IEEE Transactions on Quantum Engineering

Received August 13, 2019; revised December 10, 2019; accepted December 13, 2019; date of publication January 16, 2020; date of current version February 14, 2020.

Digital Object Identifier 10.1109/TQE.2020.2965697

Reducing the Cost of Implementing the Advanced Encryption Standard as a Quantum Circuit

BRANDON LANGENBERG[1,2], HAI PHAM[3], AND RAINER STEINWANDT[4]

[1]PQSecure Technologies, Boca Raton, FL 33431 USA
[2]Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA
[3]West Campus Math Department Valencia College, Orlando, FL 32811 USA
[4]Department of Mathematical Sciences, Florida Atlantic University, Boca Raton, FL 33431 USA

The work of R. Steinwandt was supported in part by NATO SPS Project G5448 and in part by NIST awards 60NANB18D216 and 60NANB18D217. International Workshop on Quantum Resource Estimation, Phoenix, Arizona, USA, 22 June 2019. Corresponding author: Hai Pham (hpham29@valenciacollege.edu)

ABSTRACT To quantify security levels in a postquantum scenario, it is common to use the quantum resources needed to attack the Advanced Encryption Standard (AES) as a reference value. Specifically, in the National Institute of Standards and Technology's ongoing postquantum standardization effort, different security categories are defined that reflect the quantum resources needed to attack AES-128, AES-192, and AES-256. This article presents a quantum circuit to implement the S-box of AES. Also, leveraging an improved implementation of the key expansion, we identify new quantum circuits for all three AES key lengths. For AES-128, the number of Toffoli gates can be reduced by more than 88% compared to Almazrooie *et al.*'s and Grassl *et al.*'s estimates while simultaneously reducing the number of qubits. Our circuits can be used to simplify a Grover-based key search for AES.

INDEX TERMS Advanced Encryption Standard (AES), Grover's algorithm, quantum circuit, quantum cryptanalysis, quantum engineering.

I. INTRODUCTION

Reacting to progress in the development of quantum computers, the National Institute of Standards and Technology (NIST) has initiated a process to standardize cryptographic primitives that are designed to remain secure in the presence of large-scale quantum computers [15]. To fix security categories, NIST's call for proposals offers the quantum resources for an exhaustive key search in the case of AES-128, AES-192, and AES-256 as a reference point. Relevant cost measures include the number of qubits, the number of T- and Clifford gates, and the T-depth. It is not hard to see that with exception of the highly structured S-box—the `SubByte` transform—all of the Advanced Encryption Standards (AES) can be implemented by means of NOT and CNOT gates.

A. CONTRIBUTIONS

In the following, we present a new quantum circuit to implement `SubByte`, which builds on a result by Boyar and Peralta [7]. This approach allows a substantial reduction in the number of T-gates compared to the quantum circuits proposed by Grassl *et al.* [9] and, more recently, by Almazrooie *et al.* [1]. Our circuit requires 32 qubits, 55 Toffoli gates, 314 CNOT gates, 4 NOT gates, Toffoli depth 40, and a total [neutral current transformer (NCT)] depth of 298, including "cleaning up" ancillas—a reduction of the Toffoli count by more than 88%. There are different options to compile Toffoli gates into Clifford and T-gates, and the common quantum cryptanalytic approach is to first express AES as an NCT circuit, i.e., with NOT, CNOT, and Toffoli gates. Consequently, in this article, we stay at the NCT level, leaving the choice of a particular decomposition of Toffoli gates into more elementary building blocks to a subsequent synthesis step.

Moreover, building on [1] and [9], we present new quantum circuits for all three standardized key lengths of AES, which simultaneously offer savings in the number of qubits, the number of Toffoli gates, and the number of Clifford gates.

B. ORGANIZATION

First, we briefly recall the structure of the S-box of AES and survey prior work to express this functionality as a quantum circuit. Thereafter, we present our design for implementing `SubByte` in the NCT gate set and integrate it into quantum circuits for AES-128, AES-192, and AES-256. We conclude

with updated cost estimates for an exhaustive key search with Grover's algorithm for AES.

II. PRELIMINARIES

For a full specification of AES we refer to [14], but we briefly recall the algebraic structure of `SubByte`.

