Quantum Analysis of AES

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

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

[2] Temasek Laboratories, Nanyang Technological University, Singapore

starj1023@gmail.com, anubhab001@e.ntu.edu.sg, thdrudwn98@gmail.com, khj1594012@gmail.com, hwajeong84@gmail.com, anupam@ntu.edu.sg

Abstract. Quantum computing is considered among the next big leaps in the 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 secret-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) with respect to the quantum implementation and the quantum key search using the Grover's algorithm. We develop a pool of implementations, by mostly reducing the circuit depth metrics. We consider various strategies for optimization, as well as make use of the state-of-the-art advancements in the relevant fields.

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 98 percent for all variants of AES. Our qubit count - Toffoli depth product is improved from theirs by more than 75 percent. Furthermore, we analyze the Jaques et al.'s Eurocrypt'20 implementations in details, fix its bugs and report corrected benchmarks. To the best of our finding, our work improves from all the previous works (including the recent Eprint'22 paper by Huang and Sun) in terms of Toffoli/full depth and Toffoli depth - qubit 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 the public key cryptography will have severe consequence [32], 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 [23, 41]).

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

We thank Da Lin (Hubei University, Wuhan, PR China) for the kind support.

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

[4] For example, the Public Key Encryption & Key Encapsulation Mechanism (PKE & KEM) finalist CRYSTALS-KYBER [54] and the Digital Signature (DS) finalist CRYSTALS-DILITHIUM [26] use SHA-3 in some form.

in [40, 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 quantum vulnerabilities of 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 [31] (refer to Section 2.2 for an overview). As a rule of thumb, it reduces the search space to nearly square root complexity (with a high probability).

Our work makes a humble attempt to conduct a detailed and systematic quantum assisted exhaustive search on the AES family of block ciphers (AES-128, AES-192 and AES-256) [18]. Most recent papers about AES quantum implementations focus on reducing the number of qubits, but do not give much consideration to the depth of the circuit [1, 30, 45, 58, 59, 63]. That said, until a few years ago, quantum computers could not use enough qubits. However, it is hard to say that today's quantum computers are small anymore. Quantum computers that will emerge in the near future are not small, and this can be observed in IBM's quantum computer development roadmap[5]. In the Noisy Intermediate-Scale Quantum (NISQ) era, Toffoli depth is probably the most important metric for error-prone quantum computing [63] and full depth is related to the execution time of circuits [11]. The importance of depth is also observed 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 [52].

We revisit recent research works to incorporate state-of-the art improvements in various related areas (such as the very recent works [46, 47, 48, 60]), in a bid to reduce the cost (qubit count, gate count), circuit depth (Toffoli depth, full depth) and/or cost-depth trade-off (Toffoli depth $\times$ qubit count) of the quantum circuits. 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 described in Section 2.1. 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 [40] contains some programming related issue, which probably results in underestimating the resources for non-linear components (the same issue was reported by the Asiacrypt'20 authors [63], and they did not use those results either); although the linear components are not affected. We patch the issues (such as impossible parallelism and omitting initialization of ancilla qubits) and estimate the correct quantum gates and depth from the number of qubits in Section 5.

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 [63] (such as our Toffoli depth ✦ is reduced by over 98% for AES-128) and also the bug-fixed version of the Eurocrypt'20 paper [40]. We also include the two improved implementations done in [34] for a quick comparison. In [34], 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 depth implementations so-far (with moderate number of qubits and quantum gates). 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 etc.), and finally by sharing of resources among the modules. We present a pool of three implementations, each optimized for a specific objective (see Section 3.1 for related discussion):

- ☆ The *regular* version uses the least qubit count in our work and reduces Toffoli circuit depth compared to the previous works for all the 3 variants. The MixColumn implementation is taken from [60], which supports for zero ancilla/garbage qubits and incurs 92 CNOT gates.
- ◎ The *shallow* version runs all parallel-executable parts of AES simultaneously, including reverse operations. The depth of one round only counts SubBytes + MixColumns, which is ideal. The shallow version takes the least qubit cost and Toffoli circuit depth product with an improved pipeline architecture. According

[5] https://research.ibm.com/blog/ibm-quantum-roadmap.

to [63], this is an important a notion of circuit complexity. Similar to the regular version, the MixColumn implementation is taken from [60].

◈ Further, the *shallow/low depth* version looks for reducing the circuit depth by opting for a low quantum depth implementation of MixColumn (which was found by the authors of [46]).

Table 1: Performance comparison of AES quantum implementations.

AES		Toffoli depth (TD) ✦	Qubit count (M) ❂	Toffoli depth × Qubit ($TD \times M$) ✦ × ❂	Full depth ❋
128	GLRS [30]	12672 (99.76)	984 (−84.55)	12469248 (97.96)	110799 (99.12)
	LPS [45]	1880 (98.41)	864 (−86.43)	1624320 (84.32)	28927 (96.62)
	ZWSLW [63]	2016 (98.51)	512 (−91.96)	1032192 (75.32)	·
	HS [34] † 18	820 (95.12)	492 (−92.27)	403440 (30.86)	·
	HS [34] † 9	1558 (97.43)	374 (−94.13)	582692 (56.29)	·
	❁	2394 (98.33)	1656 (−74.00)	3964464 (93.58)	33320 (97.07)
	✿	40◎◈	6368◎	254720◎	978◎
192	GLRS [30]	11088 (99.68)	1112 (−83.37)	12329856 (97.34)	96956 (98.79)
	LPS [45]	1640 (97.81)	896 (−86.60)	1469440 (78.15)	25556 (95.41)
	ZWSLW [63]	2022 (98.22)	640 (−90.43)	1294080 (75.19)	·
	❁	2682 (98.21)	1976 (−70.46)	5299632 (93.94)	37328 (96.86)
	✿	48◎◈	6688◎	321024◎	1174◎
256	GLRS [30]	14976 (99.72)	1336 (−80.85)	20007936 (98.05)	130929 (98.95)
	LPS [45]	2160 (98.06)	1232 (−82.34)	2661120 (85.32)	33525 (95.89)
	ZWSLW [63]	2292 (98.17)	768 (−88.99)	1760256 (77.81)	·
	❁	3306 (98.31)	2296 (−67.09)	7590576 (94.85)	46012 (97.01)
	✿	56◎◈	6976◎	390656◎	1377◎

Parenthesized numbers show % (positive) improvement reported in this work.
†: Choice of p in [34].

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

❁: Bug-fixed JNRV [40].

Orthogonal to the three architecture, we also use two implements for the S-box, that incur the Toffoli depth of 4 (✿) and 3 (❀) respectively; and were adopted from [34].

On top of that, we present two implementations of the bug-fixed version of Eurocrypt'20 [40] in Section 5 for AES-128 (❁). These two versions differ based on whether the in-place MixColumn from [40] is used (✚) or the Maximov's MixColumn implementation from [50] (✢) is used (both were used in [40]). In order to keep the modification at minimum, we reuse the same design choices made in [40]. For this reason, we reuse the S-box implementation as in [40], which was adopted from [15] (✣).

Combining all, we present 8 distinct implementations for each variant of AES (thus, 24 implementations altogether; the cost for the Grover's search can be observed from Table 8):

☆ Regular version.
- ❀ 3 Toffoli depth S-box [34], MixColumn from [60].
- ✿ 4 Toffoli depth S-box [34]. MixColumn from [60].

◎ Shallow version.
- ❀ 3 Toffoli depth S-box [34], MixColumn [60].
- ✿ 4 Toffoli depth S-box [34], MixColumn [60].

◈ Shallow/low depth version.
- ❀ 3 Toffoli depth S-box [34], MixColumn [46].
- ✿ 4 Toffoli depth S-box [34], MixColumn [46].

❁ Bug-fixing [40].
(a) ✤ S-box from [15], ✚ in-place MixColumn [40].
(b) ✤ S-box from [15], ✢ Maximov's MixColumn [50].

As a consequence of our analysis, the state-of-the-art bounds of the quantum security level (Section 2.3) is updated, as shown in Table 9. We conclude in Section 8, where we present the other AES related quantum analysis with respect to the updated security level in Figure 7.

Some additional information/discussion can be found in Appendices A (a short discussion on the AES variants), B (a brief comparison of classical and quantum depths for the in-place XOR/CNOT implementation of linear layers), and C (per-round based break-up of quantum resources). Our source codes are written in ProjectQ[6], which is a Python-based open-source framework for quantum computing. All our relevant source codes can be accessed as an open-source project[7].

2 Background

2.1 Quantum Gate Basics

Throughout this paper, we use the following shorthand notations: #NOT (reversible NOT gate count ✱), #CNOT (CNOT count ✲), Toffoli count (☆), TD (Toffoli depth ✦), #T (T-gate count ✢), T-depth (✤), #1qCliff as Clifford gate count ❁, #Measure (measurement gate ✯), ❋) and M (qubit count ❂). In particular, 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 (✦ × ❂) as a metric for trade-off. Our AES quantum circuits attain the least Toffoli (✦) and full (❋) depths, significantly contributing to the advancement of the state-of-the-art.

This section describes the quantum gates required to implement the quantum circuit of ciphers. There are representative quantum gates that allow classical Boolean functions to be used in quantum circuits. 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.

