using 92 XOR gates (with classical depth 6) in a classical circuit. A different implementation costing 92 XOR gates (with classical depth 6) was reported previously by [58]. These two were the least cost implementations in classical circuits, until another implementation with 91 XOR gates (with classical depth 7) was found by [53]. A new implementation of AES MixColumn was found thanks to [55], which managed to reduce the classical depth to 3 with 103 XOR gates (cf. 103 XOR/3 classical depth implementation from [11]). However, this work came as a tie with another implementation from [50], albeit the latter required 105 XOR gates.

When it comes to quantum implementation, one may observe that the following implementations operate in-place (i.e., of the form $a \leftarrow a \oplus b$ and require only 32 qubits[16]):

(a) PLU factorization in some form (used in [1,32,44,77]);
(b) 92 XOR implementation reported in [71] (used in [36]).

Note from Table 4 that, the implementation by [71] requires the least number of XOR/CNOT gates. This hugely improves from the previous in-place implementations based on the PLU factorization [32, 44, 77]. In contrast, implementations like that of [53,55,58], 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 qubits). On a different direction, the implementation from [50] appears to have lower depth than that of [55] when converted to quantum circuits (despite both having same classical depth). Related discussion can be found in [63].

---

[16]As noted in [63], the Gauss-Jordan reduction also finds an in-place implementation of a binary matrix, but it is probably never used as such (although it was used in [71] as the fallback algorithm for the A* search).

Table 4: Comparison of quantum implementations of AES MixColumn.

Method		#CNOT ✳	M ❄	Depth
MixColumn (Naïve)	GF(2^8) (Encoding [16])	184	64	25
	GF(2) (Encoding [71])			52
MixColumn [32, 77][†]		277	32	39
MixColumn [48]		194	129	15
MixColumn [1][†]		275	32	200
MixColumn [58][✛]		188	126	13
MixColumn [44][✚†]		277	32	111
MixColumn [68]		188	126	17
MixColumn [71][†]				30
MixColumn [74][†]		92	32	28
MixColumn [54][☆◎ †]				16
MixColumn [53][✳]		182	123	16
MixColumn [55][✳]		206	135	13
MixColumn [56]		204	134	13
MixColumn [11]	103 XOR/3 depth	206	135	11
	95 XOR/6 depth	190	127	15
MixColumn [50][✳]		210	137	11
MixColumn[◈]		169	96	8

✛ ✚: Reused in this work to fix [44] ✻.

☆◎: Used in regular and shallow versions.

✳: Least XOR count in classical circuit.

✳: Least depth in classical circuit.

◈: Used in shallow/low depth version.

†: In-place implementation.

We port the implementation of MixColumn in [74] to quantum and use it in our AES quantum circuit. This implementation is used in the regular (☆) and shallow (◎) versions.

The authors of [11] presented two implementations (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). Other than that, an implementation of 108 XOR count is mentioned in [29, Footnote 3/Page 42], but it is not clear to us so far.

In this work, we present our out-of-place implementation of MixColumn, by taking inspiration from [50]. This achieves the least quantum depth (8) in the literature, to the best of our knowledge. This implementation takes the least CNOT gates (namely, 169) among the out-of-place implementations (compared to the other research works that we found in the literature) and used in our shallow/low depth (◈) architecture. Moreover, our out-of-place implementation requires only 32 ancilla qubits excluding the input and output qubits. More details about the quantum depth reduction and ancilla qubit reduction can be found in Appendix D.

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 [63]) was seemingly never studied. With our implementations, one as a 4×4 matrix over GF(2^8), and the other as a 32×32 binary matrix; we notice from Table 4, the CNOT count being the same, that the depth varies — this is probably due to the compiler's inability to optimize for depth. In essence, we do not provide the binary matrix form, rather give instruction to directly implement the GF(2^8) matrix. Interpreting a 4×4 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 32×32 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 [71], while the GF(2^8) version follows the same encoding as [16]. 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 [32][17] (respectively, [11]) was not verified by [44] (respectively, us), or the reduction of quantum depth in [74] from [71]).

One may note from Table 4 that the depth for quantum circuit corresponding to the implementation by [71] 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.

A recent work by [74] looked into this, and this is the first work to directly address the issue of quantum depth optimization[18] to the best of our finding. In particular, the authors in [74] took the 92 CNOT/30 quantum depth implementation from [71], and managed to reduce the quantum depth to 28 (by keeping the same CNOT count). That is further reduced to 16 by [54, Algorithm 3]. This is adopted in our case, for the regular (✩) and shallow (◎) versions.

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 (◈), we need to account for the ancilla qubits (since the implementation [50] is not in-place). 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 (20×120 ✿, or 20×182 ❀). 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 the MixColumn from [50] (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 [1,71,74] where it is 32) when the non-standalone implementation of MixColumns (in which MixColumn does not operate in-place) is considered.

In [54], this sharing method was applied to reduce the ancilla qubits for the SubBytes and SubBytes[†] (not out-of-place MixColumns). That is, in the earlier version of this paper, we combined SubBytes and MixColumns, but in [54], the authors combined SubBytes and SubBytes[†]. When comparing these two choices, applying the sharing method to SubBytes and SubBytes[†] (presented in [54]) is more efficient, so we have adopted it. The details of the sharing method used in [54] is described in Section 4.2.

In this case, we should note that as SubBytes and SubBytes[†] fully utilize ancilla qubits, there is no idle ancilla qubit available for MixColumns (i.e., sharing method cannot be used). Thus, in the shallow/low-depth version, the number of ancilla qubits for the out-of-place MixColumn should be counted. As a result, we allocate only 4×32 ancilla qubits for the MixColumns with our out-of-place MixColumn implementation (32 ancilla qubits).

4 Architecture of AES Quantum Circuits

A combined description of the AES quantum circuits for all the 3 versions (✩, ◎, ◈) is presented here. There are several architectures for designing quantum circuits of AES. The architectures differ in how they store the 128-qubit output generated from SubBytes in each round. In [1,32,49], the basic zig-zag architecture (Figure 4(a)) was adopted that uses 4 lines to save qubits by performing reverses on rounds. In [77], an improved zig-zag architecture that requires only 2 lines of qubits (Figure 4(b)) was presented. By using a quantum circuit of S-box^{-1}, they 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. Simply put, 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 depth and gates. It is a trade-off issue, but in a sense, a generic pipeline

---

[17]In the Eurocrypt'20 paper [44], the authors remarked that they could not reproduce the result from [32] although they used same technique. The reason [44] has a higher depth (full depth: 111) in the implementation of MixColumn compared to [32] (full depth: 39), despite using same technique, is most likely because of this.

[18]Recent optimizations relying on multi-input XOR gates (e.g., [9]) are not quantum compatible.

is probably the most efficient architecture for implementing AES quantum circuits. We believe that it is much more efficient to allocate a new 128-qubits per round than doubling the gates, depth by performing reverse operations on the rounds to save qubits.

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

AES		#CNOT ❅	#NOT ✱	#Toffoli ☆	TD ◆	#qubit (M) ✿	TD-M cost $(TD \times M)$ ◆ × ✿	Full depth ❅	TD^2-M cost $(TD^2 \times M)$ ◆² × ✿
128	GLRS [32]	166548	1456	151552	12672	984	12469248	110799	158010310656
	ASAM [1]	192832	1370	150528	·	976	·	·	·
	LPS [49]	107960	1570	16940	1880	864	1624320	28927	3053721600
	ZWSLW [77]	128517	4528	19788	2016	512	1032192	·	2080899072
	HS [36] † 18	126016	2528	17888	820	492	403440	·	330820800
	HS [36] † 9	126016	2528	17888	1558	374	582692	·	907834136
	LXXZZ [52]	77984	2224	19608	476	474	225624	·	107397024
	LPZW [54] ◎	65736	800	12920	40	3688	147520	840	5900800
	LPZW [54] ◈	75024	800	12920	40	4844	193760	770	7750400
	☆	63000	816	12560	76	2896	220096	1090	16727296
	◎ ♣	63868	816	12380	40	3428	137120	731	5484800
	◈	68764	816	12380	40	4708	188320	667	7532800
	☆	120696	816	29640	57	4256	242592	1069	13827744
	◎ ❀	119636	816	29460	30	6288	188640	710	5659200
	◈	124532	816	29460	30	7568	227040	647	6811200
192	GLRS [32]	189432	1608	172032	11088	1112	12329856	96956	136713443328
	LPS [49]	125580	1692	19580	1640	896	1469440	25556	2409881600
	ZWSLW [77]	152378	5128	22380	2022	640	1294080	·	2616629760
	LXXZZ [52]	90832	2568	22800	572	538	307736	·	176024992
	LPZW [54] ◎	74456	896	14552	48	3944	189312	1010	9086976
	LPZW [54] ◈	85808	896	14552	48	5356	257088	924	12340224
	☆	71536	904	14144	92	3216	295872	1294	295872
	◎ ♣	72008	904	14000	48	3748	179904	874	8635392
	◈	77992	904	14000	48	5284	253632	797	12174336
	☆	136528	904	33384	69	4576	315744	1270	21786336
	◎ ❀	135456	904	33240	36	6608	237888	850	8563968
	◈	141440	904	33240	36	8144	293184	773	10554624
256	GLRS [32]	233836	1943	215040	14976	1336	20007936	130929	299638849536
	LPS [49]	151011	1992	23760	2160	1232	2661120	33525	5748019200
	ZWSLW [77]	177645	6103	26774	2292	768	1760256	·	4034506752
	LXXZZ [52]	110688	3069	27816	646	602	388892	·	251224232
	LPZW [54] ◎	93288	1119	18360	56	4456	249536	1176	13974016
	LPZW [54] ◈	106704	1119	18360	56	6124	342944	1074	19204864
	☆	88092	1111	17576	108	3536	381888	1516	41243904
	◎ ♣	87380	1111	17432	56	4036	226016	1025	12656896
	◈	94452	1111	17432	56	5828	326368	934	18276608
	☆	168892	1111	41496	81	4896	396576	1488	32122656
	◎ ❀	166672	1111	41316	42	6896	289632	997	12164544
	◈	173744	1111	41316	42	8688	364896	907	15325632

†: 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.
❀: S-box with Toffoli depth 3.

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 output qubits of the round. Figure 5(a) shows

the pipeline architecture of our AES-128 quantum circuit in more detail for the regular version (✩), and Figure 5(b) shows the same for the shallow and shallow/low depth versions (◎, ◈). To be more precise, each $R_{1\sim10}$ in Figure 4 represents the full round, but each $R_{1\sim10}$ in Figure 5 does not contain SubBytes.

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†. Therefore, the circuit depth is determined by the number of serial operations of SubBytes and SubBytes† (the depth of SubBytes† is larger than that of MixColumn). In this version, the SubBytes of the current round wait until the SubBytes† of the previous round is completed.