A. S-BOX OF AES

The AES algorithm uses multiple different transformations, but as detailed in [9], with the exception of `SubByte` all needed calculations can be expressed with NOT and CNOT gates alone. The nonlinear `SubByte` transformation takes a 1-B input $\vec{b} \in \mathbb{F}_2^8$ and substitutes it with a byte $S(\vec{b}) \in \mathbb{F}_2^8$ obtained by applying the following two operations.

1) Interpret $\vec{b}$ as a coefficient vector of an element $b \in \mathbb{F}_2[z]/(z^8+z^4+z^3+z+1)$, and replace $\vec{b}$ with the bitstring $\vec{b'}$ corresponding to b^{-1}. For $\vec{b}=\vec{0}$, set $\vec{b'}=\vec{b}$.
2) Apply an affine transformation, which consists of multiplication by an invertible matrix followed by the addition of a vector

$$\vec{b'} \longmapsto \begin{bmatrix} 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1 \end{bmatrix} \cdot \vec{b'} + \begin{bmatrix} 1\\1\\0\\0\\0\\1\\1\\0 \end{bmatrix}.$$

In [14, Figure 7], `SubByte` is expressed as a traditional substitution table, but when aiming at an efficient quantum circuit, one can try to leverage the available algebraic structure.

1) Observing that the map S that defines `SubBye` is a permutation on $\mathbb{F}_2^8$ and therewith inherently reversible, we can try to reduce the number of qubits by evaluating S "in place."
2) The affine map in the second step can be expressed with NOT and CNOT gates only, and one can focus on minimizing resources for the binary field inversion step, possibly exploiting the presence of intermediate fields.

B. PRIOR WORK TO IMPLEMENT `SubByte` AS QUANTUM CIRCUIT

Several authors looked into implementations of AES and its S-box as quantum circuits. In 2016, Grassl *et al.* [9] report a quantum circuit that builds on the observation that `SubByte` is a permutation. They offer a circuit that maps $|x\rangle$ with $x \in \mathbb{F}_2^8$ to $|S(x)\rangle$ using only a single ancilla. However, the circuit (found by solving a word problem in a permutation group) is quite large: while it requires only 9 qubits, it uses 1385 Toffoli plus 1551 CNOT or NOT gates.

In [9], another circuit is offered, which exploits the algebraic structure of `SubByte` and maps the input $|x\rangle\,|0^{32}\rangle$ to $|x\rangle\,|S(x)\rangle\,|0^{24}\rangle$. With 40 qubits, this circuit needs only 512 Toffoli along with 469 CNOT and 4 NOT gates. Kim *et al.* [13] suggest an improvement to the design in [9], saving one $\mathbb{F}_{2^8}$-multiplication. Almazrooie *et al.* [1] improve on Grassl *et al.*'s Toffoli count, using the same number of $\mathbb{F}_{2^8}$-multiplications as Kim *et al.* Exploiting again the algebraic structure of the S-box, Almazrooie *et al.* identify a quantum circuit with 56 qubits, 448 Toffoli gates, 494 CNOT gates, and 4 NOT gates. Banegas, in his forthcoming Ph.D. dissertation [4], reports a circuit on 40 qubits with the same number of Toffoli gates as Almazrooie *et al.* and Kim *et al.*, using only 425 CNOT and 4 NOT gates.

When aiming at a reduction of T-gates and T-depth, work on classical reversible circuits offers helpful results, and it seems these have not been fully leveraged yet. For instance, in 2018, Saravanan and Kalpana [16] suggest an implementation of `SubByte` involving only 35 Toffoli, 152 CNOT, and 4 NOT gates. This design (which again exploits the algebraic structure of the S-box) produces dozens of "garbage outputs," and for our purposes the cost to "clean up" wires is to be taken into account. Still, combining Bennett's method [5] with [16] leads to a quantum circuit with a Toffoli count of only $2 \cdot 35 = 70$, less than one-sixth of the Toffoli counts in [1] and [9].