Note that the Toffoli gate is decomposed gates in terms of the Cliiford and T gates, the cost and depth of such a decomposition varies based on the method [3,33,55]. Further, a Clifford gate can refer to CNOT and 1qCliff gates. Also, 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), and T-depth 4, and full depth 8 following one of the methods in [3].

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 [31] 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 (some basic familiarity with the quantum notations/terminology is assumed, one may refer to, e.g., [22,51] for a more detailed description):

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:

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

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

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).

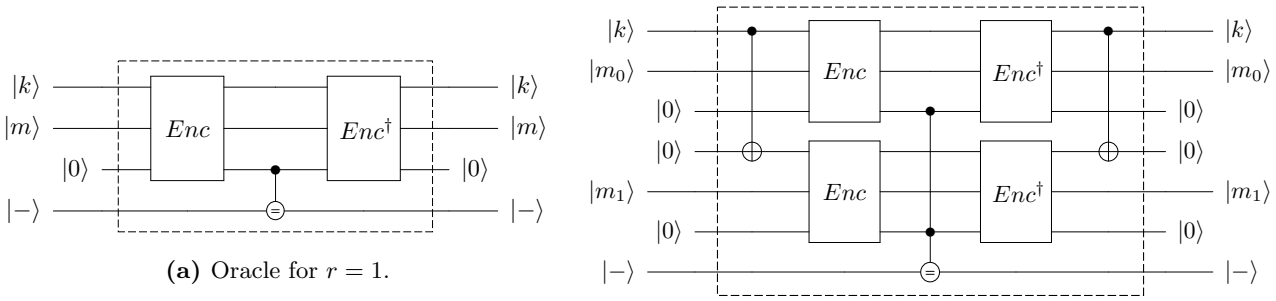
$$f(x) = \begin{cases} 1 \text{ if } Enc_K(p) = c \\ 0 \text{ if } Enc_K(p) \neq c \end{cases} \tag{2}$$

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

3. Lastly, the diffusion operator[8] amplifies the amplitude of the negative sign state. Diffusion operator is implemented with the following (H gates layer → X gates layer → k-qubit controlled Z gate → X gates layer → H gates layer). In [53], 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 → k-qubit controlled Z gate → 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 $\lfloor \frac{\pi}{4}\sqrt{2^k} \rfloor$ [16], 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.



(a) Oracle for $r = 1$.

(b) Oracle for $r = 2$.

Fig. 1: Schematic architecture for key search using Grover's algorithm.

2.3 Related Works

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 [13,40,45,63][9], SIMON [6], SPECK [5,35], PRESENT and GIFT [37], SHA-2 and SHA-3 [2], FSR-based ciphers [4], ChaCha [10], SM3 [56,61], RECTANGLE and KNOT [8], DEFAULT [36], ARIA [17], few Korean ciphers [38,39].

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 [28,42], or specialized quantum attacks like [12,24,25]. These avenues, though important, are out-of-scope for our current work.

[8] Since the diffusion operator is usually generic, it does not require any special techniques to implement.

[9] As far as we can tell, the authors of [13] only made some estimates but did not present any implementation.

Reflection on Huang and Sun (Eprint'22) We are aware of the parallel development by Huang and Sun (Eprint'22) [34]. The content of this paper only revolves with AES-128, and can be summarized as:

- Improve from the Asiacrypt'20 paper's [63] qubit count and performance.
- Choose an improved S-box implementation atop the Eurocrypt'20 implementation [40] with proposal for a quick fix for the qubit count.

In our humble opinion, this patch done by [34] on the Eurocrypt'20 implementation is not perfect (based on the Q# code[10]. Also, the number of qubits was estimated manually in [34, Table 7] in the bug-fix of [40]. 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 [60]), 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. Our approaches are mostly disjoint from that of [34]; and when their S-box implementation is used in our implementation, our result outperforms theirs (thus we have the best-known implementation so far). Parallelism is a major focus in their work, which we pursue through our shallow version. As one can see from Table 1, our results are indeed better than those are reported in [34]. Further, we cover optimized quantum implementations of AES-192 and AES-256 as well.

NIST Security Levels The following security levels were defined by NIST [52] to assess the post-quantum security[11]:

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.

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 [30]), NIST estimated used in [52] 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 [30] 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. A comprehensive synopsis of the notable works can be seen from Table 9, where we show the impact on our work in reshaping the security levels. In particular, the following new bounds are achieved (see also Table 8):

- Level 1: $\mathbf{2^{156.9709}}$ (total Clifford, T gates $= 2^{82.6772}$; full depth $= 2^{74.2940}$)
- Level 3: $\mathbf{2^{222.3219}}$ (total Clifford, T gates $= 2^{115.7706}$; full depth $= 2^{106.5509}$)
- Level 5: $\mathbf{2^{286.8205}}$ (total Clifford, T gates $= 2^{148.0426}$; full depth $= 2^{138.7782}$)

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). However, if it turns out that the depth restriction is not within the stipulated bound, then the following approaches can be undertaken [43]:

1. *Outer parallelization:* Restrict depth at 2^{96} at the expense of lower success probability.
2. *Inner parallelization:* Split into multiple sub-circuits with shallow depth (each recovering part of the secret key) and then combine to get the full key.
3. Cost is calculated as-is without considering MAXDEPTH (see, e.g., [43, Table 2]). It is worth noting that the previous implementations like [1, 34, 45, 63] also did not consider the MAXDEPTH limit.

[10] https://github.com/AES-quantum-circuit/AES-quantum-circuit.

[11] The quantum circuits should have less full depth than the so-called MAXDEPTH limit. The maximum allowable limit of MAXDEPTH is 2^{96}.

Coming back to our analysis, one may note that the depth of quantum attack on AES-128 (i.e., Level 1) is within the permitted MAXDEPTH limit ($2^{74.2940}$). However, the same cannot be stated for -192 and -256, since the full depth figures are respectively $2^{106.5509}$ and $2^{138.7782}$. In this work, we adopt the 3rd approach (i.e., only reporting the cost without considering the MAXDEPTH limit) for the sake of brevity. In future, one may be interested in applying the outer and inner parallelizations to find adjusted cost – success probability trade-offs.

As of now, we remark that the depths (❄ and ✦) of our AES quantum circuits are the lowest when compared to other quantum circuits available in the literature [1, 34, 45, 63]. Table 2 displays a quick view where the related works (namely, GLRS [29] and LPS [45]) are compared with respect to our implementations in terms of full depth (❄). Note that, only AES-128 satisfies the MAXDEPTH criterion (i.e., $\leq 2^{96}$).

Table 2: Summary of AES implementations with respect to MAXDEPTH.

AES ❄	GLRS [29]	LPS [45]	This work				MAXDEPTH ($\leq 2^{96}$)
			☆	◎	◈	❁	
128	$2^{81.2141}$	$2^{79.4751}$	✿: $2^{75.0649}$	✿: $2^{74.5859}$	✿: $2^{74.2940}$	✚: $2^{79.6576}$	✓
			❀: $2^{75.0029}$	❀: $2^{74.5400}$	❀: $2^{74.2388}$	✢: $2^{79.7011}$	
192	$2^{113.4114}$	$2^{111.2987}$	✿: $2^{107.3196}$	✿: $2^{106.8488}$	✿: $2^{106.5509}$	✚: $2^{111.8395}$	✗
			❀: $2^{107.2570}$	❀: $2^{106.8041}$	❀: $2^{106.4957}$	✢: $2^{111.8673}$	
256	$2^{145.6508}$	$2^{143.6871}$	✿: $2^{139.5489}$	✿: $2^{139.0786}$	✿: $2^{138.7782}$	✚: $2^{144.1412}$	✗
			❀: $2^{139.4865}$	❀: $2^{139.0342}$	❀: $2^{138.7225}$	✢: $2^{144.1679}$	

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

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

❁: Bug-fixed JNRV [40] (using S-box from [15] ✣).

✚: Using in-place MixColumn [40].
✢: Using Maximov's MixColumn [50].

3 AES in Quantum

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 qubit-depth. The shallow version also considers the trade-off of qubit-depth, but further reduces the depth by burdening the use of qubit. The shallow version has the best performance in terms of Toffoli depth (✦) and Toffoli depth - qubit count product (✦ × ❂); and taken as the default option in this paper. The shallow/low depth version seems to achieve the lowest depth for quantum circuit implementation.

The regular version of AES (☆) focuses on the parallelism within the round. In this version, while the current round awaits, the previous round goes through the reverse (i.e., uncompute) operation. In other words, the next round cannot start until the reverse operation on the current round is complete.

On the other hand, the shallow version of AES (◎) manages to parallelize the processing for all the rounds. 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 uncompute 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, though the Toffoli depth (✦) is unchanged. In our shallow version, we choose the MixColumn implementation from [60], 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 [46] is chosen instead. Thus, everything else being inherited from the shallow version (◎), the shallow/low depth version (◈) achieves lower full depth.

Most papers implementing quantum circuits for AES focus on reducing the usage of qubits [1, 30, 45, 59, 63]. However, 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 best possible balance between the number of qubits required with its relation to increment of the circuit depth. Thanks to the careful choices, our AES quantum circuits provide arguably the best trade-offs in terms of $TD \times M$ 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 [63].

3.2 Implementation of S-box (SubByte)

Table 3 shows the resources required for the naïve implementations by Boyer-Peralta [14, 15] and the resources for the S-boxes used by the previous authors [45, 63]. 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. Note that the S-box implementation in [30] is based on a field inversion technique, while the rest are based on some version of the Boyar-Peralta's algorithm [14, 15].