As shown in Figure 5(a), SubBytes generates 128-qubit output and SubBytes† cleans the ancilla qubits. In total, SubBytes runs 10 times and SubBytes† 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†) and AES-256 operates 27 times (14 SubBytes plus 13 SubBytes†).

In SubBytes, S-boxes operate simultaneously. The depth (✳) of SubBytes is 58 equal to the depth of S-box (with Toffoli depth 4) once. Finally, when S-box with Toffoli depth 4 is used, our AES quantum circuits provide a depth of 1090 (about 58×19) for AES-128, 1294 (about 58×23) for AES-256, and 1516 (about 58×27) for AES-256.

It is to be noted that the regular version was originally conceived in [44], 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, while the the role of the original authors [44] cannot be understated, we believe that the current regular version (✩) is partly our own 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 is used, this can be achieved by using 2 sets of 20×120 ancilla qubits. In the shallow version, the first SubBytes in Figure 5(b) uses the first 20×120 ancilla qubits. The second SubBytes uses the second 20×120 ancilla qubits, and at the same time SubBytes† cleans the first 20×120 ancilla qubits. That is, SubBytes† operates simultaneously with the SubBytes of the next round. Conceptually, this can be thought as all SubBytes† in Figure 5(a) are pushed one space to the right. This is possible because SubBytes and SubBytes† 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 (16), which is the ideal depth. The circuit depth of AES-128 is 731 (about 9 rounds $\times$ 74 + 58), that of AES-192 is 874 (about 11 rounds $\times$ 74 + 58), and the same for AES-256 is 1025 (about 13 rounds $\times$ 74 + 58). In the shallow version, up to SubBytes† 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 (◆ $\times$ ◉), Toffoli depth^2 $\times$ qubit count (◆^{2} $\times$ ◉), and , and full depth^2 $\times$ qubit count (✳^{2} $\times$ ◉).

Similar to [54], 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, see Section 2.3 for more details). Just as the authors in [54] 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 [54] use the cleaned ancilla qubits immediately after the first SubBytes† in the second SubBytes. Consequently, the second SubBytes can use the idle ancilla qubits from the first SubBytes†, resulting in a reduction in the number of qubits for the second ancilla set. We have adopted this method, and the required number of qubits has been reduced more than in [54].

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 [50]). In the earlier version of this paper, by sharing ancilla qubits of SubBytes, only output qubits were required for any MixColumn implementation in Table 4 (as described in Section 3.7). However, since sharing the ancilla qubits between SubBytes and SubBytes† (presented in [54]) with this method is more efficient, we have adopted it.

For this reason, in the shallow/low-depth version (utilizing out-of-place MixColumn), implementing the lowest-depth MixColumn is optimal while also considering the number of qubits. The low depth version counts the depth for one round as SubBytes (58) + MixColumns (8) and 32×4 ancilla qubits for MixColumns are allocated (see Section 3.7). The ancilla qubits allocated for MixColumns are reused in subsequent rounds by reverse operation. The low depth version of AES (◈) offers the least Toffoli depth (◆), full depth (✳), and full depth $\times$ total gates (metric to estimate quantum attack cost, ✳ $\times$ ✳).

(a) Basic (GLRS, LPS, ASAM)

(b) Modified (ZWSLW).
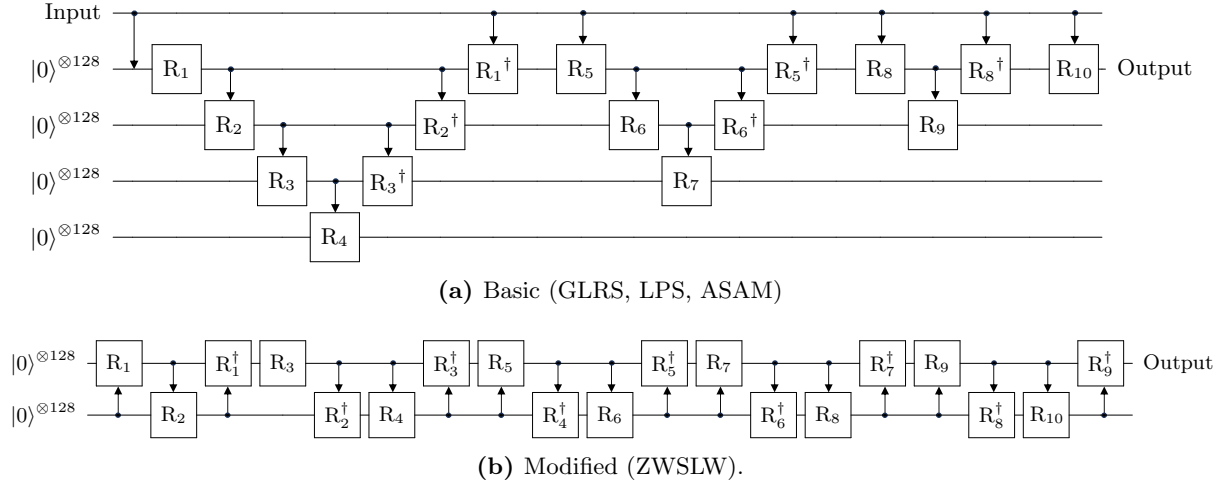
Figure 4: Zig-zag architecture for AES-128 quantum circuit.

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 [44]. It is already well-known the resource estimation in their paper was incorrect due to some problem in Q# (unrelated to the coding of [44]), as already indicated by at least two previous works [36, 77][19] as well as acknowledged by the Eurocrypt'20 authors[20] themselves[21]. 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 [44] by porting their codes to ProjectQ.

5.1 Issues with Q#

Non-parallelizable SubBytes. In their implementation, the S-box of [16] is adopted and ported to the quantum domain. The quantum resources required for the S-box quantum circuit reported in the Eurocrypt'20 paper [44, Table 1] are only correct for the stand-alone S-box (except for T-depth, this will be described in Section 5.1). 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 2(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 stand-alone 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 applies to the SubWord implementation of key schedule.

Issue with AND Gate. This issue is also found in their use of AND gates. Suppose that 5 Toffoli gates are operated in parallel during the S-box process. Toffoli gates (the method used in [3]) 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 6(a), an ancilla qubit (bottom line in Figure 6(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 a nutshell, 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 [43] that the Eurocrypt'23 authors themselves fixed, a more efficient AND† gate (Figure 6(b)) was introduced. However, in this paper the AND† gate from [44] (Figure 6(c)) is used to fix the bug of [44].

---

[19]The same bug appeared in context of another cipher, as mentioned in [39].

[20]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.

[21]As noted in Section 1, the authors recently updated their own bug-fixing in [43].

SB: SubBytes

SB†: Clean ancilla qubits used in preceding SubBytes

R: MixColumns → ShiftRows → AddRoundKey (except for last round)
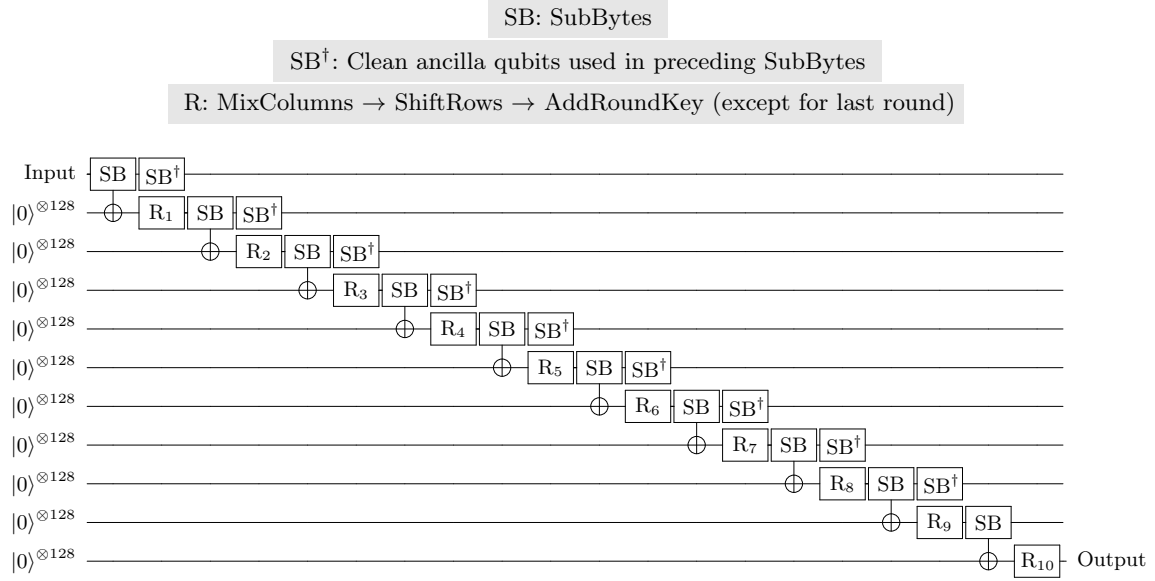
(a) Regular version (originally conceived in JNRV, bug-fixed by us).
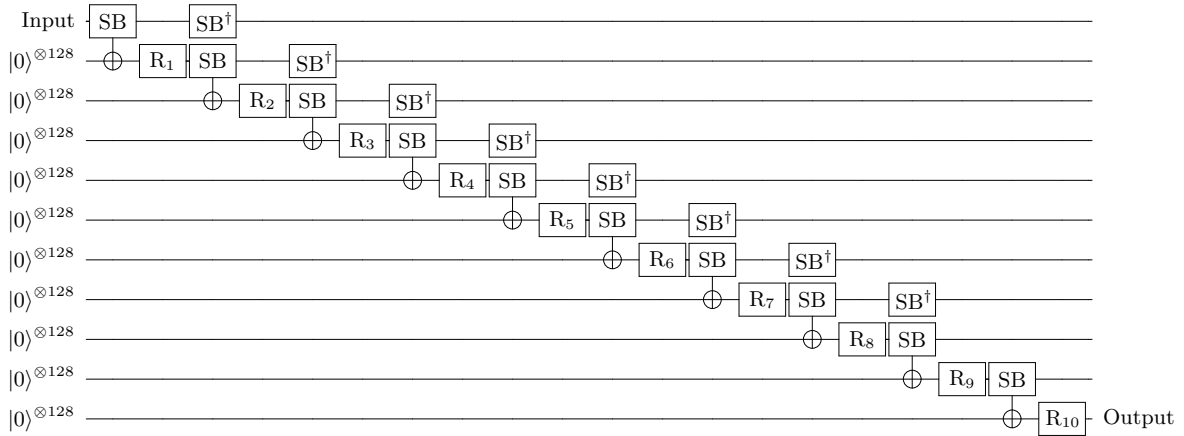
(b) Shallow and shallow/low depth versions (Ours).

Figure 5: Pipeline architecture of AES-128.

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 AES-192 (12 rounds) is higher than that of AES-128 (10 rounds). This case is observed in the zig-zag architecture [32,49] 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 × 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 × 101 × 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 × 101 × 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

(a) AND gate.