Exploring the S-box in AES from the perspective of identifying a low-depth combinational circuit, Boyar and Peralta present, in [8], a proposal with only 34 AND gates. Their design again leverages the algebraic structure of `SubByte`, and by naïvely combining Boyar and Peralta's work with Bennett's method, we could derive a quantum circuit for the S-box of AES with no more than 68 Toffoli gates—though possibly a solid number of ancillas. As starting point for our work, we use an older design by Boyar and Peralta [7], which involves only 32 AND gates to evaluate `SubByte`. In the following, we transform the latter into a quantum circuit for `SubByte` that avoids the direct application of Bennett's method. In particular, this limits the number of Toffoli gates to 55, including all necessary "clean up."

III. PROPOSED QUANTUM CIRCUIT FOR THE S-BOX IN AES

In [7], Boyar and Peralta discuss a technique for combinational logic optimization, which involves two steps. The first step identifies nonlinear circuit components and reduces the number of AND gates—which, for our purposes, can be interpreted as saving Toffoli gates. The second step finds maximal linear components of the circuit and minimizes the number of XOR gates needed—therewith reducing the number of CNOT gates.

A. DECOMPOSITION OF THE S-BOX BY BOYAR AND PERALTA

Making use of the intermediate fields $\mathbb{F}_2 < \mathbb{F}_{2^2} < \mathbb{F}_{2^4} < \mathbb{F}_{2^8}$, Boyar and Peralta derive a representation $S(\vec{x}) = B \cdot$

$$
\begin{array}{lllllllll}
t_2 = y_{12} \cdot y_{15} & & t_3 = y_3 \cdot y_6 & & t_4 = t_3 + t_2 \\
t_5 = y_4 \cdot x_7 & & t_6 = t_5 + t_2 & & t_7 = y_{13} \cdot y_{16} \\
t_8 = y_5 \cdot y_1 & & t_9 = t_8 + t_7 & & t_{10} = y_2 \cdot y_7 \\
t_{11} = t_{10} + t_7 & & t_{12} = y_9 \cdot y_{11} & & t_{13} = y_{14} \cdot y_{17} \\
t_{14} = t_{13} + t_{12} & & t_{15} = y_8 \cdot y_{10} & & t_{16} = t_{15} + t_{12} \\
t_{17} = t_4 + t_{14} & & t_{18} = t_6 + t_{16} & & t_{19} = t_9 + t_{14} \\
t_{20} = t_{11} + t_{16} & & t_{21} = t_{17} + y_{20} & & t_{22} = t_{18} + y_{19} \\
t_{23} = t_{19} + y_{21} & & t_{24} = t_{20} + y_{18} & & \\
 & & & & \\
t_{25} = t_{21} + t_{22} & & t_{26} = t_{21} \cdot t_{23} & & t_{27} = t_{24} + t_{26} \\
t_{28} = t_{25} \cdot t_{27} & & t_{29} = t_{28} + t_{22} & & t_{30} = t_{23} + t_{24} \\
t_{31} = t_{22} + t_{26} & & t_{32} = t_{31} \cdot t_{30} & & t_{33} = t_{32} + t_{24} \\
t_{34} = t_{23} + t_{33} & & t_{35} = t_{27} + t_{33} & & t_{36} = t_{24} \cdot t_{35} \\
t_{37} = t_{36} + t_{34} & & t_{38} = t_{27} + t_{36} & & t_{39} = t_{29} \cdot t_{38} \\
t_{40} = t_{25} + t_{39} & & & & \\
 & & & & \\
t_{41} = t_{40} + t_{37} & & t_{42} = t_{29} + t_{33} & & t_{43} = t_{29} + t_{40} \\
t_{44} = t_{33} + t_{37} & & t_{45} = t_{42} + t_{41} & & z_0 = t_{44} \cdot y_{15} \\
z_1 = t_{37} \cdot y_6 & & z_2 = t_{33} \cdot x_7 & & z_3 = t_{43} \cdot y_{16} \\
z_4 = t_{40} \cdot y_1 & & z_5 = t_{29} \cdot y_7 & & z_6 = t_{42} \cdot y_{11} \\
z_7 = t_{45} \cdot y_{17} & & z_8 = t_{41} \cdot y_{10} & & z_9 = t_{44} \cdot y_{12} \\
z_{10} = t_{37} \cdot y_3 & & z_{11} = t_{33} \cdot y_4 & & z_{12} = t_{43} \cdot y_{13} \\
z_{13} = t_{40} \cdot y_5 & & z_{14} = t_{29} \cdot y_2 & & z_{15} = t_{42} \cdot y_9 \\
z_{16} = t_{45} \cdot y_{14} & & z_{17} = t_{41} \cdot y_8 & &
\end{array}
$$

FIGURE 1. Nonlinear portion $F : \mathbb{F}_2^{22} \longrightarrow \mathbb{F}_2^{18}$, $(x_7, y_1, y_2, \ldots, y_{21}) \longmapsto (z_0, \ldots, z_{17})$ of the `SubByte` S-box in AES as given in [7, Appendix C, Figure 3].