Apart from these, another method which is a courtesy of Dansarie [19, 20] exists. This is rather generic, as it can find implementation of an arbitrary 8-bit S-box (i.e., not specific to the AES S-box, which is the case for [14, 15]), with respect to a user-provided set of logic gates. With the publicly available source codes[12] we checked the implementation of the AES S-box. However, the cost for the AES S-box seems to be more than 400 gates, therefore we kept the proper applicability of [19, 20] as a future work.

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

Method	#CNOT ❄	#1qCliff ❁	#T ✢	TD ✦	M ❂	Full depth ❄
S-box [30]	1,818	124	1,792	88	40	951
S-box [14]	358	68	224	8	123	104
S-box [15] ✜	392	72	238	6	136	85
S-box [45]	628	98	367	40	32	514
S-box [63]	437	72	245	55	22	339
S-box [34] ✿	418	72	238	4	136	72
S-box [34] ❀	824	160	546	3	198	69

✜: Reused in this work to fix [40] ❄.
✿: Used in this work (Toffoli depth 4).
❀: Used in this work (Toffoli depth 3).

If the Boyer-Peralta's S-box implementations [14, 15] are directly ported to quantum, then the version of [15] requires more ancilla qubits (120 ancilla qubits) than the quantum version of [14] (107 ancilla qubits), but attains lower depth. JNRV adopted the implementation of the S-box of [15] on a quantum circuit [40] as-is.

Very recently, Huang and Sun reported an improved quantum implementation for the S-box of [40] in their Eprint'22 paper [34]. They presented two quantum implementations of reduced Toffoli depth with new observations of the classical implementation of the AES S-box as given in [15]. 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.

In [45, 63], the authors extended the first S-box implementation by Boyar-Peralta [14] and presented the S-box quantum circuit with a reduced number of qubits. Consequently, it leaves us with a few of ways to choose from.

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. In this case, 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,

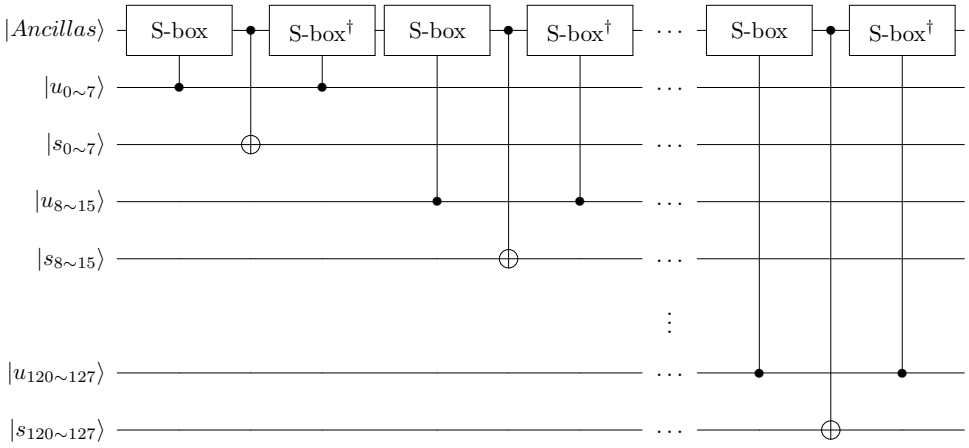
[12] https://github.com/dansarie/sboxgates.

since we believe the benefit of reducing the number of qubits outweighs the cost of performing additional reverse operations. However, only the initial allocation is burdened because the ancilla qubits are reused. Thus, we use Huang and Sun's [34] S-box implementations with relatively high qubit count but low depth. That is, we increase the initial burden and use fast (low depth) S-boxes for free (without ancilla qubits) until the end.

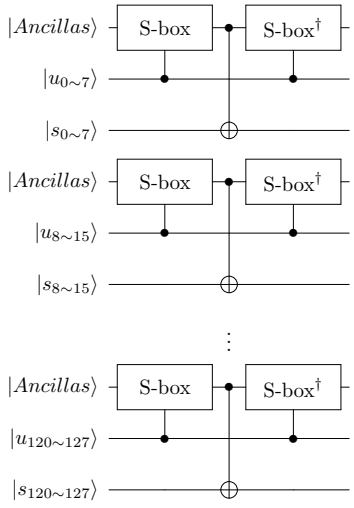
One may note that the AES implementation in [63] required the implementation of the inverse S-box. In our case, however, we do not use the inverse S-box.

3.3 Implementation of SubBytes

After we decide upon the implementation of one 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 ancillas set initially. The notation S-box† is described in Appendix A.



(a) Using 1 set of ancillas.



(b) Using multiple sets of ancillas.

Fig. 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 20×120 ancilla qubits for S-box with Toffoli depth 4 (✿) and 20×182 ancilla qubits for S-box with Toffoli depth 3 (❀) to run all S-boxes simultaneously. Figure 2(b) shows 16 S-boxes

operation in parallel using multiple ancillas 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.

In [63], 16 S-boxes of SubBytes were implemented in parallel using residual ancillas, but key schedule was not implemented in parallel with SubBytes.

3.4 Implementation of Key Schedule

In the key schedule of AES, SubWord operates on rearranged 32-qubit. Out of the $20 \times$ (120 or 182) ancilla qubits previously decided to use (refer to Section 3.3), $4 \times$ (120 or 182) ancilla qubits are used to simultaneously operate S-boxes for 32-qubit in the key schedule (16×120 or 16×182 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 [40]. 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 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 [30], GLRS used Itoh–Tsujii inversion to implement S-box of AES, which requires a lot of quantum resources. Recently, the hardware design for AES has been adopted to implement an efficient S-box quantum circuit. In particular, S-box implementation techniques [14,15] proposed by Boyar-Peralta were frequently used. In [45], Langenberg et al. adopted the S-box implementation of [14] and converted it to suit their purpose of reducing qubits. The S-box implementation of [14] was adopted and improved in [62]. ZWSLW [63] also used the S-box^{-1} 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.

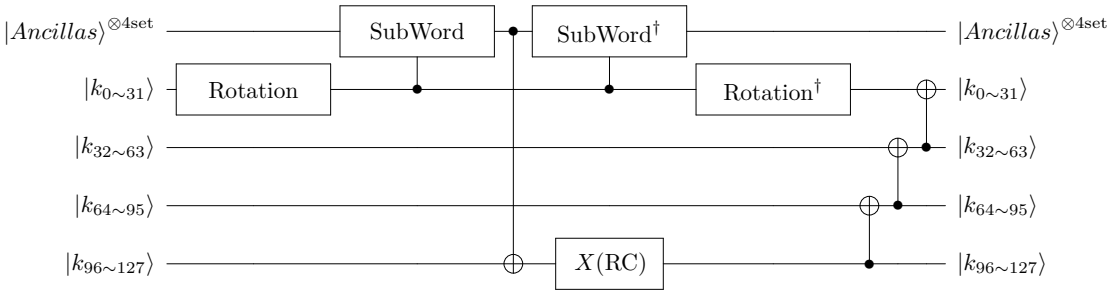


Fig. 3: AES-128 key schedule in quantum.

In most implementations of AES quantum circuits, the full depth and Toffoli depth of AES-128 are higher [30, 40, 45] or similar [63] 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 an overhead for key schedule in implementing AES quantum circuits. However, in our AES quantum circuits there is no overhead for key schedule (except for gates). 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 mostly used in quantum circuit implementations.

3.6 Implementation of MixColumn

In [60], 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 [50]. These two were the least cost implementations in classical circuits, until another implementation with 91 XOR gates (with classical depth 7) was found by [47]. Recently, a new implementation of AES MixColumn was found thanks to [48], which managed to reduce the classical depth to 3 with 103 XOR gates (cf. 103 XOR/3 classical depth implementation from [9]). However, this work came as a tie with another implementation from [46], albeit the latter requires 105 XOR gates.

While dealing with quantum circuit, the best option (in terms of gate cost) seems to be that one reported by [60], as can be seen from Table 4. This implementation is relatively easier to port to a quantum circuit, since the operations are done in-place (i.e., of the form: $a \leftarrow a \oplus b$). In contrast, implementations like that of [47,48,50], are not compatible per-se, as those require usage of temporary variables. These temporary variables would incur additional cost (due to extra ancilla qubits in parallel implementation) and/or depth (due to cleaning up qubits) when these are converted to quantum circuits. On a different direction, the implementation from [46] appears to have lower depth than that of [48] when converted to quantum circuits.

In addition to that, a direct comparison with the quantum MixColumn implementations used by the previous papers [1,29,40,63], this representation is the most efficient in terms of number of qubits (32, a tie with [1,29,40,63]), CNOT gates (92, which is the same as the number of XOR gates in classical, and by far and large the best).

To the best of our knowledge, the result by [60] has never been applied to the implementation of quantum circuits for AES (save for the very recent [34]), and so is the case for [46]. We port the implementation of MixColumn in [60] to quantum and use it in our AES quantum circuit. This implementation is used in the regular (☆) and shallow (◎) versions. Additionally, in order to minimize the circuit depth, we also use the MixColumn implementation from [46] in our shallow/low depth (◈) version.

Apart from the specialized MixColumn implementations just narrated, it is perhaps worth noting that the naïve quantum implementation (i.e., directly porting the matrix to quantum circuit) was seemingly never studied for whatever reason. 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.