(b) AND† gate (Eurocrypt'20).

(c) AND† gate (ePrint'23).
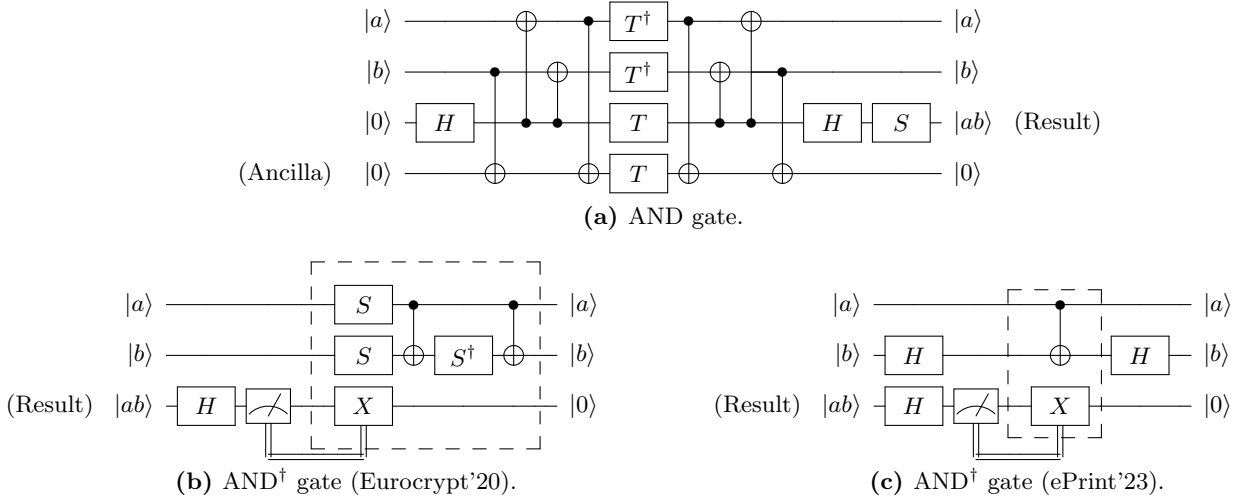
Figure 6: Quantum AND and AND† gates in JNRV.

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 × 111 × 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 [44]).

5.2 Architecture Consideration

If we correct the Eurocrypt'20 implementation [44] 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 [58], 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 [43] 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 [74] rather than an out-of-place MixColumn implementation [50]. 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 $TD\text{-}M$ and $FD\text{-}M$ 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.

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. 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 2(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 [44]. As shwon in Figure 2(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 [44], the authors' design philosophy focuses on optimizing depth. Thus, we believe that this modification (✱ fixed qubit count, increasing qubits to achieve low depth) aligns correctly with their intention.

2. We correct the implementation of MixColumns where the same issue occurs. In Eurocrypt'20 paper [44], two MixColumn implementations were presented. The in-place method of MixColumn implementation (which uses PLU decomposition, and derived by the authors themselves [44]) does not cause this issue. On the other hand, similar to S-box, the same issue applies to the MixColumn implementation by Maximov [58], 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 [44] 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 [3]) 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 [44], 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 [44], 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.

In the fixed qubit count version (✱), the qubit count is increased to enable the simultaneous execution of all S-boxes and MixColumns operations. This apprach embodies the design philosophy of minimizing depth by increasing the qubit count, as outlined by the authors of [44], and conceptually aligns with our regular version (✩).

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 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 [1,32], the Itoh–Tsujii based inversion is implemented on a quantum circuit,

Table 6: Reported benchmarks for JNRV (Eurocrypt'20) implementation of AES.

(a) AES-128 gate costs.

Method	S-box	MixColumn implementation	
	(SubByte)	In-place [44] ✚	Maximov [58] ✛
#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 [44] ✚			Maximov's MixColumn [58] ✛		
	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

so many resources are used for SubBytes. In [49, 77], more efficient quantum circuits are implemented by extending the S-box of [16], 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 [77], 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 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 the next Section 7). 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 [44], the quantum resources required to implement quantum circuits for AES were also estimated. However, there seem to be some issues with Q#'s `ResourcesEstimator`[22] used in their work, specially in implementing quantum circuits for SubBytes. Therefore, the results of [44] are not used here. In the NCT level analysis, replacing Toffoli gates with AND gates does not make much sense. As decomposition-based estimation is meaningful, we compare the required quantum resources by decomposing Toffoli and AND gates. Similar to [77, Table 10], Table 9 shows the detailed quantum resources by decomposing Toffoli gates (Table 9(a)) and AND gates taken from [43] (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 [3]. 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† gate (Figure 6(c)) is decomposed into (5 Clifford gates + 1 Measurement gate), and incurs full depth of 4.

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, AAA ancilla qubits are allocated for replacement.

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 [58] ✛	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 [58] ✛	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 [58] ✛	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 [58] ✛	1248	0	0	0	0	504	22
One round✝	15392	5519	2720	680	6	2912	123

✳: Bug-fixed depth.   ✲: Bug-fixed qubit count.
✛: Maximov's MixColumn [58].
✝: One typical round (that includes MixColumn).

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 [6,32,49]. 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. [32] and Langenberg et al.'s AES paper [49], the authors added $32n - 84$ T gates to their estimate for the n-multi controlled NOT gate [70]. 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 [32,49]. Also in [44], the estimation of the n-multi controlled NOT gate was totally ignored. So, for the n-multi controlled NOT gate, we estimate the number of T gates to be ($32n - 84$) according to the decomposition method in [70] and T-depth is maintained.

---

[22]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

In quantum exhaustive key search, to recover a unique key, not a spurious key, Grassl et al. in [32] estimated the attack cost for r known (plaintext, ciphertext) pairs ($r = 3$, $r = 4$ and $r = 5$, respectively). Later in [49], 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 [44] also estimated the cost for the same r (plaintext, ciphertext) pairs in [49] 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 (✳ $\times$ ✳).

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 has the lowest attack complexity (circuit size). However, when considering the more realistic metric of $FD\text{-}M$ cost, the regular version using S-box with Toffoli depth 4 shows the highest efficiency. If the T-depth metric for error correction takes priority (i.e., $Td\text{-}M$ cost), then the shallow version using S-box with Toffoli depth 4 is the optimal choice (although it is not shown in Tables 10(a) and 10(b), it can be found in Tables 9(a) and 9(b)). In contrast to AES-128, AES-192 and AES-256 require parallelization of the Grover' search due to the MAXDEPTH limitation. As specified in Section 2.4, parallelizing Grover's search is highly inefficient, and in such cases, we should minimize $FD^2\text{-}M$ and $Td^2\text{-}M$ costs (i.e., Cost under MAXDEPTH in Table 10). Therefore, under the MAXDEPTH limit, the shallow or the shallow/low depth version using S-box with Toffoli depth 4 is the most efficient in terms of attack complexity ($FD\text{-}G$ cost) and the realistic metric ($FD^2\text{-}M$). If we consider the T-depth ($Td^2\text{-}M$), then the shallow version using S-box with Toffoli depth 3 is the optimal choice. When we replace Toffoli gates with AND gates (as shown in Table 10(b)), $Td^2\text{-}M$ of the regular version using S-box with Toffoli depth 3 is the lowest. This is because the AND† gate does not use any T gate (see Figure 6).

Additionally, a quick comparison of NIST's security level[23] (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 product of decomposed (Clifford and T) gate count and full depth. Also, the MAXDEPTH constraint (see Section 2.4) is not considered in

---

[23]One may note that the number of qubits was not included in NIST's estimation, probably because the researchers in NIST were more focused on gates and depths that increase drastically with the number of serial steps needed in the Grover's search.

(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

✳: Bug-fixed depth.   ✿: Bug-fixed qubit count.
✚: In-place MixColumn [44].   ✦: Maximov's MixColumn [58].

the computation. For instance, the figure of $2^{156.2654}$ corresponding to the shallow/low (◎) version 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 designs (which achieves noticably lower cost) as well as S-box implementations.

Among other results, we show the least Toffoli depth (✦) and full depth (✳) implementations of all variants of AES (more than 98% and 95% improvement from [77] and [36] respectively). At the same time, we improve the Toffoli depth - qubit count product (✦ × ✿) by more than 75% and 30%, and more than 84% and 99% in the Toffoli depth-square - qubit count product compared to the respective papers. A bird's-eye view can be seen from Figure 7, where we show our work contributes in lowering the quantum circuit complexity (in terms of qubit count and full depth) compared to GLRS [32] and LPS [49]. In total, we present 20 implementations per variant of AES (including bug-fixing of [44]), each incorporating a special design idea/optimization. We show improvement over the recent papers, including Asiacrypt'22 [36] and Asiacrypt'23 [54]. 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 [1, 32, 49, 69, 77]. 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.

Finding optimizations for the cipher building blocks can be considered among the top priorities for the future research works. As far as we can tell, there is a vacant niche for a tool that can efficiently find such implementation for 8×8 S-boxes. The tools described in [21, 22, 23] can possibly be considered as starting points. Besides, the idea in [24] 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., [65]) can also be considered in the future scope. Other than that, one may also consider implementation of combined components (as one 128×128 binary matrix), such as; 4 combined MixColumn's (128×128 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

Table 9: Quantum circuit resources required for variants of AES (this work).

(a) Using Toffoli gate.

AES			#CNOT	#1qCliff	#T	T-depth	#qubit (M)	Full depth	Td-M cost (Td × M)	FD-M cost (FD × M)
			❄	❄	✛	♣	❂	❄	♣ × ❂	❄ × ❂
128	☆	♣	138360	15496	87920	304	2896	1090	880384	3156640
	◎		138148	19096	86660	160	3428	731	548480	2505868
	◈		143044	19096	86660	160	4708	667	753280	3140236
	☆	❀	298536	38496	207480	228	4256	1069	970368	4549664
	◎		296396	39096	206220	120	6288	710	754560	4464480
	◈		301292	39096	206220	120	7568	647	908160	4896496
192	☆	♣	156400	17360	99008	368	3216	1294	1183488	4161504
	◎		156008	21272	98000	192	3748	874	719616	3275752
	◈		161992	21272	98000	192	5284	797	1014528	4211348
	☆	❀	336832	43192	233688	276	4576	1270	1262976	5811520
	◎		334896	43864	232680	144	6608	850	951552	5616800
	◈		340880	43864	232680	144	8144	773	1172736	6295312
256	☆	♣	193548	21415	123032	432	3536	1516	1527552	5360576
	◎		191772	26607	122024	224	4036	1025	904064	4136900
	◈		198844	26607	122024	224	5828	934	1305472	5443352
	☆	❀	417868	53383	290472	324	4896	1488	1586304	7285248
	◎		414568	53983	289212	168	6896	997	1158528	6875312
	◈		421640	53983	289212	168	8688	907	1459584	7880016

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

♣: S-box with Toffoli depth 4.
❀: S-box with Toffoli depth 3.