$F(U \cdot \vec{x})$ with matrices $B \in \mathbb{F}_2^{8\times 18}$, $U \in \mathbb{F}_2^{22\times 8}$, and a nonlinear function $F : \mathbb{F}_2^{22} \longrightarrow \mathbb{F}_2^{18}$. The matrices B and U are given in [7, Appendix A], and the function F can be computed as shown in FIGURE 1.

From this, we see that no more than 32 Toffoli gates are needed to evaluate `SubByte`, but we still need to take care of "cleaning up" ancillas—and would like to keep the number of qubits small. To optimize the linear portion of `SubByte`, Boyar and Peralta derive short linear programs, which we do not reproduce here; they are available in [7, Appendix C, FIGURES 2 and 4] and involve XOR and XNOR operations only. The four NOT gates in our quantum circuit originate in the four XNOR gates used by Boyar and Peralta.

B. DERIVING A COMPACT QUANTUM CIRCUIT

A naïve conversion of Boyer and Peralta's circuit yields a quantum circuit with 126 qubits, 32 Toffoli gates, 166 CNOT gates, and 4 NOT gates—not yet taking into account the "clean up" cost. The circuit we aim at is to map $|\vec{x}\rangle\,|0^a\rangle$ to $|\vec{x}\rangle\,S(\vec{x})\,|0^{a-8}\rangle$ with a small number a of ancilla qubits. Our circuit uses $a = 24$. We also identified a circuit with $a = 23$, but that circuit came at the expense of increasing the Toffoli count by 2, and our primary objective is to reduce the number of Toffoli gates.

To reduce the number of qubits in a straightforward translation, we notice that certain wires, after being accessed for a few immediate computations, remain idle until the end. Uncomputing these wires early on enables us to reuse them instead of introducing additional ancillas. Another observation is that wires that store the output of the S-box do not need to be cleaned up. Thus, we try to have Toffoli gates applied directly to those wires to avoid involving them in the clean-up process. Also, some computations would target a wire, and later on, the result is just added somewhere else. We try to place gates so that such "intermediate wires" are avoided. The final circuit we obtain, including "cleaning up" requires 32 qubits, 55 Toffoli, 314 CNOT, and 4 NOT gates. The Toffoli depth is 40, and the overall S-box depth is 298. FIGURE 2 gives a high-level view of the circuit and Appendix A gives a detailed description.

To produce the circuit description in Appendix A, we used the open-source software framework for quantum computing ProjectQ [11], [18]. The 8-b input of `SubByte` is represented by `U`; `T` and `Z` represent ancillas, which are used in the intermediate computations and returned to zero at the end, and `S` represents the output of the S-box.

The main portion of the source code is the translation of equations in FIGURE 1. We treat `U[0]`,..., `U[7]` as basis elements, and we update them as we progress to provide needed input values for a calculation. For instance, to compute t_2, we first compute y_{12} and y_{15} and then apply a Toffoli gate. The value y_{12} can be obtained as a linear combination of `U[0]`, `U[3]`, `U[5]`, and `U[6]`, and we store this result on `U[5]`. Similarly, y_{15} is a linear combination of `U[0]`, `U[3]`, `U[4]`, and `U[6]`, and we store this result on `U[4]`. The Toffoli will use `U[5]` and `U[4]` as controls and target `T[0]`, which is t_2 at this moment. Our basis elements remain the same except for `U[4]` and `U[5]`. We have `U[0]` + `U[3]` + `U[4]` + `U[6]` and `U[0]` + `U[3]` + `U[5]` + `U[6]` for the next computation. We repeat this technique until t_{45} is computed. Notice that we are able to reuse the qubit `T[8]`.