The authors of [9] 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; then it incurs 190 CNOT gates (✻) with 127 #qubits (❂) with depth 15. ; however we could not verify the results (probably due to an encoding issue). Second, an implementation of 108 XOR count is mentioned in [27, Footnote 3/Page 42], but it is not clear to us so far. Speaking about MixColumn, it becomes evident from our survey of literature that the quantum optimization is yet to receive a widespread attention[13], for instance the MixColumn has not been optimized for quantum depth yet.

[13]Recent optimizations those rely on multi-input XOR gates [7,49] are not quantum compatible.

Table 4: Comparison of quantum implementations of AES MixColumn.

Method		#CNOT ✻	M ❂	Depth
MixColumn (Naïve)	GF(2^8)	184	64	25
	GF(2)			52
MixColumn [29, 63]†		277	32	39
MixColumn [44]		194	129	15
MixColumn [1]†		275	32	200
MixColumn [50]✛		188	126	13
MixColumn [40]✚†		277	32	111
MixColumn [57]		188	126	17
MixColumn [60]☆◎ †		92	32	30
MixColumn [47]✳		182	123	16
MixColumn [48]✶		206	135	13
MixColumn [9]	103 XOR/3 depth	206	135	11
	95 XOR/6 depth	190	127	15
MixColumn [46]✶◈		210	137	11

✛✚: Reused in this work to fix [40] ❁.

☆◎: Used in regular and shallow versions; in [34].

✳: Least XOR count in classical circuit.

✶: Least depth in classical circuit.

◈: Used in shallow/low depth version.

†: In-place implementation.

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 [46] 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 (Sections 3.2, 3.3) the S-box implementation is also not in-place, requiring ancila qubits (20×120 ✿ or 20×182 ❀). Those ancilla qubits are initialized as 0 after one SubBytes operation (to use in the next round). Therefore, duing the MixColumns operations those qubits are idle. As we only need 64 qubits to implement the MixColumn from [46] (32 as input plus 32 as output qubits), those idle qubits are reused. 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, 60] where it is 32) when the standalone implementation of MixColumns (in which MixColumn does not operate in-place) is considered. However, when the combined SubBytes and MixColumns is considered, because of efficient resource sharing, the total qubit count (❂) does not increase, also the full depth (✻) does not increase.

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, 30, 45], the basic zig-zag architecture (Figure 4(a)) was adopted that uses 4 lines to save qubits by performing reverses on rounds. In [63], 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 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.

In our approach, which has already allocated many ancilla qubits, the overhead of increasing the number of qubits according to the architecture 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. 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 (◎, ◈).

4.1 Regular Version

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†.

As show 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 72 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 1364 (about 72×19) for AES-128, 1627 (about 72×23) for AES-256, and 1907 (about 72×27) for AES-256.

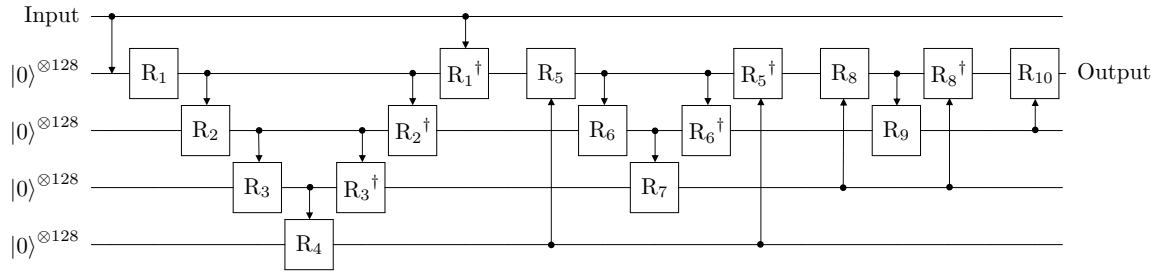
4.2 Shallow Version and Shallow/Low Depth Version

Further, 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 ancilla qubit. The shallow version counts the depth for one round as SubBytes (72) + MixColumns (30), which is the ideal depth. The circuit depth of AES-128 is 978 (about 9 rounds × 102 + 72), AES-192 is 1174 (about 11 rounds × 102 + 72), and AES-256 is 1377 (about 13 rounds × 102 + 72). In the shallow version, up to SubBytes† operates concurrently within one round, providing maximum parallelism. Finally, the shallow and shallow/low depth versions (◎, ◈) offer the least Toffoli depth (✦) of the S-box's Toffoli depth × rounds and Toffoli depth × qubit count (✦ × ❂).

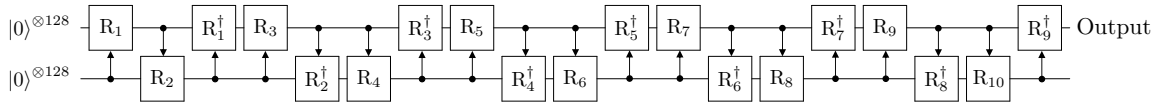
The low depth version changes only the MixColumn from the shallow version to a MixColumn based on [46]. The low depth version counts the depth for one round as SubBytes (72) + MixColumns (11). The low depth version of AES (◈) offers the least Toffoli depth (✦) and full depth (❄).

5 Bug-fixing JNRV (Eurocrypt'20) AES Implementations

In this part, we report errors from the AES implementation and resource estimation by Jaques, Naehrig, Roetteler and Virdia in Eurocrypt'20 [40]. To this end, we analyze the Q# code of their AES implementation and cross-compare it with the quantum resources reported in the Eurocrypt'20 paper. Furthermore, we fix bugs in the AES implementation of Eurocrypt'20 and report the corrected resources for the lower-bound resources in their work.



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



(b) Modified (ZWSLW).

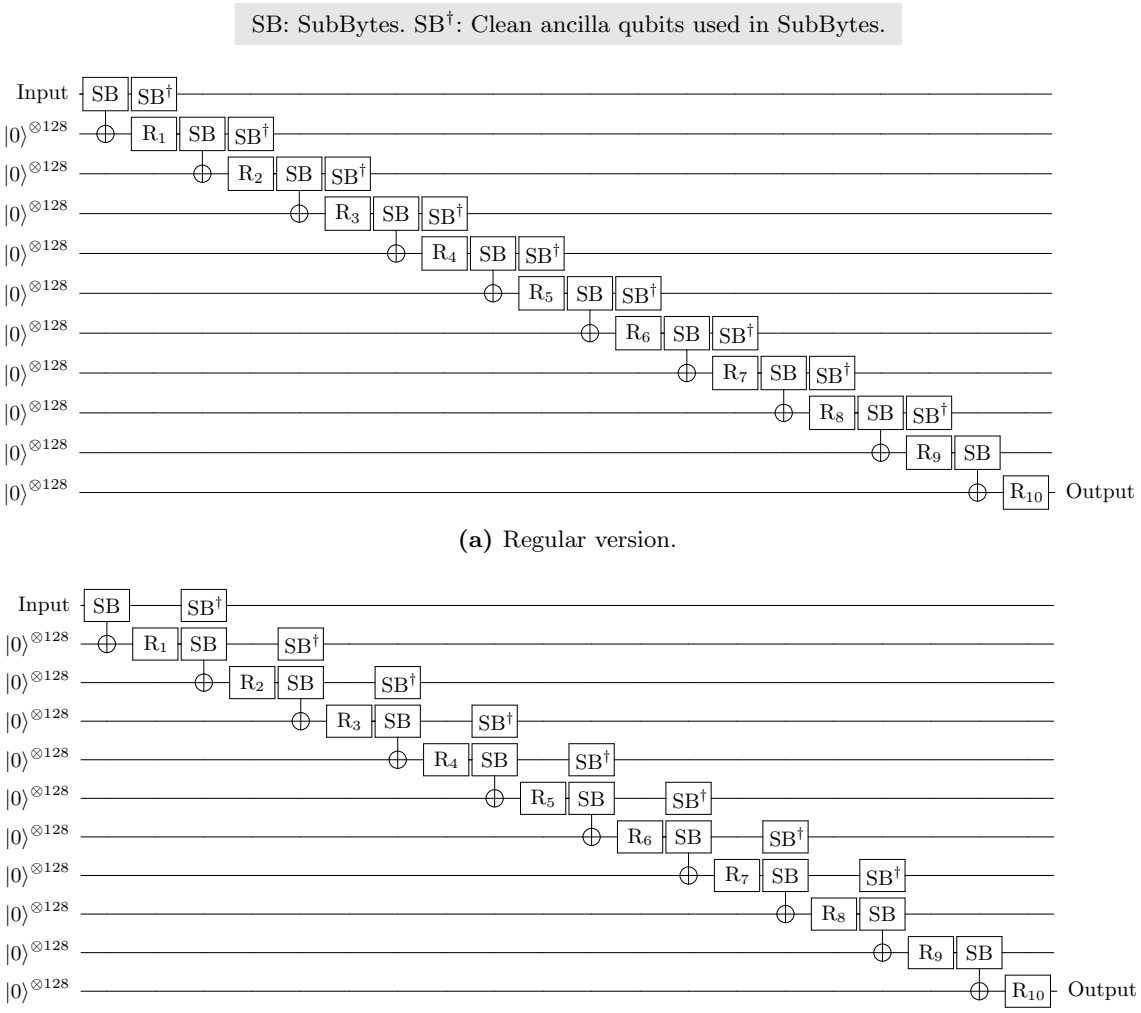
Fig. 4: Zig-zag architecture for AES-128 quantum circuit.