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 [36, 51, 52, 54, 74]. This enables us to pick the suitable candidates for each building block. On top of that, we devote in finding the optimum architecture, whence we propose/categorize 3 versions. Indeed, the shallow architecture (◎) proposed by us was used in [54]. We present three new implementations for the AES S-box and one new implementation for the AESMixColumn. 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 [59, 60], the bug and bug-fix of JNRV (Eurocrypt'20) [44], or the depth/quabit reduction (Appendix D); which we expect to be useful to future researchers.

A   Concise Description of AES Variants

The Advanced Encryption Standard (AES) [20] is an SPN block cipher family with a block of 128 bits. The state of AES is arranged as a 4×4 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 found 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 8×8 S-box.

ShiftRows. This operation cyclically rotates the rth row of state to the left by i places, for $i = 0, 1, 2, 3$.

(b) Using AND gate.

AES		#CNOT ✳	#1qCliff ❄	#T ✛	#Measure ☆	T-depth ♣	#qubit (M) ✿	Full depth ✳	Td-M cost (Td × M) ♣ × ✿	FD-M cost (FD × M) ✳ × ✿
128	☆	121160	40296	27200	5760	40	3256	826	130240	2689456
	◎ ♣	122608	39936	27200	5580	40	3828	711	153120	2721708
	◈	127504	39936	27200	5580	40	5108	647	204320	3304876
	☆	259536	94056	62400	14040	30	4976	895	149280	4453520
	◎ ❀	258296	93456	62400	13860	30	7008	680	210240	4765440
	◈	263192	93456	62400	13860	30	8288	617	248640	5113696
192	☆	136752	45376	30464	6528	48	3576	972	171648	3475872
	◎ ♣	137936	45088	30464	6384	48	4148	850	199104	3525800
	◈	143920	45088	30464	6384	48	5684	773	272832	4393732
	☆	292216	105952	69888	15912	36	5296	1058	190656	5603168
	◎ ❀	291000	105472	69888	15768	36	7328	814	263808	5964992
	◈	296984	105472	69888	15768	36	8864	737	319104	6532768
256	☆	168596	56399	37536	8192	56	3896	1136	218176	4425856
	◎ ♣	168604	56111	37536	8048	56	4436	997	248416	4422692
	◈	175676	56111	37536	8048	56	6228	906	348768	5642568
	☆	361084	131743	86112	19968	42	5616	1238	235872	6952608
	◎ ❀	358684	131143	86112	19788	42	7616	955	319872	7273280
	◈	365756	131143	86112	19788	42	9408	865	395136	8137920

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

♣: S-box with Toffoli depth 4.
❀: S-box with Toffoli depth 3.

MixColumns. The MixColumn operation pre-multiplies each of the state column with the right circulant matrix $(02, 03, 01, 01)$, over $\text{GF}(2^8)[x]$ 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 32×32 (one may refer to [8, Chapter 2.4.1] for the binary matrix form).

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 rth 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

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

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

For AES-192, the word W_i is generated by

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

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 × G) ✳ × ✳	FD-M cost (FD × M) ✳ × ⚙	FD²-M ✳² × ⚙	Td²-M ✣² × ⚙
128	☆	1	2,897	$1.460 \cdot 2^{82}$	$1.672 \cdot 2^{74}$	$1.221 \cdot 2^{157}$	$1.18 \cdot 2^{86}$	$1.98 \cdot 2^{160}$	$1.23 \cdot 2^{157}$
	◎ ♣		3,429	$1.473 \cdot 2^{82}$	$1.121 \cdot 2^{74}$	$1.651 \cdot 2^{156}$	$\mathbf{1.88 \cdot 2^{85}}$	$\mathbf{1.05 \cdot 2^{160}}$	$\mathbf{1.61 \cdot 2^{155}}$
	◈		4,709	$1.502 \cdot 2^{82}$	$1.022 \cdot 2^{74}$	$\mathbf{1.536 \cdot 2^{156}}$	$1.17 \cdot 2^{86}$	$1.20 \cdot 2^{160}$	$1.11 \cdot 2^{156}$
	☆		4,257	$1.637 \cdot 2^{83}$	$1.64 \cdot 2^{74}$	$1.342 \cdot 2^{158}$	$1.7 \cdot 2^{86}$	$1.4 \cdot 2^{161}$	$1.02 \cdot 2^{157}$
	◎ ✿		6,289	$1.629 \cdot 2^{83}$	$1.089 \cdot 2^{74}$	$1.773 \cdot 2^{157}$	$1.67 \cdot 2^{86}$	$1.82 \cdot 2^{160}$	$1.66 \cdot 2^{155}$
	◈		7,569	$1.643 \cdot 2^{83}$	$1.984 \cdot 2^{73}$	$1.63 \cdot 2^{157}$	$1.83 \cdot 2^{86}$	$1.82 \cdot 2^{160}$	$1.99 \cdot 2^{155}$
192	☆	2	6,241	$1.53 \cdot 2^{115}$	$1.984 \cdot 2^{106}$	$1.518 \cdot 2^{222}$	$1.51 \cdot 2^{119}$	$1.5 \cdot 2^{226}$	$1.94 \cdot 2^{222}$
	◎ ♣		7,305	$1.539 \cdot 2^{115}$	$1.34 \cdot 2^{106}$	$1.031 \cdot 2^{222}$	$\mathbf{1.19 \cdot 2^{119}}$	$\mathbf{1.6 \cdot 2^{225}}$	$\mathbf{1.23 \cdot 2^{221}}$
	◈		10,377	$1.575 \cdot 2^{115}$	$1.222 \cdot 2^{106}$	$\mathbf{1.924 \cdot 2^{221}}$	$1.55 \cdot 2^{119}$	$1.89 \cdot 2^{225}$	$1.75 \cdot 2^{221}$
	☆		8,961	$1.715 \cdot 2^{116}$	$1.947 \cdot 2^{106}$	$1.67 \cdot 2^{223}$	$1.06 \cdot 2^{120}$	$1.04 \cdot 2^{227}$	$1.56 \cdot 2^{222}$
	◎ ✿		13,025	$1.706 \cdot 2^{116}$	$1.304 \cdot 2^{106}$	$1.112 \cdot 2^{223}$	$1.04 \cdot 2^{120}$	$1.35 \cdot 2^{226}$	$1.24 \cdot 2^{221}$
	◈		16,097	$1.724 \cdot 2^{116}$	$1.186 \cdot 2^{106}$	$1.022 \cdot 2^{223}$	$1.17 \cdot 2^{120}$	$1.38 \cdot 2^{226}$	$1.53 \cdot 2^{221}$
256	☆	2	6,817	$1.847 \cdot 2^{147}$	$1.163 \cdot 2^{139}$	$1.074 \cdot 2^{287}$	$1.94 \cdot 2^{151}$	$1.12 \cdot 2^{291}$	$1.46 \cdot 2^{287}$
	◎ ♣		7,817	$1.859 \cdot 2^{147}$	$1.572 \cdot 2^{138}$	$1.461 \cdot 2^{286}$	$\mathbf{1.5 \cdot 2^{151}}$	$\mathbf{1.18 \cdot 2^{290}}$	$1.79 \cdot 2^{285}$
	◈		11,401	$1.901 \cdot 2^{147}$	$1.433 \cdot 2^{138}$	$\mathbf{1.362 \cdot 2^{286}}$	$1.99 \cdot 2^{151}$	$1.43 \cdot 2^{290}$	$1.31 \cdot 2^{286}$
	☆		9,537	$1.037 \cdot 2^{149}$	$1.141 \cdot 2^{139}$	$1.183 \cdot 2^{288}$	$1.33 \cdot 2^{152}$	$1.52 \cdot 2^{291}$	$1.15 \cdot 2^{287}$
	◎ ✿		13,537	$1.031 \cdot 2^{149}$	$1.529 \cdot 2^{138}$	$1.577 \cdot 2^{287}$	$1.26 \cdot 2^{152}$	$1.93 \cdot 2^{290}$	$\mathbf{1.74 \cdot 2^{285}}$
	◈		17,121	$1.042 \cdot 2^{149}$	$1.391 \cdot 2^{138}$	$1.449 \cdot 2^{287}$	$1.45 \cdot 2^{152}$	$1.01 \cdot 2^{291}$	$1.1 \cdot 2^{286}$

☆: Regular version.
◎: Shallow version.
◈: Shallow/low depth version.
♣: S-box with Toffoli depth 4.
✿: S-box with Toffoli depth 3.

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.3 Notations Related to Singular and Plural Forms

The AES state is represented as a 4×4 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 32×32 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 4×4 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 32×32, MixColumns can be represented as the diagonal matrix (M, M, M, M) of dimension 128×128 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 ith row to the left by i bytes, where $i = 1, 2, 3$. Thus, one ShiftRows operation is equivalent to $4\times$ ShiftRow operations on different rows in the 4×4 matrix with the shift parameter varies from 0 to 3.

The SubBytes in the round function updates every byte in the 4×4 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 4×4 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 4×4 matrix.

(b) Using AND gate.