The computations annotated by "for z16" to "for z14" are preparations for later usages, because we do not want to uncompute Toffoli gates that can target output qubits directly. Some of z_i can be computed directly onto an output qubit and copied to other designated locations. For others, we compute them onto the ancilla `Z[0]`, then copy the result to the needed output qubits before cleaning up `Z[0]` for reuse.

IV. QUANTUM RESOURCE ESTIMATES FOR AES-{128, 192, 256}

Aside from the reduced S-box above, we offer a reduction for AES in terms of circuit depth as well as a number of qubits over prior work in [1] and [9]. This saving is due to a new cost-saving design in the architecture of the key expansion along with the reduced qubit requirements of our S-box. For our round generation, we adopt the "zig-zag" method from [9]. This is kept identical in AES-128 and AES-256, and a minimal change is made at the very end for AES-192. Expanding on ideas discussed in [1], we recognize that by storing all k_{4n+3} for AES-128 and AES-256 and k_{4n+5} for AES-192, where n represents the round number, we could not only use a combination to construct future keys, but also gain the ability to remove keys once they are no longer used in future constructions. Since there is a direct correlation between T-depth and "S-box depth" (as the S-box is the only use of

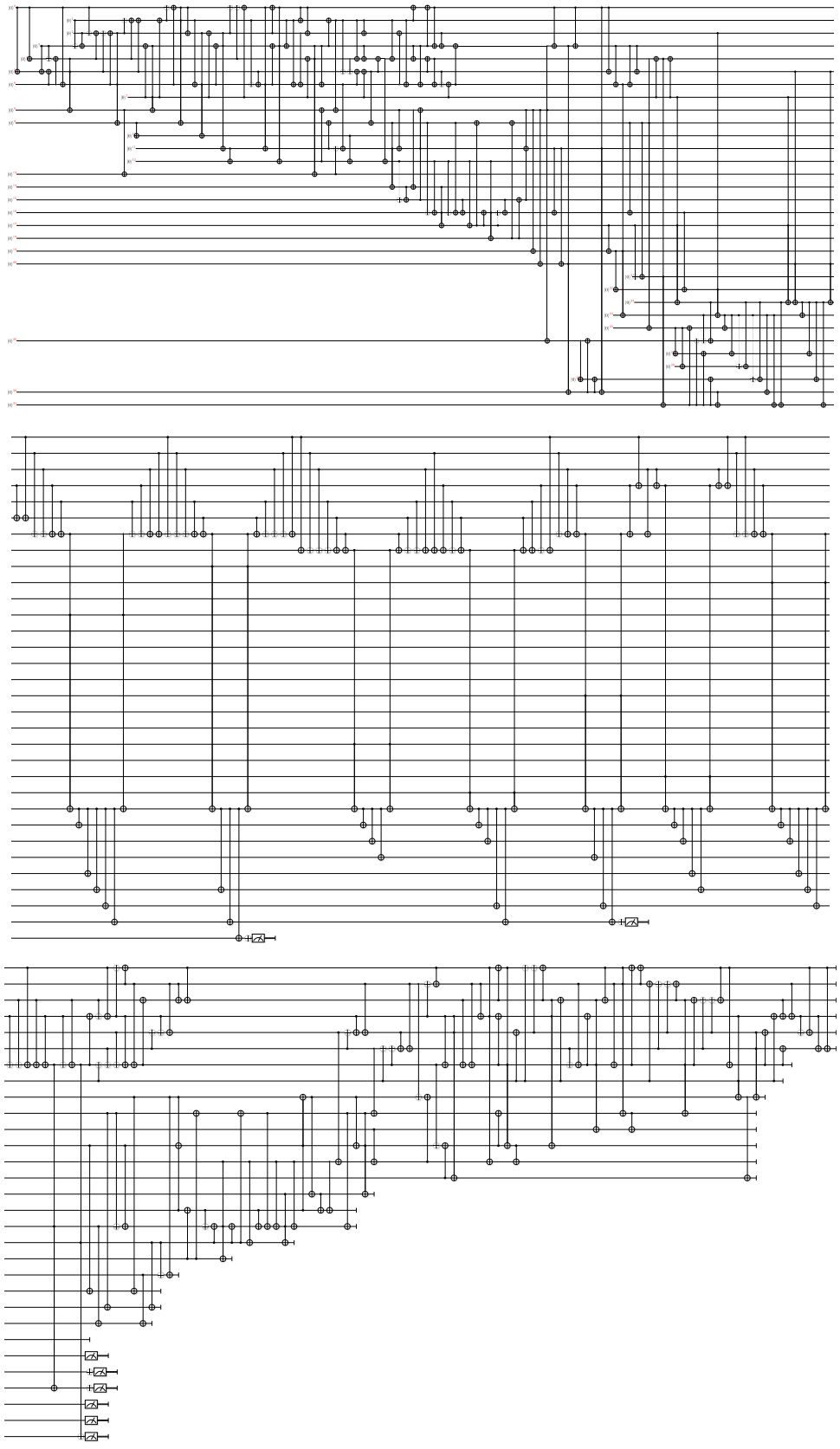