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

AES		#CNOT ✻	#NOT ✱	#Toffoli ☆	TD ✦	#qubits (M) ❂	$TD \times M$ ✦ × ❂	Full depth ❉
128	GLRS [30]	166,548	1,456	151,552	12,672	984	12,469,248	110,799
	ASAM [1]	192,832	1,370	150,528	·	976	·	·
	LPS [45]	107,960	1,570	16,940	1,880	864	1,624,320	28,927
	ZWSLW [30]	128,517	4,528	19,788	2,016	512	1,032,192	·
	☆	84,120	800	12,920	76	3,936	299,136	1,364
	◎ ✿	81,312	800	12,240	40	6,368	254,720	978
	◈	90,816	800	12,240	40	7,520	300,800	799
	☆	138,080	800	29,640	57	5,176	295,032	1,307
	◎ ❀	132,432	800	28,080	30	8,848	265,440	948
	◈	141,936	800	28,080	30	10,000	300,000	769
192	GLRS [30]	189,432	1,608	172,032	11,088	1,112	12,329,856	96,956
	LPS [45]	125,580	1,692	19,580	1,640	896	1,469,440	25,556
	ZWSLW [30]	152,378	5,128	22,380	2,022	640	1,294,080	·
	☆	96,112	896	14,688	92	4,256	391,552	1,627
	◎ ✿	92,856	896	14,008	48	6,688	321,024	1,174
	◈	104,472	896	14,008	48	8,096	388,608	955
	☆	157,456	896	33,696	69	5,496	379,224	1,558
	◎ ❀	151,360	896	32,136	36	9,168	330,048	1,138
	◈	162,976	896	32,136	36	10,576	380,736	919
256	GLRS [30]	233,836	1,943	215,040	14,976	1,336	20,007,936	130,929
	LPS [45]	151,011	1,992	23,760	2,160	1,232	2,661,120	33,525
	ZWSLW [30]	177,645	6,103	26,774	2,292	768	1,760,256	·
	☆	117,704	1,103	18,088	108	4,576	494,208	1,907
	◎ ✿	113,744	1,103	17,408	56	6,976	390,656	1,377
	◈	127,472	1,103	17,408	56	8,640	483,840	1,118
	☆	193,248	1,103	41,496	81	5,816	471,096	1,826
	◎ ❀	186,448	1,103	39,936	42	9,456	397,152	1,335
	◈	200,176	1,103	39,936	42	11,120	467,040	1,076

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

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



(a) Regular version.

(b) Shallow and shallow/low depth versions.

Fig. 5: Pipeline architecture of AES-128.

5.1 Issues with Q#

For a clearer context, we give a brief description of the cases where Q#'s `ResourcesEstimator` issues arise and how those issues affect the quantum benchmarks given in the Eurocrypt'20 paper [40]. This was discovered when we tried to cross-check their publicly available source codes[14]. Indeed, this was also noted in [63] as a bug; and this apparently led to underestimation of gate count, qubit count and depth reported in [40] for the non-linear components (namely the S-box and S-box^{-1} of AES).

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.

Non-parallelizable SubBytes In their implementation, the S-box of [14] is adopted and ported to the quantum domain. The quantum resources required for the S-box quantum circuit reported in the Eurocrypt'20 paper [40, 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

[14] https://github.com/microsoft/grover-blocks.

SubBytes quantum circuit is the serial structure of Figure 2(a). 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 Sbox 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, one garbage qubit (bottom line in Figure 6) is used. Thus, if replaced with AND gates, 5 garbage qubits for 5 AND gates must be allocated for parallel operation. Clearly, the garbage 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.

We could not understand how the AND† gate was supposed to work in [40]. As after the measurement gate the we have a classical bit, [40, Figure 7(b)], which is shown in the classical bit. However, there seems to be a feedback (based on if the classical bit is measured 1 or 0). This should not respect the properties of a quantum circuit[15], to the best of our knowledge/understanding.

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.

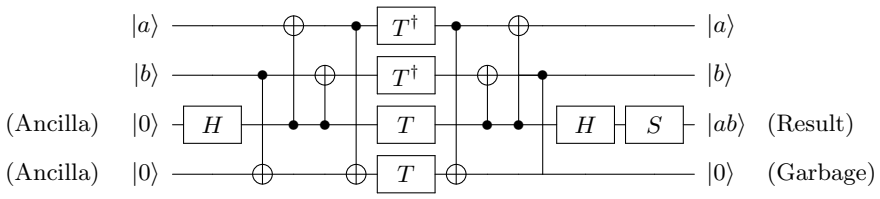


Fig. 6: Quantum AND gate in JNRV (Eurocrypt'20).

Uninitialized Ancilla Qubits in SubBytes To generate the quantum S-box output for [15], an ancilla set consisting of 120 qubits is required. Ancilla qubits are responsible for storing the temp values to compute the correct output for the input. After S-box is operated, these ancilla qubits still have their temp values stored. Thus, in order to reuse these ancilla qubits in the subsequent S-boxes, it is essential to initialize the stored temp values to 0. For this, reverse operation must be performed as shown in Figure 2(a) or 2(b).

However, in their quantum circuit implementation, initialization of qubits of ancilla set used in S-box is omitted. The authors say that they do not clean up ancilla qubits by not performing a reverse operation until the ciphertext is generated. This implementation is possible only when new clean ancilla qubits are allocated for every S-box used in encryption. Their intention is to reduce circuit depth, but we believe that omitting the reverse operation will not produce the correct ciphertext. Since the unclean ancilla qubits are reused in subsequent S-boxes, the correct output cannot be generated, so their implementation cannot be verified. Also, reduced quantum gates and depth due to the omitted reverse operations are estimated.

When we tested the SubWord quantum circuit for a 32-qubit input `ffffffff` by omitting the reverse operation and reusing the ancilla set, the output was `6a4e6216`. Only the first S-box generates the correct output (16), and the subsequent S-boxes generate incorrect outputs. By analyzing their Q# source code, we think that this problem occurs probably because ancilla qubits were allocated by C#'s `using` command.

5.2 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.

[15]For perspective, see Figure 2 of https://www.ibm.com/blogs/research/2021/02/quantum-mid-circuit-measurement/, where the classical bit is not reused in the quantum circuit.

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

(a) AES-128 gate costs

Method	S-box (SubByte)	MixColumns	
		In-place [40] ✚	Maximov [50] ✛
#CNOT ✲	654	1,108	1,248
#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 [40] ✚			Maximov's MixColumn [50] ✛		
	AES-128	AES-192	AES-256	AES-128	AES-192	AES-256
#CNOT ✲	292,313	329,697	404,139	294,863	332,665	407,667
#1qCliff ❁	84,428	94316	116,286	84,488	94,092	116,062
#T ✢	54,908	61,436	75,580	54,908	61,436	75,580
#Measure ✯	13,727	15,359	18,895	13,727	15,359	18,895
T-depth ✤	121	120	126	121	120	126
#qubits (M) ❂	1,665	1,985	2,305	2,817	3,393	3,969
Full depth ❋	2,816	2,978	3,353	2,086	1,879	1,951

(c) AES-128 Modules

Method	#CNOT ✲	#1qCliff ❁	#T ✢	T-depth ✤	M ❂	Full depth ❋
SubBytes	12,000	1,220	7,328	768	376	2,672
Key schedule	3,096	355	1,832	192	248	669
MixColumns (Maximov [50]) ✛	1,248	0	0	0	318	88
One round†	16,472	1,507	9,160	960	632	3,417

†: One typical round (that includes MixColumn).

(d) Summary

AES	#CNOT ✲	#1qCliff ❁	#T ✢	T-depth ✤	M ❂	Full depth ❋
128	161,982	14,400	91,380	9,576	1,656	33,320
192 ✚	182,774	16,128	102,372	10,728	1,976	37,328
256	224,214	19,871	126,188	13,224	2,296	46,012
128	163,242	14,994	91,380	9,576	2,808	33,914
192 ✛	184,314	16,854	102,372	10,728	3,384	38,054
256	226,034	20,729	126,188	13,224	3,960	46,870

✚: Using in-place MixColumn [40].
✛: Using Maximov's MixColumn [50].

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 that ancilla qubits allocated by `using` command are always prepared in a clean state. After S-box operation, ancilla qubits are not in a clean state (i.e., not all zero), so these cannot be used in the next S-box as-is. However, the qubits allocated by the `using` command are always set to 0, the impossible S-box operation becomes possible. This is possible if new ancilla qubits are allocated for every S-box, but the qubits do not increase in resource estimation.

These issues allow designing quantum circuit structures that are impossible. In these issues, we estimate the corrected results from the results reported as lower-bound in their AES quantum circuit architecture. We contribute to three major modifications:

1. We reflect on the increasing depth in their number of qubits using only one ancilla set. 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.
2. We modify the omitted reverse operations to be performed so that they produce the correct output, which further increases the depth.
3. We correct the implementation of MixColumns where the same issue occurs. In Eurocrypt'20 paper [40], two MixColumn implementations were presented. The in-place method of MixColumn implementation (which uses PLU decomposition, and derived by the authors themselves [40]) does not cause this issue. On the other hand, similar to S-box, the same issue applies to the MixColumn implementation by Maximov [50], which requires ancilla qubits, so this is also solved in the same way as the S-box.