AES		r	#qubit (M) ♻	Total gates (G) ✳	Full depth (FD) ❄	FD-G cost (FD × G) ❄ × ✳	FD-M cost (FD × M) ❄ × ♻	Cost under MAXDEPTH FD²-M ❄² × ♻	Td²-M ♣² × ♻
128	☆	1	3,257	$1.176 \cdot 2^{82}$	$1.267 \cdot 2^{74}$	$1.49 \cdot 2^{156}$	$\mathbf{1.01 \cdot 2^{86}}$	$1.28 \cdot 2^{160}$	$1.49 \cdot 2^{151}$
	◎ ♣		3,829	$1.182 \cdot 2^{82}$	$1.09 \cdot 2^{74}$	$1.288 \cdot 2^{156}$	$1.02 \cdot 2^{86}$	$\mathbf{1.11 \cdot 2^{160}}$	$1.75 \cdot 2^{151}$
	◈		5,109	$1.211 \cdot 2^{82}$	$1.984 \cdot 2^{73}$	$\mathbf{1.202 \cdot 2^{156}}$	$1.24 \cdot 2^{86}$	$1.23 \cdot 2^{160}$	$1.17 \cdot 2^{152}$
	☆		4,977	$1.294 \cdot 2^{83}$	$1.372 \cdot 2^{74}$	$1.776 \cdot 2^{157}$	$1.67 \cdot 2^{86}$	$1.14 \cdot 2^{161}$	$\mathbf{1.31 \cdot 2^{151}}$
	◎ ❀		7,009	$1.288 \cdot 2^{83}$	$1.043 \cdot 2^{74}$	$1.343 \cdot 2^{157}$	$1.78 \cdot 2^{86}$	$1.86 \cdot 2^{160}$	$1.85 \cdot 2^{151}$
	◈		8,289	$1.303 \cdot 2^{83}$	$1.893 \cdot 2^{73}$	$1.233 \cdot 2^{157}$	$1.92 \cdot 2^{86}$	$1.81 \cdot 2^{160}$	$1.09 \cdot 2^{152}$
192	☆	2	7,161	$1.253 \cdot 2^{115}$	$1.644 \cdot 2^{106}$	$1.03 \cdot 2^{222}$	$1.437 \cdot 2^{119}$	$1.181 \cdot 2^{226}$	$1.198 \cdot 2^{217}$
	◎ ♣		8,105	$1.231 \cdot 2^{115}$	$1.304 \cdot 2^{106}$	$1.605 \cdot 2^{221}$	$\mathbf{1.29 \cdot 2^{119}}$	$\mathbf{1.68 \cdot 2^{225}}$	$1.36 \cdot 2^{217}$
	◈		11,177	$1.267 \cdot 2^{115}$	$1.186 \cdot 2^{106}$	$\mathbf{1.502 \cdot 2^{221}}$	$1.62 \cdot 2^{119}$	$1.92 \cdot 2^{225}$	$1.87 \cdot 2^{217}$
	☆		10,401	$1.353 \cdot 2^{116}$	$1.622 \cdot 2^{106}$	$1.097 \cdot 2^{223}$	$1.03 \cdot 2^{120}$	$1.67 \cdot 2^{226}$	$\mathbf{1.94 \cdot 2^{216}}$
	◎ ❀		14,465	$1.346 \cdot 2^{116}$	$1.248 \cdot 2^{106}$	$1.68 \cdot 2^{222}$	$1.1 \cdot 2^{120}$	$1.38 \cdot 2^{226}$	$1.35 \cdot 2^{217}$
	◈		17,537	$1.364 \cdot 2^{116}$	$1.13 \cdot 2^{106}$	$1.541 \cdot 2^{222}$	$1.21 \cdot 2^{120}$	$1.37 \cdot 2^{226}$	$1.64 \cdot 2^{217}$
256	☆	2	7,537	$1.482 \cdot 2^{147}$	$1.742 \cdot 2^{138}$	$1.291 \cdot 2^{286}$	$\mathbf{1.6 \cdot 2^{151}}$	$1.4 \cdot 2^{290}$	$1.7 \cdot 2^{281}$
	◎ ♣		8,617	$1.479 \cdot 2^{147}$	$1.529 \cdot 2^{138}$	$1.131 \cdot 2^{286}$	$1.61 \cdot 2^{151}$	$\mathbf{1.23 \cdot 2^{290}}$	$1.94 \cdot 2^{281}$
	◈		12,201	$1.521 \cdot 2^{147}$	$1.39 \cdot 2^{138}$	$\mathbf{1.057 \cdot 2^{286}}$	$1.04 \cdot 2^{152}$	$1.44 \cdot 2^{290}$	$1.38 \cdot 2^{282}$
	☆		10,977	$1.632 \cdot 2^{148}$	$1.898 \cdot 2^{138}$	$1.549 \cdot 2^{287}$	$1.27 \cdot 2^{152}$	$1.21 \cdot 2^{291}$	$\mathbf{1.38 \cdot 2^{281}}$
	◎ ❀		14,977	$1.623 \cdot 2^{148}$	$1.465 \cdot 2^{138}$	$1.189 \cdot 2^{287}$	$1.34 \cdot 2^{152}$	$1.96 \cdot 2^{290}$	$1.89 \cdot 2^{281}$
	◈		18,561	$1.644 \cdot 2^{148}$	$1.326 \cdot 2^{138}$	$1.09 \cdot 2^{287}$	$1.5 \cdot 2^{152}$	$1.99 \cdot 2^{290}$	$1.17 \cdot 2^{282}$

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

♣: S-box with Toffoli depth 4.
❀: S-box with Toffoli depth 3.

A.4 Notations Related to Reverse Computation

S-box and S-box† in quantum. S-box in quantum denotes before storing values from ancilla qubits to output qubits. Denote the reverse operation of S-box as S-box† and uses input qubits to clean up ancilla qubits.

SubBytes and SubBytes† in quantum. SubBytes of AES in quantum denotes parallel operation for 16 S-boxes. Denote the reverse operation of SubBytes as SubBytes† and cleans up all used ancilla qubits in 16 S-boxes.

Rotation and Rotation† in quantum. Rotation of AES in quantum denotes the same RotWord. The reverse operation of Rotation is denoted as Rotation†.

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

B Novelty/New Construction

We basically collect practically all the relevant research works and collate in one place. Our coverage of the literature includes papers as recent as [51, 52, 54, 74]. 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 new 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 [36, 54]. Indeed, we currently hold the record for the least full depth implementation of the AES S-box (56, reduced from 69 as reported in [36]), to the best of our knowledge. Of the three new S-box implementations, two are used in this work (❀, ♣).
2. We propose our new (out-of-place) implementation of MixColumn that takes only 8 quantum depth, and use it in our shallow/low depth (◈) implementation. From what we found, this is the least quantum depth for the AES MixColumn implementation reported so far. Further, the required number of ancilla qubits is only 32, which is the fewest among the out-of-place quantum MixColumn implementations (gate count also). The idea behind the quantum depth reduction and ancilla qubit reduction is described in Appendix D.

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

Level (AES)	N$^+$'16 [59] (G$^+$ [32])	L$^+$ [49]	N$^+$'22 [60]	This work						
				☆	◎	◈	❋✳ (Tof)	❋✱ (Tof)	❋✳ (AND)	❋✱ (AND)
1 (128)	2^{170} ($2^{168.6683}$)	$2^{162.6093}$	2^{157}	✿: $2^{156.5753}$	✿: $2^{156.3651}$	✿: $\mathbf{2^{156.2654}}$	➕: $2^{162.3577}$	➕: $2^{158.1337}$	➕: $2^{161.6042}$	➕: $2^{157.9949}$
				❀: $2^{157.8286}$	❀: $2^{157.4255}$	❀: $2^{157.3022}$	✛: $2^{162.5641}$	✛: $2^{157.9591}$	✛: $2^{161.6510}$	✛: $2^{157.3327}$
3 (192)	2^{233} ($2^{233.4645}$)	$2^{227.6491}$	2^{221}	✿: $2^{222.0426}$	✿: $2^{221.6826}$	✿: $\mathbf{2^{221.5869}}$	➕: $2^{227.5867}$	➕: $2^{223.4821}$	➕: $2^{226.9368}$	➕: $2^{223.4355}$
				❀: $2^{223.1336}$	❀: $2^{222.7485}$	❀: $2^{222.6239}$	✛: $2^{227.6260}$	✛: $2^{223.3002}$	✛: $2^{226.9885}$	✛: $2^{222.7643}$
5 (256)	2^{298} ($2^{298.3467}$)	$2^{292.3100}$	2^{285}	✿: $2^{286.3685}$	✿: $2^{286.1776}$	✿: $\mathbf{2^{286.0800}}$	➕: $2^{292.1520}$	➕: $2^{287.9871}$	➕: $2^{291.5326}$	➕: $2^{287.9595}$
				❀: $2^{287.6313}$	❀: $2^{287.2497}$	❀: $2^{287.1243}$	✛: $2^{292.1900}$	✛: $2^{287.7966}$	✛: $2^{291.5822}$	✛: $2^{287.2801}$

☆: Regular version (using AND gate).
◎: Shallow version (using AND gate).
◈: Shallow/low depth version (using AND gate).

✿: S-box with Toffoli depth 4.
❀: S-box with Toffoli depth 3.

❋: Bug-fixed JNRV [44] (using S-box from [17] ✛).

✳: Bug-fixed depth.   ✱: Bug-fixed qubit count.

➕: In-place MixColumn [44].   ✛: Maximov's MixColumn [58].

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 5(b)), which reduces the T-depth and full depth from the previous works (as in Figure 5(a)). This involves combining the previous round's reverse operation with the current round's operation by using two alternate ancilla sets.

5. We propose five new architecture in this paper:

    (a) The regular (☆) architecture was originally conceived by Jaques et al. in Eurocrypt'20 [44]. 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 (☆) by patching that 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 [54] directly 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 [44], ancilla qubits are not free when the Q# bug is patched). Similarly, for SubBytes† 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 discussed in [54]).

    (c) The bug-fixing of [44] 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 [44] improves from the authors' own bug-fixed benchmark presented recently in [43].

We present 60 implementations for the AES variants in this work:

(i) **Ours:** 3 architecture (☆, ◎, ◈) × 2 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).

Besides, the Asiacrypt'23 authors [54] acknowledged in a private communication that our work served as the inspiration for their work: "*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 [44]. We encountered two issues. First (non-parallelizable) and second (issue with AND gate) problems analyzed in Section 5.1 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 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 [44]. Only 234 should be derived as depth for SubBytes × 14 rounds. This depth margin, therefore, cannot be derived even with excessive parallelization.

The Q# compiler finds non-trivial parallelism in the circuit, but according to our examples, this parallelism is excessive in the Eurocrypt'20 paper [44]. In our case also, the estimated depth of the circuit is slightly reduced,
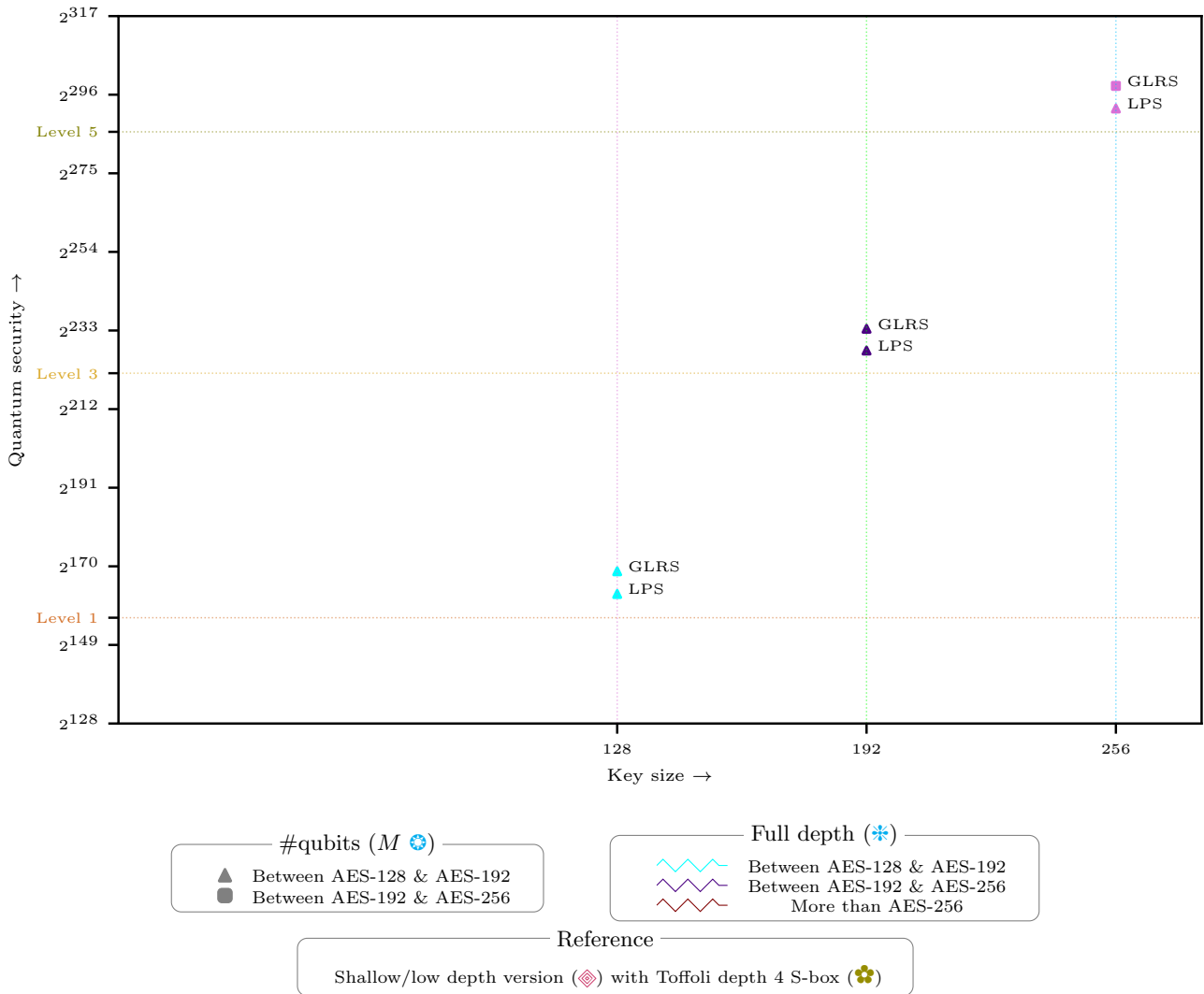
Figure 7: Comparison of quantum circuit complexities for AES variants.