FIGURE 2. Circuit diagram for implementing `SubByte` with 32 qubits and 55 Toffoli gates; the input value x is stored on the topmost eight wires; the output $S(x)$ of `SubByte` is stored on the last eight wires.

k_4	:	k_3, k_0	k_5	:	k_4, k_1	k_6	:	k_5, k_2	k_7	:	k_6, k_3
k_8	:	k_7, k_3, k_2, k_1	k_9	:	k_8, k_7, k_3, k_2	k_{10}	:	k_7, k_3	k_{11}	:	k_{10}, k_7
k_{12}	:	k_{11}, k_7, k_2	k_{13}	:	k_{12}, k_{11}, k_3	k_{14}	:	k_{13}, k_{11}, k_7	k_{15}	:	k_{14}, k_{11}
k_{16}	:	k_{15}, k_{11}, k_7, k_3	k_{17}	:	k_{16}, k_{15}, k_7	k_{18}	:	k_{17}, k_{15}, k_{11}	k_{19}	:	k_{18}, k_{15}
k_{20}	:	$k_{19}, k_{15}, k_{11}, k_7$	k_{21}	:	k_{20}, k_{19}, k_{11}	k_{22}	:	k_{21}, k_{19}, k_{15}	k_{23}	:	k_{22}, k_{19}
k_{24}	:	$k_{23}, k_{19}, k_{15}, k_{11}$	k_{25}	:	k_{24}, k_{23}, k_{15}	k_{26}	:	k_{25}, k_{23}, k_{19}	k_{27}	:	k_{26}, k_{23}
k_{28}	:	$k_{27}, k_{23}, k_{19}, k_{15}$	k_{29}	:	k_{28}, k_{27}, k_{19}	k_{30}	:	k_{29}, k_{27}, k_{23}	k_{31}	:	k_{30}, k_{27}
k_{32}	:	$k_{31}, k_{27}, k_{23}, k_{19}$	k_{33}	:	k_{32}, k_{31}, k_{23}	k_{34}	:	k_{33}, k_{31}, k_{27}	k_{35}	:	k_{34}, k_{31}
k_{36}	:	$k_{35}, k_{31}, k_{27}, k_{23}$	k_{37}	:	k_{36}, k_{35}, k_{27}	k_{38}	:	k_{37}, k_{35}, k_{31}	k_{39}	:	k_{38}, k_{35}
k_{40}	:	$k_{39}, k_{35}, k_{31}, k_{27}$	k_{41}	:	k_{40}, k_{39}, k_{31}	k_{42}	:	k_{41}, k_{39}, k_{35}	k_{43}	:	k_{42}, k_{39}

FIGURE 3. Keys required to construct each key in AES-128. The leftmost column requires four S-boxes, while the rightmost column is what is stored at the end of each round.