We have modified from the quantum circuit base of [40] and implemented it on ProjectQ. Our source code generates the correct ciphertext and the corrected cost of the resources are estimated. To avoid confusion, we estimate quantum resources using Toffoli gates (using the method from [3]), rather than applying AND gates (which could lead to some coding-related issues).

Results with bug-fixed Eurocrypt'20 implementation can be found in Table 6. 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 the 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 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). Table 6(c) shows the estimated resources for SubBytes, key schedule, MixColumns, and one round where the issue occurs. The difference for the corrected MixColumns is relatively small, but the depth estimated as lower-bound for SubBytes is corrected high. The resources estimated in Table 6(c) include a reverse operation to clean ancilla qubits. Table 6(d) shows the corrected quantum resources for AES quantum circuits, and it is confirmed that the depth increases significantly when maintaining the number of qubits.

6 Performance of AES 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`.

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. In [1,30], Itoh–Tsujii-based inversion is implemented on a quantum circuit, so many resources are used for SubBytes. In [45, 63], more efficient quantum circuits are implemented by extending the S-box of [14], 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 [63], $TD \times M$ (where TD is the Toffoli depth ✦, and M is the number of qubits ✪ is used to measure the trade-off of quantum circuits. 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 [40], the quantum resources required to implement quantum circuits for AES were also estimated. However, there seem to be some issues with Q#'s `ResourcesEstimator`[16] used in their work, especially in implementing quantum circuits for SubBytes. Therefore, the results of [40] are not used here.

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

Following [63, Table 10], Table 7 shows detailed quantum resources by decomposing Toffoli gates 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].

Table 7: Quantum circuit resources required for variants of AES (this work).

AES		#CNOT	#1qCliff	#T	T-depth	M	Full depth
128	☆	161,640	14,400	90,440	304	3,936	1,364
	◎ ✿	154,752	14,400	85,680	160	6,368	978
	◈	164,256	16,832	85,680	160	7,520	799
	☆	315,920	32,000	207,480	228	5,176	1,307
	◎ ❀	300,912	32,000	196,560	120	8,848	948
	◈	310,416	33,248	196,560	120	10,000	769
192	☆	184,240	16,400	102,816	368	4,256	1,627
	◎ ✿	176,904	16,400	98,056	192	6,688	1,174
	◈	188,520	19,440	98,056	192	8,096	955
	☆	359,632	36,464	235,872	276	5,496	1,558
	◎ ❀	344,176	36,464	224,952	144	9,168	1,138
	◈	355,792	38,024	224,952	144	10,576	919
256	☆	226,232	19,871	126,616	432	4,576	1,907
	◎ ✿	218,192	19,871	121,856	224	6,976	1,377
	◈	231,920	23,519	121,856	224	8,640	1,118
	☆	442,224	44,159	290,472	324	5,816	1,826
	◎ ❀	426,064	44,159	279,552	168	9,456	1,335
	◈	439,792	46,031	279,552	168	11,120	1,076

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

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

7 Performance of Quantum Key Search on AES

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 [5, 30, 45]. 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. This occupies a small part in the oracle, and since the main part is a block cipher's quantum circuit, the cost for an n-multi controlled NOT gate is omitted for simplicity of analysis.

In quantum exhaustive key search, to recover a unique key, not a spurious key, Grassl et al. in [30] estimated the attack cost for r known (plaintext, ciphertext) pairs ($r = 3$, $r = 4$ and $r = 5$, respectively). Later in [45], 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 [40] also estimated the cost for the same r (plaintext, ciphertext) pairs in [45] 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 cost of quantum exhaustive key search for the

Table 8: Quantum resources required for Grover's search for AES (this work).

AES		r	#qubits (M) ❂	Total gates (Decomposed) ✻	Full depth ❊	Cost (Complexity) ✻ × ❊
128	☆	1	3,937	$1.597 \cdot 2^{82}$	$1.046 \cdot 2^{75}$	$1.671 \cdot 2^{157}$
	◎ ✿		6,369	$1.527 \cdot 2^{82}$	$1.501 \cdot 2^{74}$	$1.146 \cdot 2^{157}$
	◈		7,521	$1.599 \cdot 2^{82}$	$1.226 \cdot 2^{74}$	$1.960 \cdot 2^{156}$
	☆		5,177	$1.664 \cdot 2^{83}$	$1.002 \cdot 2^{75}$	$1.668 \cdot 2^{158}$
	◎ ❀		8,849	$1.586 \cdot 2^{83}$	$1.454 \cdot 2^{74}$	$1.153 \cdot 2^{158}$
	◈		10,001	$1.619 \cdot 2^{83}$	$1.18 \cdot 2^{74}$	$1.909 \cdot 2^{157}$
192	☆	2	7,841	$1.683 \cdot 2^{115}$	$1.248 \cdot 2^{107}$	$1.05 \cdot 2^{223}$
	◎ ✿		12,225	$1.619 \cdot 2^{115}$	$1.801 \cdot 2^{106}$	$1.457 \cdot 2^{222}$
	◈		15,041	$1.706 \cdot 2^{115}$	$1.465 \cdot 2^{106}$	$1.25 \cdot 2^{222}$
	☆		10,073	$1.753 \cdot 2^{116}$	$1.195 \cdot 2^{107}$	$1.048 \cdot 2^{224}$
	◎ ❀		16,689	$1.682 \cdot 2^{116}$	$1.746 \cdot 2^{106}$	$1.469 \cdot 2^{223}$
	◈		19,505	$1.722 \cdot 2^{116}$	$1.41 \cdot 2^{106}$	$1.214 \cdot 2^{223}$
256	☆	2	8,417	$1.012 \cdot 2^{148}$	$1.463 \cdot 2^{139}$	$1.481 \cdot 2^{287}$
	◎ ✿		12,737	$1.955 \cdot 2^{147}$	$1.056 \cdot 2^{139}$	$1.032 \cdot 2^{287}$
	◈		16,065	$1.03 \cdot 2^{148}$	$1.715 \cdot 2^{138}$	$1.766 \cdot 2^{286}$
	☆		10,649	$1.055 \cdot 2^{149}$	$1.401 \cdot 2^{139}$	$1.477 \cdot 2^{288}$
	◎ ❀		17,201	$1.018 \cdot 2^{149}$	$1.024 \cdot 2^{139}$	$1.042 \cdot 2^{288}$
	◈		20,529	$1.041 \cdot 2^{149}$	$1.65 \cdot 2^{138}$	$1.719 \cdot 2^{287}$

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

target block cipher is roughly the cost of oracle $\times \lfloor \frac{\pi}{4}\sqrt{2^k} \rfloor$ (where k is the key size). Costs are estimated at the (Clifford + T) level and computed as the number of total decomposed gates × full depth (✻ × ❊).

We show the cost of quantum key search by the Grover's algorithm for AES-128, AES-192, AES-256; using S-boxes with Toffoli depth of 4 and 3 in Table 8. Additionally, a quick comparison of NIST's security level (under the Grover's search) of our work together with the previous works is given in Table 9. One may note that the number of qubits is not included in NIST's estimation, probably because NIST is focusing more on gates and depths that increase drastically with the number of serial steps needed in the Grover's search. Anyway, when compared with the current state-of-the-art security bounds (Section 2.3), we reduce the quantum cost for running the Grover's search on the AES family, thereby setting up a new benchmark for the NIST security levels (Table 9). The cost is calculated in terms of the product of decomposed (Clifford and T) gate count and full depth.

8 Conclusion

In this work, we collate multiple contributions reported in the last couple of years, together with up-to-date improvements in the quantum technology as well as optimizations on the building blocks of the ciphers. 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 [63] and [34] respectively). At the same time, we improve the qubit count - Toffoli depth product (✦ × ❂) respectively by more than 75% and 30% from the same papers. We also patch the implementations from [40]. Figure 7 shows the contribution of our work compared to GLRS [29] and LPS [45] in a nutshell.

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 [19, 20, 21] can possibly be considered as starting points.

Table 9: Comparison of NIST security levels based on variants of AES.

Level (AES)	GLRS [30]	NIST [52]	LPS [45]	This work			
				☆	◎	◈	❁
1 (128)	$2^{168.6683}$	2^{170}	$2^{162.6093}$	✿: $2^{157.7407}$	✿: $2^{157.1966}$	✿: $2^{156.9709}$	✚: $2^{162.3577}$
				❀: $2^{158.7381}$	❀: $2^{158.2054}$	❀: $2^{157.9328}$	✛: $2^{162.5641}$
3 (192)	$2^{233.4645}$		$2^{227.6491}$	✿: $2^{223.0704}$	✿: $2^{222.5430}$	✿: $2^{222.3219}$	✚: $2^{227.5867}$
				❀: $2^{224.0676}$	❀: $2^{223.5548}$	❀: $2^{223.2798}$	✛: $2^{227.6260}$
5 (256)	$2^{298.3467}$		$2^{292.3100}$	✿: $2^{287.5666}$	✿: $2^{287.0454}$	✿: $2^{286.8205}$	✚: $2^{292.1520}$
				❀: $2^{288.5627}$	❀: $2^{288.0594}$	❀: $2^{287.7816}$	✛: $2^{292.1900}$

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

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

❁: Bug-fixed JNRV [40] (using S-box from [15] ✤).