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 → MixColumns → SubBytes → 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 [44]. 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) × 111, the full depth is already 1443. Then only 234 (= 1677 − 1443) should be derived as depth for SubBytes × 14 rounds. Therefore, the full depth derived from Sbox in each round should be only about 17 (= 234 ÷ 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 [58]) reported in JNRV [44] 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).

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

$k_{i,j}^r$ represents the ith byte in the jth *word* of the rth round key

(b) Key schedule.
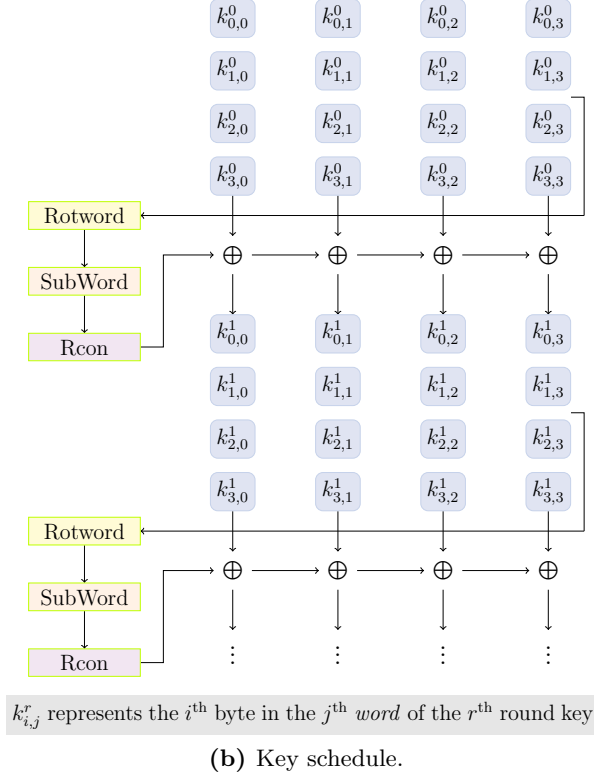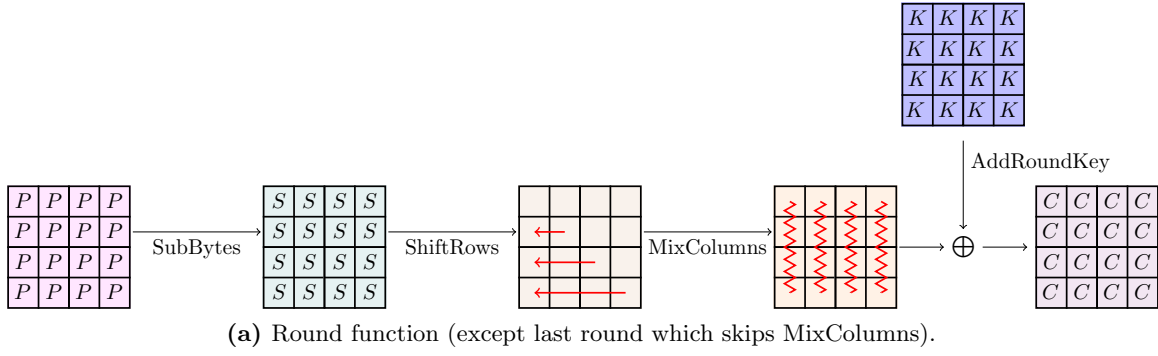
Figure 8: Schematic of AES encryption.

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 [74] 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 [54] implementation of the S-box is given in [54, 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.

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

(a) Initial target (9 full depth)

```
1   def Reorder_0 (a, b): # Full depth : 9
2       CNOT | (a[0], a[11])
3       CNOT | (a[3], a[11])
4
5
6       CNOT | (a[0], a[4])
7       CNOT | (a[0], a[5])
8       CNOT | (a[0], a[6])
9       CNOT | (a[0], a[7])
10
11      CNOT | (a[1], a[4])
12      CNOT | (a[1], a[5])
13      CNOT | (a[1], a[6])
14      CNOT | (a[1], a[7])
15
16      CNOT | (a[2], a[4])
17      CNOT | (a[2], a[5])
18      CNOT | (a[2], a[6])
19      CNOT | (a[2], a[7])
20      CNOT | (a[2], a[8])
21
22      X | a[6]
23      X | a[8]
24      X | a[10]
25      X | a[11]
```

(b) First reordering (8 full depth)

```
1   def Reorder_1 (a, b): # Full depth : 8
2       CNOT | (a[0], a[4])
3       CNOT | (a[0], a[5])
4       CNOT | (a[0], a[6])
5       CNOT | (a[0], a[7])
6
7       CNOT | (a[1], a[4])
8       CNOT | (a[1], a[5])
9       CNOT | (a[1], a[6])
10      CNOT | (a[1], a[7])
11
12      CNOT | (a[2], a[4])
13      CNOT | (a[2], a[5])
14      CNOT | (a[2], a[6])
15      CNOT | (a[2], a[7])
16      CNOT | (a[2], a[8])
17
18      CNOT | (a[0], a[11]) # Reordered 1
19      CNOT | (a[3], a[11]) # Reordered 1
20
21      X | a[6]
22      X | a[8]
23      X | a[10]
24      X | a[11]
```

(c) Second reordering (7 full depth)

```
1   def Reorder_2 (a, b): # Full depth : 7
2       X | a[6]  # Reordered 2
3       X | a[8]  # Reordered 2
4       X | a[10] # Reordered 2
5       X | a[11] # Reordered 2
6
7       CNOT | (a[0], a[4])
8       CNOT | (a[0], a[5])
9       CNOT | (a[0], a[6])
10      CNOT | (a[0], a[7])
11
12      CNOT | (a[1], a[4])
13      CNOT | (a[1], a[5])
14      CNOT | (a[1], a[6])
15      CNOT | (a[1], a[7])
16
17      CNOT | (a[2], a[4])
18      CNOT | (a[2], a[5])
19      CNOT | (a[2], a[6])
20      CNOT | (a[2], a[8])
21      CNOT | (a[2], a[7])
22
23      CNOT | (a[0], a[11]) # Reordered 1
24      CNOT | (a[3], a[11]) # Reordered 1
```

(d) Third reordering (6 full depth)

```
1   def Reorder_3 (a, b): # Full depth : 6
2       X | a[6]  # Reordered 2
3       X | a[8]  # Reordered 2
4       X | a[10]  # Reordered 2
5       X | a[11]  # Reordered 2
6
7       CNOT | (a[0], a[4])
8       CNOT | (a[0], a[5])
9       CNOT | (a[0], a[6])
10      CNOT | (a[0], a[7])
11      CNOT | (a[2], a[8]) # Reordered 3
12
13      CNOT | (a[1], a[4])
14      CNOT | (a[1], a[5])
15      CNOT | (a[1], a[6])
16      CNOT | (a[1], a[7])
17
18      CNOT | (a[2], a[4])
19      CNOT | (a[2], a[5])
20      CNOT | (a[2], a[6])
21      CNOT | (a[2], a[7])
22
23      CNOT | (a[0], a[11])  # Reordered 1
24      CNOT | (a[3], a[11])  # Reordered 1
```

As it can be seen, Code 1(a) ⤳ Code 1(b) ⤳ Code 1(c) ⤳ Code 1(d) show the progression of depth reduction (9 ⤳ 8 ⤳ 7 ⤳ 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 2 and 3 (Code 1(a)) to Lines 18 and 19 (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 21, 22, 23 and 24 in Code 1(b)) to the top (see Code 1(c); Lines 2, 3, 4 and 5).

In the third reordering step, similar to the first reordering (Code 1(b)), Line 20 ("`CNOT | (a[2], a[8])`"), successive calls to `a[2]`) is moved to Line 11 (Code 1(d)). As a result, "`CNOT | (a[2], a[8])`" is synchronized well with the preceding gate operations (i.e., Lines 7, 8, 9 and 10 in Code 1(d)).

With three rounds of randomized reordering, the full depth, which was initially 9, is noticeably reduced to 6. With this randomized greedy approach, we reduced the full depth of S-boxes as presented in [36] 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 applied this method to the out-of-place MixColumn from [50] and achieved the depth reduction (11 ⤳ 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 example, in simple case, 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 Result

Similar to [77, Table 6], we show the per-round benchmark for our implementations of the AES family in Table 12 (using the S-box implementation with Toffoli depth 3 and 4 in Table 12(a) and Table 12(b), respectively).

Table 12: Quantum resources required per round for variants of AES (this work).

(a) Using S-box with Toffoli depth 4.