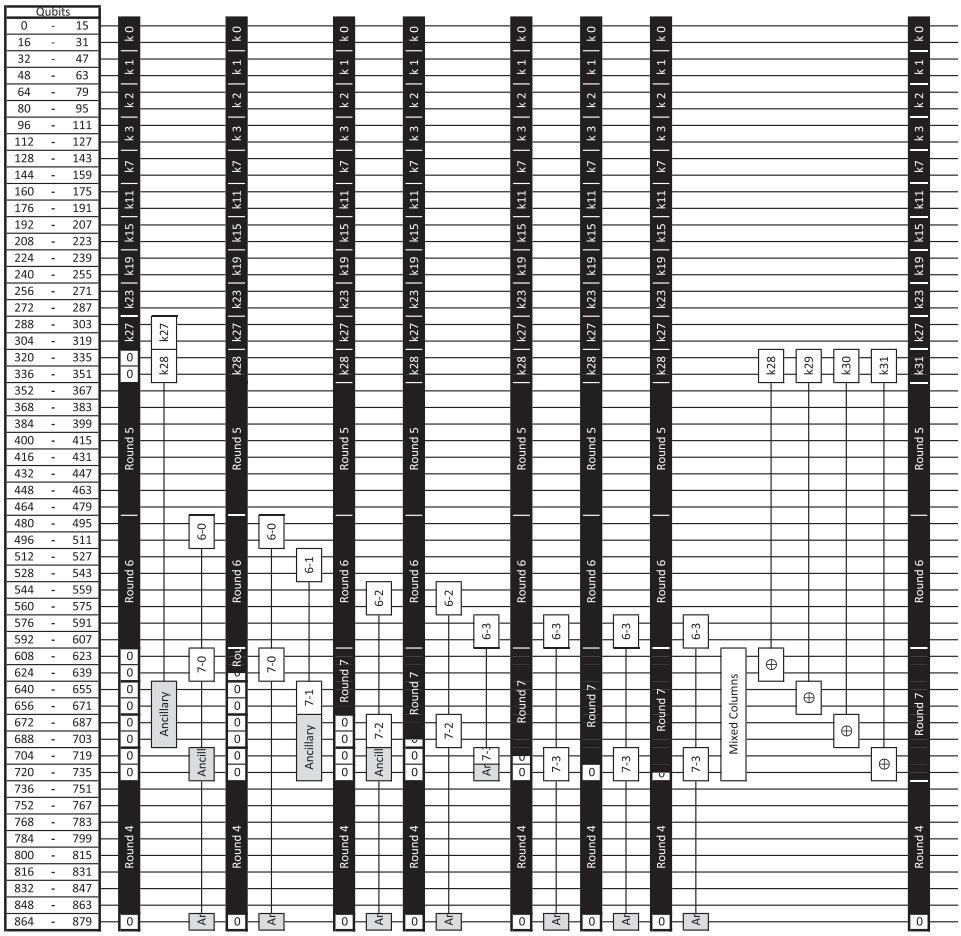

FIGURE 4. AES-128 diagram of Round 7 computations that have an S-box depth of seven. Each column represents an S-box depth of one.

T-gates in our construction), we stop at S-box depth; the S-box can be replaced if a different design is preferred. Note the S-box design proposed in this article uses 8 less qubits than in [9] and 24 less qubits than in [1], which allows for greater parallelization and, thus, a generally reduced "S-box depth."

A. SAVINGS IN AES-128

As part of the key expansion, various keywords k_i are computed. Each k_{4n} requires the use of four S-boxes and an XOR of previous keys. After Round 3 (after k_{12}), all keys have a similar structure that can be seen in Figure 3. In our design, we store k_{4n+3} once round n has been fully computed. To save depth, each keyword will be constructed at the same time as the round it is used in, except for round one. This is because the plaintext and cipher key (k_0, k_1, k_2, and k_3) are XORed together to produce Round 0. However, both are required to construct Round 1, so Round 1 and k_4 must be constructed at sequential times. For the remaining rounds, the