✚: Using in-place MixColumn [40]. ✛: Using Maximov's MixColumn [50].

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 2, 6, 7, 8, 11, 12, 14, 18
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 5
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 4, 8, 16, 18, 19
4. 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.: 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) 179–198 5
5. 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 5, 19
6. 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. 5
7. 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 11
8. 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) 5
9. 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 11, 12
10. Bathe, B.N., Anand, R., Dutta, S.: Evaluation of Grover's algorithm toward quantum cryptanalysis on ChaCha. Quantum Inf. Process. **20**(12) (2021) 394 5
11. Bhattacharjee, D., Chattopadhyay, A.: Depth-optimal quantum circuit placement for arbitrary topologies. arXiv preprint arXiv:1703.08540 (2017) 2
12. 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 5
13. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: Quantum security analysis of AES. IACR Transactions on Symmetric Cryptology **2019**(2) (Jun. 2019) 55–93 5
14. 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 8, 10, 15, 18
15. 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, 4, 7, 8, 10, 16, 21

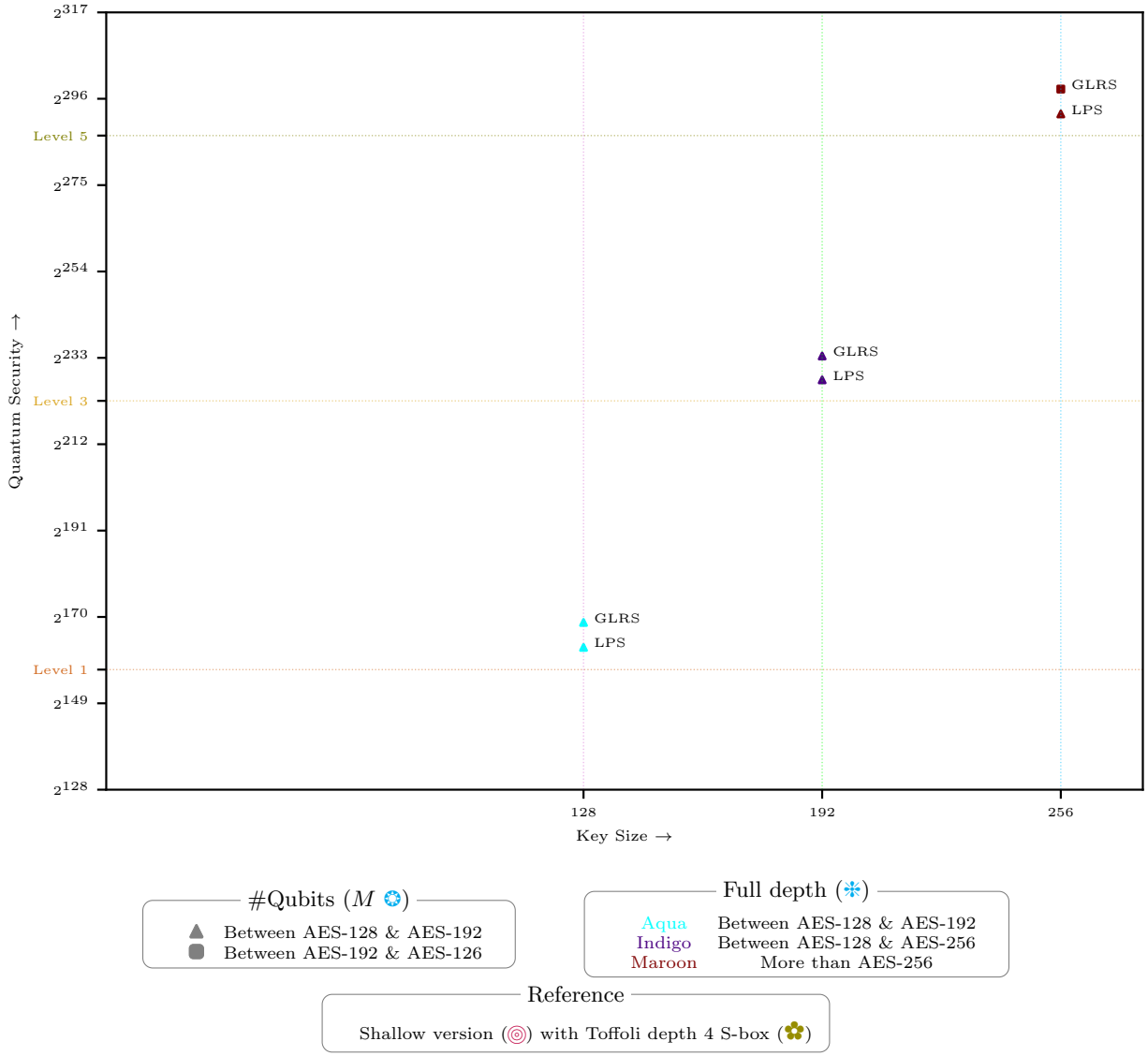


Fig. 7: View of other analysis of AES with respect to this work.

16. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik **46**(4-5) (Jun 1998) 493–505 5
17. 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 5
18. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer (2002) 2, 24
19. 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. 8, 20
20. Dansarie, M.: sboxgates: A program for finding low gate count implementations of S-boxes. Journal of Open Source Software **6**(62) (2021) 2946 8, 20
21. 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 20
22. de Wolf, R.: Quantum Computing: Lecture Notes. (2019) https://arxiv.org/pdf/1907.09415v1.pdf. 4
23. Dong, X., Dong, B., Wang, X.: Quantum attacks on some feistel block ciphers. Des. Codes Cryptogr. **88**(6) (2020) 1179–1203 1
24. 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.: Advances in Cryptology - ASIACRYPT 2020 -

26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II. Volume 12492 of Lecture Notes in Computer Science., Springer (2020) 727–757 5
25. 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.: 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) 241–271 5
26. 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. 1
27. 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 11
28. Frixons, P., Naya-Plasencia, M., Schrottenloher, A.: Quantum boomerang attacks and some applications. IACR Cryptol. ePrint Arch. (2022) 60 5
29. Grassi, L., Rechberger, C., Rønjom, S.: Subspace trail cryptanalysis and its applications to AES. Volume 2016. (2016) 192–225 7, 11, 12, 20
30. 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 2, 3, 6, 8, 10, 12, 14, 18, 19, 21
31. 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, 4
32. 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
33. 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 4
34. Huang, Z., Sun, S.: Synthesizing quantum circuits of aes with lower t-depth and less qubits. Cryptology ePrint Archive, Report 2022/620 (2022) https://eprint.iacr.org/2022/620. 2, 3, 6, 7, 8, 9, 11, 12, 20
35. Jang, K., Choi, S., Kwon, H., Kim, H., Park, J., Seo, H.: Grover on Korean block ciphers. Applied Sciences **10**(18) (2020) 5
36. 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. 5
37. 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) 5
38. 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 5
39. Jang, K., Song, G., Kwon, H., Uhm, S., Kim, H., Lee, W.K., Seo, H.: Grover on pipo. Electronics **10**(10) (2021) 1194 5
40. 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, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21
41. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking symmetric cryptosystems using quantum period finding. In: CRYPTO, Springer (2016) 207–237 1
42. 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 5
43. 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 6
44. 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 12
45. 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 2, 3, 5, 6, 7, 8, 10, 12, 14, 18, 19, 20, 21
46. 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, 7, 11, 12, 13
47. 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, 11, 12
48. 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, 11, 12

49. Liu, Q., Wang, W., Sun, L., Fan, Y., Wu, L., Wang, M.: More inputs makes difference: Implementations of linear layers using gates with more than two inputs. IACR Transactions on Symmetric Cryptology **2022**(2) (Jun. 2022) 351–378 11
50. Maximov, A.: AES MixColumn with 92 XOR gates. Cryptology ePrint Archive, Report 2019/833 (2019) https://eprint.iacr.org/2019/833. 3, 4, 7, 11, 12, 17, 18, 21
51. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information (10th Anniversary Edition). Cambridge University Press (2010) 4
52. 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. 2, 6, 21
53. Perriello, S.: Design and development of a quantum circuit to solve the information set decoding problem. (2019) 5
54. 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. 1
55. Selinger, P.: Quantum circuits of t-depth one. Physical Review A **87**(4) (2013) 042302 4
56. 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. 5
57. Tan, Q.Q., Peyrin, T.: Improved heuristics for short linear programs. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1) (2020) 203–230 12
58. Wang, Z.G., Wei, S.J., Long, G.L.: A quantum circuit design of aes requiring fewer quantum qubits and gate operations. Frontiers of Physics **17**(4) (2022) 1–7 2
59. Wang, Z., Wei, S., Long, G.: A quantum circuit design of AES (2021) https://arxiv.org/abs/2109.12354. 2, 8
60. 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, 3, 6, 7, 11, 12, 26
61. 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 5
62. 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. 10
63. 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, 5, 6, 7, 8, 9, 10, 11, 12, 15, 18, 19, 20, 27

A Concise Description of AES Variants

The Advanced Encryption Standard (AES) [18] 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.

Round Function The round function of AES consists of AddRoundKey $\circ$ MixColumns $\circ$ ShiftRows $\circ$ 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 r^{th} row of state to the left by i places, for $i = 0, 1, 2, 3$.