AES		#CNOT ✳			#NOT ✲	#Toffoli ☆		TD ◆	
❀	Round	☆	◎	◈	☆◎◈	☆	◎◈	☆	◎◈
128	1ℓ	8960	5064	6120	79	1360	680	8	4
	2	8832	8960	10016	79	1360	1360	8	4
	3	8832	8960	10016	81	1360	1360	8	4
	4	8832	8960	10016	81	1360	1360	8	4
	5	8832	8960	10016	81	1360	1360	8	4
	6	8832	8960	10016	79	1360	1360	8	4
	7	8832	8960	10016	79	1360	1360	8	4
	8	8832	8960	10016	81	1360	1360	8	4
	9	8832	8960	10016	80	1360	1360	8	4
	10	4504	4568	4568	80	680	680	4	4
192	1ℓ	9024	9056	10112	79	1360	1360	8	4
	2	8896	8992	10048	79	1360	1360	8	4
	3	7088	7152	8208	64	1088	1088	8	4
	4	8896	8928	9984	81	1360	1360	8	4
	5	8896	8992	10048	81	1360	1360	8	4
	6	7088	7152	8208	64	1088	1088	8	4
	7	8896	8928	9984	81	1360	1360	8	4
	8	8896	8992	10048	79	1360	1360	8	4
	9	7088	7152	8208	64	1088	1088	8	4
	10	8896	8928	9984	79	1360	1360	8	4
	11	8896	5032	6088	81	1360	680	8	4
	12	3552	3552	3552	64	544	544	4	4
256	1ℓ	7216	4048	5104	64	1088	544	8	4
	2	8832	8040	9096	79	1360	1224	8	4
	3	8832	8832	9888	80	1360	1360	8	4
	4	8832	8832	9888	79	1360	1360	8	4
	5	8832	8832	9888	80	1360	1360	8	4
	6	8832	8832	9888	81	1360	1360	8	4
	7	8832	8832	9888	80	1360	1360	8	4
	8	8832	8832	9888	81	1360	1360	8	4
	9	8832	8832	9888	80	1360	1360	8	4
	10	8832	8832	9888	81	1360	1360	8	4
	11	8832	8832	9888	80	1360	1360	8	4
	12	8832	8832	9888	79	1360	1360	8	4
	13	8832	8832	9888	80	1360	1360	8	4
	14	4504	4504	4504	79	680	680	4	4

ℓ: Including initial key XOR.

☆: Regular version.

◎: Shallow version.

◈: Shallow/low depth version.

(b) Using S-box with Toffoli depth 3.

AES	Round	#CNOT ☆	#CNOT ◎	#CNOT ◈	#NOT ☆◎◈	#Toffoli ☆	#Toffoli ◎◈	TD ☆	TD ◎◈
128	1♮	14640	7904	8960	79	3120	1560	6	3
	2	14512	14640	15696	79	3120	3120	6	3
	3	14512	14640	15696	81	3120	3120	6	3
	4	14512	14640	15696	81	3120	3120	6	3
	5	14512	14640	15696	81	3120	3120	6	3
	6	14512	14640	15696	79	3120	3120	6	3
	7	14512	14640	15696	79	3120	3120	6	3
	8	14512	14640	15696	81	3120	3120	6	3
	9	14512	14640	15696	80	3120	3120	6	3
	10	7344	7408	7408	80	1560	1560	3	3
192	1♮	14704	14736	15792	79	3120	3120	6	3
	2	14576	14672	15728	79	3120	3120	6	3
	3	11632	11696	12752	64	2496	2496	6	3
	4	14576	14608	15664	81	3120	3120	6	3
	5	14576	14672	15728	81	3120	3120	6	3
	6	11632	11696	12752	64	2496	2496	6	3
	7	14576	14608	15664	81	3120	3120	6	3
	8	14576	14672	15728	79	3120	3120	6	3
	9	11632	11696	12752	64	2496	2496	6	3
	10	14576	14608	15728	79	3120	3120	6	3
	11	14576	7872	8928	81	3120	1560	6	3
	12	5824	5824	5824	64	1248	1248	3	3
256	1♮	11760	6320	7376	64	2496	1248	6	3
	2	14512	13152	14208	79	3120	2808	6	3
	3	14512	14512	15568	80	3120	3120	6	3
	4	14512	14512	15568	79	3120	3120	6	3
	5	14512	14512	15568	80	3120	3120	6	3
	6	14512	14512	15568	81	3120	3120	6	3
	7	14512	14512	15568	80	3120	3120	6	3
	8	14512	14512	15568	81	3120	3120	6	3
	9	14512	14512	15568	80	3120	3120	6	3
	10	14512	14512	15568	81	3120	3120	6	3
	11	14512	14512	15568	80	3120	3120	6	3
	12	14512	14512	15568	79	3120	3120	6	3
	13	14512	14512	15568	80	3120	3120	6	3
	14	7344	7344	7344	79	1560	1560	3	3

♮: Including initial key XOR.

☆: Regular version.

◎: Shallow version.

◈: Shallow/low depth version.

References

1. Almazrooie, M., Samsudin, A., Abdullah, R., Mutter, K.N.: Quantum reversible circuit of AES-128. Quantum Information Processing **17**(5) (may 2018) 1–30 8, 10, 13, 14, 15, 16, 21, 25

2. Amy, M., Di Matteo, O., Gheorghiu, V., Mosca, M., Parent, A., Schanck, J.: Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In Avanzi, R., Heys, H., eds.: Selected Areas in Cryptography – SAC 2016, Cham, Springer International Publishing (2017) 317–337 6

3. Amy, M., Maslov, D., Mosca, M., Roetteler, M., Roetteler, M.: 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) (Jun 2013) 818–830 5, 10, 18, 21, 22

4. Anand, R., Isobe, T.: Quantum security analysis of rocca. Quantum Inf. Process. **22**(4) (2023) 164 4

5. Anand, R., Maitra, A., Maitra, S., Mukherjee, C.S., Mukhopadhyay, S.: Quantum resource estimation for FSR based symmetric ciphers and related Grover's attacks. In Adhikari, A., Küsters, R., Preneel, B., eds.: INDOCRYPT, Jaipur, India, December 12-15, 2021, Proceedings. Volume 13143 of Lecture Notes in Computer Science., Springer (2021) 179–198 6

6. Anand, R., Maitra, A., Mukhopadhyay, S.: Evaluation of quantum cryptanalysis on SPECK. In Bhargavan, K., Oswald, E., Prabhakaran, M., eds.: Progress in Cryptology – INDOCRYPT 2020, Cham, Springer International Publishing (2020) 395–413 6, 23

7. Anand, R., Maitra, A., Mukhopadhyay, S.: Grover on SIMON. Quantum Information Processing **19**(9) (Sep 2020) http://dx.doi.org/10.1007/s11128-020-02844-w. 6

8. Baksi, A.: Classical and Physical Security of Symmetric Key Cryptographic Algorithms. PhD thesis, School of Computer Science & Engineering, Nanyang Technological University, Singapore (2021) https://dr.ntu.edu.sg/handle/10356/152003. 27

9. Baksi, A., Dasu, V.A., Karmakar, B., Chattopadhyay, A., Isobe, T.: Three input exclusive-or gate support for boyar-peralta's algorithm. In Adhikari, A., Küsters, R., Preneel, B., eds.: Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings. Volume 13143 of Lecture Notes in Computer Science., Springer (2021) 141–158 15

10. Baksi, A., Jang, K., Song, G., Seo, H., Xiang, Z.: Quantum implementation and resource estimates for rectangle and knot. Quantum Information Processing **20**(12) (dec 2021) 6

11. Banik, S., Funabiki, Y., Isobe, T.: Further results on efficient implementations of block cipher linear layers. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **104-A**(1) (2021) 213–225 13, 14, 15

12. Bathe, B.N., Anand, R., Dutta, S.: Evaluation of Grover's algorithm toward quantum cryptanalysis on ChaCha. Quantum Inf. Process. **20**(12) (2021) 394 6

13. Bhattacharjee, D., Chattopadhyay, A.: Depth-optimal quantum circuit placement for arbitrary topologies. arXiv preprint arXiv:1703.08540 (2017) 5

14. Bonnetain, X., Leurent, G., Naya-Plasencia, M., Schrottenloher, A.: Quantum linearization attacks. In Tibouchi, M., Wang, H., eds.: Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I. Volume 13090 of Lecture Notes in Computer Science., Springer (2021) 422–452 6

15. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: Quantum security analysis of AES. IACR Transactions on Symmetric Cryptology **2019**(2) (Jun. 2019) 55–93 6

16. Boyar, J., Peralta, R.: A new combinational logic minimization technique with applications to cryptology. In Festa, P., ed.: Experimental Algorithms, Berlin, Heidelberg, Springer Berlin Heidelberg (2010) 178–189 10, 11, 12, 13, 14, 18, 22

17. Boyar, J., Peralta, R.: A depth-16 circuit for the AES S-box. Cryptology ePrint Archive, Report 2011/332 (2011) https://eprint.iacr.org/2011/332. 3, 9, 10, 11, 12, 13, 30

18. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik **46**(4-5) (Jun 1998) 493–505 6

19. Chauhan, A.K., Sanadhya, S.K.: Quantum resource estimates of grover's key search on aria. In: International Conference on Security, Privacy, and Applied Cryptography Engineering, Springer (2020) 238–258 6

20. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer (2002) 2, 26

21. Dansarie, M.: Cryptanalysis of the SoDark family of cipher algorithms. PhD thesis, Naval Postgraduate School, Dudley Knox Library (2017) https://calhoun.nps.edu/handle/10945/56118. 10, 11, 25

22. Dansarie, M.: sboxgates: A program for finding low gate count implementations of S-boxes. Journal of Open Source Software **6**(62) (2021) 2946 10, 11, 25

23. Dasu, V.A., Baksi, A., Sarkar, S., Chattopadhyay, A.: LIGHTER-R: optimized reversible circuit implementation for sboxes. In: 32nd IEEE International System-on-Chip Conference, SOCC 2019, Singapore, September 3-6, 2019. (2019) 260–265 11, 25

24. Davenport, J.H., Pring, B.: Improvements to quantum search techniques for block-ciphers, with applications to aes. In Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C., eds.: Selected Areas in Cryptography, Cham, Springer International Publishing (2021) 360–384 25

25. Dong, X., Dong, B., Wang, X.: Quantum attacks on some feistel block ciphers. Des. Codes Cryptogr. **88**(6) (2020) 1179–1203 1

26. Dong, X., Sun, S., Shi, D., Gao, F., Wang, X., Hu, L.: Quantum collision attacks on AES-like hashing with low quantum random access memories. In Moriai, S., Wang, H., eds.: ASIACRYPT, South Korea, December 7-11, 2020, Proceedings, Part II. Volume 12492 of Lecture Notes in Computer Science., Springer (2020) 727–757 6

27. Dong, X., Zhang, Z., Sun, S., Wei, C., Wang, X., Hu, L.: Automatic classical and quantum rebound attacks on AES-like hashing by exploiting related-key differentials. In Tibouchi, M., Wang, H., eds.: ASIACRYPT, Singapore, December 6-10, 2021, Proceedings, Part I. Volume 13090 of Lecture Notes in Computer Science., Springer (2021) 241–271 6