MixColumns. The MixColumn operation pre-multiplies each of the state column with the right circulant matrix $(\mathtt{02}, \mathtt{03}, \mathtt{01}, \mathtt{01})$, over $\mathrm{GF}(2^8)[x]$ with modulus $x^4 + 1$. 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.

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).

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

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

For AES-128, the word W_i is generated by

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

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

For AES-192, the word W_i is generated by

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

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

Notes

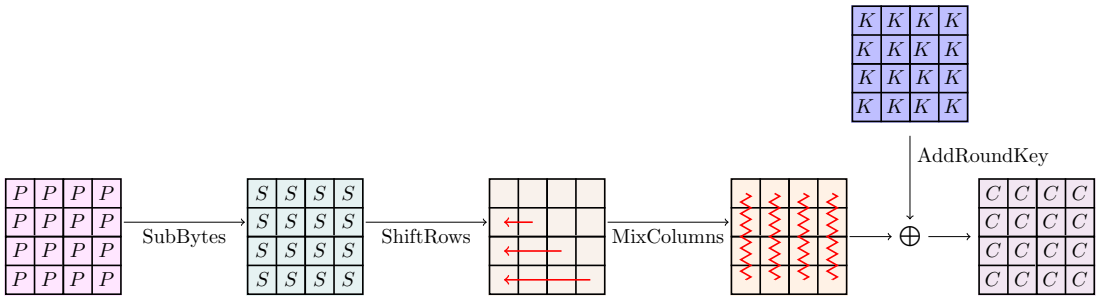
Singular and Plural Forms The AES state is represented as a 4×4 matrix and the operation on one column of the matrix is denoted here as MixColumn. As described earlier, MixColumn corresponds to a matrix multiplication over GF(2^8), which can equivalently be expressed as multiplication by a matrix of 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.

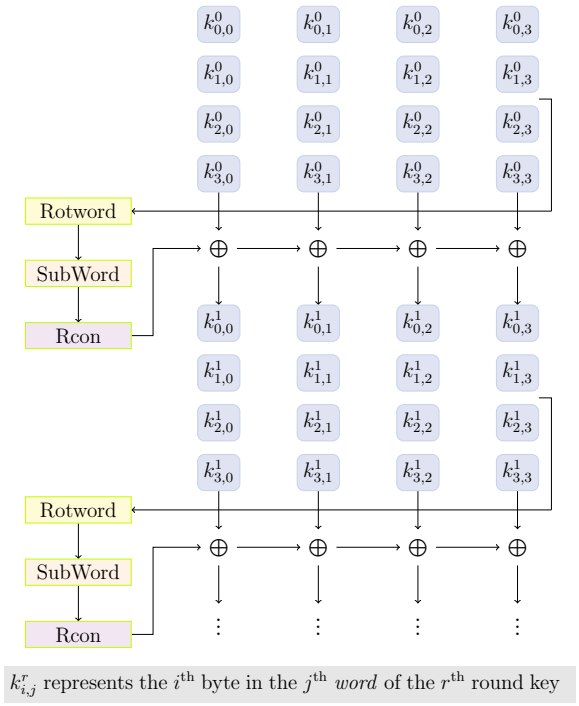
Different from MixColumns and ShiftRows, the SubBytes in the round function updates every byte in the 4×4 matrix in the same way. The process of applying the S-box to one byte in the AES state is denoted here as SubByte. In each round, the SubBytes updates all the bytes in the 4×4 matrix by replacing 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.

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.



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



(b) Key schedule for encryption.

Fig. 8: Schematic of AES construction.

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 Depth of Sequential XOR: Classical vs. Quantum

One may note from Table 4 that the depth for quantum circuit corresponding to the implementation by [60] 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.

C Further Result

Similar to [63, Table 6], we show the per-round benchmark for our implementations of the AES family in Table 10 (using the S-box implementation with Toffoli depth 3 and 4 in Table 10(a) and 10(b), respectively).

Table 10: 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	1≀	8,960	5,064	6,120	79	1,360	680	8	4
	2	8,832	8,960	10,016	79	1,360	1,360	8	4
	3	8,832	8,960	10,016	81	1,360	1,360	8	4
	4	8,832	8,960	10,016	81	1,360	1,360	8	4
	5	8,832	8,960	10,016	81	1,360	1,360	8	4
	6	8,832	8,960	10,016	79	1,360	1,360	8	4
	7	8,832	8,960	10,016	79	1,360	1,360	8	4
	8	8,832	8,960	10,016	81	1,360	1,360	8	4
	9	8,832	8,960	10,016	80	1,360	1,360	8	4
	10	4,504	4,568	4,568	80	680	680	4	4
192	1≀	9,024	9,056	10,112	79	1,360	1,360	8	4
	2	8,896	8,992	10,048	79	1,360	1,360	8	4
	3	7,088	7,152	8,208	64	1,088	1,088	8	4
	4	8,896	8,928	9,984	81	1,360	1,360	8	4
	5	8,896	8,992	10,048	81	1,360	1,360	8	4
	6	7,088	7,152	8,208	64	1,088	1,088	8	4
	7	8,896	8,928	9,984	81	1,360	1,360	8	4
	8	8,896	8,992	10,048	79	1,360	1,360	8	4
	9	7,088	7,152	8,208	64	1,088	1,088	8	4
	10	8,896	8,928	9,984	79	1,360	1,360	8	4
	11	8,896	5,032	6,088	81	1,360	680	8	4
	12	3,552	3,552	3,552	64	544	544	4	4
256	1≀	7,216	4,048	5,104	64	1,088	544	8	4
	2	8,832	8,040	9,096	79	1,360	1,224	8	4
	3	8,832	8,832	9,888	80	1,360	1,360	8	4
	4	8,832	8,832	9,888	79	1,360	1,360	8	4
	5	8,832	8,832	9,888	80	1,360	1,360	8	4
	6	8,832	8,832	9,888	81	1,360	1,360	8	4
	7	8,832	8,832	9,888	80	1,360	1,360	8	4
	8	8,832	8,832	9,888	81	1,360	1,360	8	4
	9	8,832	8,832	9,888	80	1,360	1,360	8	4
	10	8,832	8,832	9,888	81	1,360	1,360	8	4
	11	8,832	8,832	9,888	80	1,360	1,360	8	4
	12	8,832	8,832	9,888	79	1,360	1,360	8	4
	13	8,832	8,832	9,888	80	1,360	1,360	8	4
	14	4,504	4,504	4,504	79	680	680	4	4

≀: Including initial key XOR.

☆: Regular version.

◎: Shallow version.

◈: Shallow/low depth version.

(b) Using S-box with Toffoli depth 3

AES		#CNOT ✲			#NOT ✱	#Toffoli ☆		*TD* ✦	
❀	Round	☆	◎	◈	☆◎◈	☆	◎◈	☆	◎◈
128	1≀	14,640	7,904	8,960	79	3,120	1,560	6	3
	2	14,512	14,640	15,696	79	3,120	3,120	6	3
	3	14,512	14,640	15,696	81	3,120	3,120	6	3
	4	14,512	14,640	15,696	81	3,120	3,120	6	3
	5	14,512	14,640	15,696	81	3,120	3,120	6	3
	6	14,512	14,640	15,696	79	3,120	3,120	6	3
	7	14,512	14,640	15,696	79	3,120	3,120	6	3
	8	14,512	14,640	15,696	81	3,120	3,120	6	3
	9	14,512	14,640	15,696	80	3,120	3,120	6	3
	10	7,344	7,408	7,408	80	1,560	1,560	3	3
192	1≀	14,704	14,736	15,792	79	3,120	3,120	6	3
	2	14,576	14,672	15,728	79	3,120	3,120	6	3
	3	11,632	11,696	12,752	64	2,496	2,496	6	3
	4	14,576	14,608	15,664	81	3,120	3,120	6	3
	5	14,576	14,672	15,728	81	3,120	3,120	6	3
	6	11,632	11,696	12,752	64	2,496	2,496	6	3
	7	14,576	14,608	15,664	81	3,120	3,120	6	3
	8	14,576	14,672	15,728	79	3,120	3,120	6	3
	9	11,632	11,696	12,752	64	2,496	2,496	6	3
	10	14,576	14,608	15,728	79	3,120	3,120	6	3
	11	14,576	7,872	8,928	81	3,120	1,560	6	3
	12	5,824	5,824	5,824	64	1,248	1,248	3	3
256	1≀	11,760	6,320	7,376	64	2,496	1,248	6	3
	2	14,512	13,152	14,208	79	3,120	2,808	6	3
	3	14,512	14,512	15,568	80	3,120	3,120	6	3
	4	14,512	14,512	15,568	79	3,120	3,120	6	3
	5	14,512	14,512	15,568	80	3,120	3,120	6	3
	6	14,512	14,512	15,568	81	3,120	3,120	6	3
	7	14,512	14,512	15,568	80	3,120	3,120	6	3
	8	14,512	14,512	15,568	81	3,120	3,120	6	3
	9	14,512	14,512	15,568	80	3,120	3,120	6	3
	10	14,512	14,512	15,568	81	3,120	3,120	6	3
	11	14,512	14,512	15,568	80	3,120	3,120	6	3
	12	14,512	14,512	15,568	79	3,120	3,120	6	3
	13	14,512	14,512	15,568	80	3,120	3,120	6	3
	14	7,344	7,344	7,344	79	1,560	1,560	3	3

≀: Including initial key XOR.

☆: Regular version.

◎: Shallow version.

◈: Shallow/low depth version.