28. Ducas, L., Lepoint, E.K.T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium – algorithm specifications and supporting documentation (version 3.1). Technical report, 2021 (2021) https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf. 2

29. Ekdahl, P., Johansson, T., Maximov, A., Yang, J.: A new snow stream cipher called snow-v. IACR Transactions on Symmetric Cryptology **2019**(3) (Sep. 2019) 1–42 14

30. Frixons, P., Naya-Plasencia, M., Schrottenloher, A.: Quantum boomerang attacks and some applications. IACR Cryptol. ePrint Arch. (2022) 60 6

31. Gidney, C.: Halving the cost of quantum addition. Quantum **2** (jun 2018) 74 5

32. Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying Grover's algorithm to AES: Quantum resource estimates. In Takagi, T., ed.: Post-Quantum Cryptography, Cham, Springer International Publishing (2016) 29–43 4, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19, 21, 23, 24, 25, 30

33. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. (1996) 212–219 2, 5

34. Grumbling, E., Horowitz, M.: Quantum Computing: Progress and Prospects. The National Academies Press, Washington DC (2019) https://www.nap.edu/catalog/25196/quantum-computing-progress-and-prospects. 1

35. He, Y., Luo, M.X., Zhang, E., Wang, H.K., Wang, X.F.: Decompositions of n-qubit toffoli gates with linear circuit complexity. International Journal of Theoretical Physics **56**(7) (2017) 2350–2361 5

36. Huang, Z., Sun, S.: Synthesizing quantum circuits of AES with lower t-depth and less qubits. In Agrawal, S., Lin, D., eds.: 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., Springer (2022) 614–644 2, 3, 4, 5, 6, 7, 8, 10, 11, 13, 16, 18, 25, 26, 29, 35

37. Jang, K., Choi, S., Kwon, H., Kim, H., Park, J., Seo, H.: Grover on Korean block ciphers. Applied Sciences **10**(18) (2020) 6

38. Jang, K., Baksi, A., Breier, J., Seo, H., Chattopadhyay, A.: Quantum implementation and analysis of default. Cryptology ePrint Archive, Paper 2022/647 (2022) https://eprint.iacr.org/2022/647. 6

39. Jang, K., Baksi, A., Kim, H., Seo, H., Chattopadhyay, A.: Improved quantum analysis of SPECK and lowmc. In Isobe, T., Sarkar, S., eds.: Progress in Cryptology - INDOCRYPT 2022 - 23rd International Conference on Cryptology in India, Kolkata, India, December 11-14, 2022, Proceedings. Volume 13774 of Lecture Notes in Computer Science., Springer (2022) 517–540 2, 4, 6, 18

40. Jang, K., Song, G., Kim, H., Kwon, H., Kim, H., Seo, H.: Efficient implementation of PRESENT and GIFT on quantum computers. Applied Sciences **11**(11) (2021) 6

41. Jang, K., Song, G., Kim, H., Kwon, H., Kim, H., Seo, H.: Parallel quantum addition for Korean block cipher. IACR Cryptol. ePrint Arch. (2021) 1507 6

42. Jang, K., Song, G., Kwon, H., Uhm, S., Kim, H., Lee, W.K., Seo, H.: Grover on PIPO. Electronics **10**(10) (2021) 1194 6

43. Jaques, S., Naehrig, M., Roetteler, M., Virdia, F.: Implementing grover oracles for quantum key search on aes and lowmc. Cryptology ePrint Archive, Paper 2019/1146 (2019) https://eprint.iacr.org/2019/1146. 3, 5, 18, 20, 22, 30

44. Jaques, S., Naehrig, M., Roetteler, M., Virdia, F.: Implementing grover oracles for quantum key search on AES and LowMC. In Canteaut, A., Ishai, Y., eds.: Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II. Volume 12106 of Lecture Notes in Computer Science., Springer (2020) 280–310 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 20, 21, 22, 23, 24, 25, 26, 30, 31

45. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking symmetric cryptosystems using quantum period finding. In: CRYPTO, Springer (2016) 207–237 1

46. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Quantum differential and linear cryptanalysis. IACR Transactions on Symmetric Cryptology **2016**(1) (Dec. 2016) 71–94 6

47. Kim, P., Han, D., Jeong, K.C.: Time–space complexity of quantum search algorithms in symmetric cryptanalysis: applying to aes and sha-2. Quantum Information Processing **17**(12) (2018) 1–39 8

48. Kranz, T., Leander, G., Stoffelen, K., Wiemer, F.: Shorter linear straight-line programs for mds matrices. IACR Transactions on Symmetric Cryptology **2017**(4) (Dec. 2017) 188–211 14

49. Langenberg, B., Pham, H., Steinwandt, R.: Reducing the cost of implementing the advanced encryption standard as a quantum circuit. IEEE Transactions on Quantum Engineering **1** (01 2020) 1–12 4, 6, 8, 9, 10, 11, 12, 13, 15, 16, 19, 22, 23, 24, 25, 30

50. Li, S., Sun, S., Li, C., Wei, Z., Hu, L.: Constructing low-latency involutory mds matrices with lightweight circuits. IACR Transactions on Symmetric Cryptology (2019) 84–117 2, 3, 10, 13, 14, 15, 17, 20, 35

51. Li, Z., Gao, F., Qin, S., Wen, Q.: New record in the number of qubits for a quantum implementation of aes. Cryptology ePrint Archive, Paper 2023/018 (2023) https://eprint.iacr.org/2023/018. 4, 11, 26, 29

52. Lin, D., Xiang, Z., Xu, R., Zhang, S., Zeng, X.: Optimized quantum implementation of aes. Cryptology ePrint Archive, Paper 2023/146 (2023) https://eprint.iacr.org/2023/146. 2, 4, 10, 11, 16, 26, 29

53. Lin, D., Xiang, Z., Zeng, X., Zhang, S.: A framework to optimize implementations of matrices. In Paterson, K.G., ed.: 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., Springer (2021) 609–632 2, 13, 14

54. Liu, Q., Preneel, B., Zhao, Z., Wang, M.: Improved quantum circuits for aes: Reducing the depth and the number of qubits. Cryptology ePrint Archive, Paper 2023/1417 (2023) https://eprint.iacr.org/2023/1417. 2, 3, 4, 5, 6, 7, 10, 11, 14, 15, 16, 17, 25, 26, 29, 30, 33

55. Liu, Q., Wang, W., Fan, Y., Wu, L., Sun, L., Wang, M.: Towards low-latency implementation of linear layers. IACR Transactions on Symmetric Cryptology **2022**(1) (Mar. 2022) 158–182 2, 13, 14

56. Liu, Q., Zhao, Z., Wang, M.: Improved heuristics for low-latency implementations of linear layers. In: Cryptographers' Track at the RSA Conference, Springer (2023) 524–550 14

57. Luo, Q.b., Yang, G.w., Li, X.y., Li, Q.: Quantum reversible circuits for $GF(2^8)$multiplicative inverse. EPJ Quantum Technology (2022) 11

58. Maximov, A.: AES MixColumn with 92 XOR gates. Cryptology ePrint Archive, Report 2019/833 (2019) https://eprint.iacr.org/2019/833. 3, 4, 9, 13, 14, 20, 21, 22, 23, 25, 30, 31

59. NIST.: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016) https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf. 5, 7, 8, 26, 30

60. NIST.: Call for additional digital signature schemes for the post-quantum cryptography standardization process (2022) https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf. 7, 8, 10, 26, 30

61. Oh, Y., Jang, K., Baksi, A., Seo, H.: Depth-optimized implementation of ascon quantum circuit. Cryptology ePrint Archive, Paper 2023/1030 (2023) https://eprint.iacr.org/2023/1030. 6, 7

62. Perriello, S.: 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) https://hdl.handle.net/10589/147855. 6

63. Roy, S., Baksi, A., Chattopadhyay, A.: Quantum implementation of ascon linear layer. Cryptology ePrint Archive, Paper 2023/617 (2023) https://eprint.iacr.org/2023/617. 6, 13, 14

64. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehle, D.: CRYSTALS-KYBER. Submission to PQC third round (2021) https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf. 2

65. Selinger, P.: Quantum circuits of t-depth one. Physical Review A **87**(4) (2013) 042302 5, 25

66. Song, G., Jang, K., Kim, H., Lee, W., Hu, Z., Seo, H.: Grover on SM3. IACR Cryptol. ePrint Arch. (2021) https://eprint.iacr.org/2021/668. 6

67. Srivastava, V., Gupta, N., Jati, A., Baksi, A., Breier, J., Chattopadhyay, A., Debnath, S.K., Hou, X.: Ascon-sign. NIST PQC Additional Round 1 Candidates (2023) https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/Ascon-sign-spec-web.pdf. 2

68. Tan, Q.Q., Peyrin, T.: Improved heuristics for short linear programs. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1) (2020) 203–230 14

69. Wang, Z., Wei, S., Long, G.: A quantum circuit design of AES (2021) https://arxiv.org/abs/2109.12354. 10, 25

70. Wiebe, N., Roetteler, M.: Quantum arithmetic and numerical analysis using repeat-until-success circuits (2014) https://arxiv.org/abs/1406.2040. 23

71. Xiang, Z., Zeng, X., Lin, D., Bao, Z., Zhang, S.: Optimizing implementations of linear layers. IACR Trans. Symmetric Cryptol. **2020**(2) (2020) 120–145 2, 7, 13, 14, 15

72. Yang, Y., Jang, K., Baksi, A., Seo, H.: Optimized implementation and analysis of cham in quantum computing. Applied Sciences **13**(8) (2023) 6

73. Zalka, C.: Grover's quantum searching algorithm is optimal. Physical Review A **60**(4) (1999) 2746 8

74. Zhu, C., Huang, Z.: Optimizing the depth of quantum implementations of linear layers. In: Information Security and Cryptology - 18th International Conference, Inscrypt 2022, Beijing, China, December 11-13, 2022. Volume 13837 of Lecture Notes in Computer Science., Springer (2022) 129–147 2, 4, 14, 15, 20, 26, 29, 32

75. Zou, J., Li, L., Wei, Z., Luo, Y., Liu, Q., Wu, W.: New quantum circuit implementations of sm4 and sm3. Quantum Information Processing **21**(5) (2022) 1–38 6

76. Zou, J., Liu, Y., Dong, C., Wu, W., Dong, L.: Observations on the quantum circuit of the SBox of AES. Cryptology ePrint Archive, Report 2019/1245 (2019) https://eprint.iacr.org/2019/1245. 13

77. Zou, J., Wei, Z., Sun, S., Liu, X., Wu, W.: Quantum circuit implementations of AES with fewer qubits. In Moriai, S., Wang, H., eds.: Advances in Cryptology – ASIACRYPT 2020, Cham, Springer International Publishing (2020) 697–726 2, 3, 4, 6, 8, 10, 11, 13, 14, 15, 16, 18, 22, 25, 35