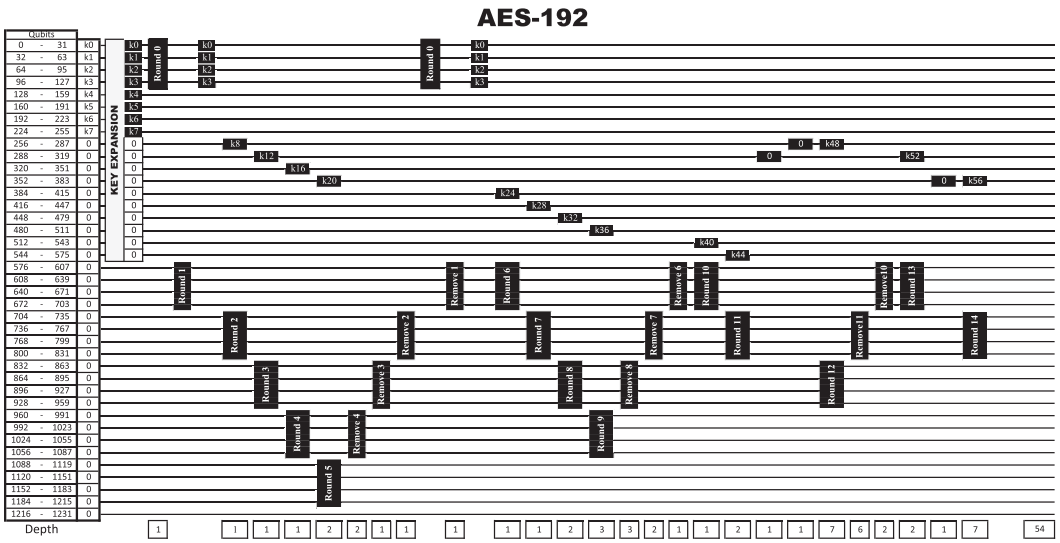

FIGURE 5. AES-256 circuit diagram showing when keys are constructed and the S-box depth of each round computation.

parallelization greatly reduces depth. For example, during Round 2, all S-box computations for k_8 as well as Round 2 can be computed with an S-box depth of one, using 320 auxiliary qubits. Once these S-boxes and `MixColumns` are computed, k_8 can be `XOR`ed onto Round 1, followed by the construction of k_9, k_{10}, and k_{11}, each being `XOR`ed onto the round one construction. Thus, Round 2 is fully computed, and k_{11} is stored, and this entire construction took a total S-box depth of one. When 320 auxiliary qubits are not available, not all S-box computations can be done in parallel and the depth must be increased (up to 7). Round 1 (without k_4), Round 2, removing Round 1, and Round 5 all are computed with an S-box depth of one.

While computing the keywords along with the rounds substantially reduces the overall depth, it does mean when auxiliary qubits are unavailable, the 20 S-boxes (16 for the round and 4 for the key) may require an increased S-box depth to be computed. While Round 2 has an S-box depth of one, Round 7 has an S-box depth of 7 (see Figure 4) since there are only 16 auxiliary qubits available (plus any qubits that have not stored part of Round 7 so far). Sometimes, round keys can be computed during the clean up of previous rounds. For example, when reversing and cleaning up Round 8, the key for Round 10 (k_{40}) can be computed, thus needing only 16 S-boxes with a depth of 6 to compute Round 10 and, hence, complete the computations of AES-128.

By storing k_{4n+3} for each $n \leq 7$, once we get to Round 7, and store keyword k_{31}, we can remove keywords k_{15}, k_{11}, and k_7 from Rounds 3, 2, and 1, thus gaining storage space to place keywords for Rounds 8, 9, and10 in this space. This removal is done using S-boxes in reverse after the keys are returned to their k_{4n} values. This is equivalent to the "zig-zag" method used in [9] to remove rounds, but here we use it to remove keys. This saves 96 qubits over [9] and 64 qubits over [1] who only removed one keyword. Since each keyword uses four S-boxes, the removal requires the use of 12 additional S-boxes for a substantial savings in qubits. The removal of keyword k_{15} can be done during the removal of Round 5 without additional depth. Similarly, the removal of keyword k_7 can be done during the construction of Round 8 without additional depth. Thus, the S-box depth for the key expansion is two, which includes computing k_4 and removing k_{11}, both with an S-box depth of one. If in the future, it is found the savings in qubits is not worth the additional gates and a depth of three S-boxes, this can be ignored and extra qubits can be used. The total depth of this circuit uses 47 S-boxes, 15 `MixColumns` computations with a depth of 39 each, and a depth of 142 to apply the `AddRoundKey` to each round.

B. SAVINGS IN AES-192

AES-192 differs slightly in the key generation. Recall AES-192 only uses an S-box for every sixth key, and since only four keys are needed per round, some rounds only need 16 S-boxes to be fully computed and some need 16 plus the additional 4 for the key generation. So even though there are more rounds than in AES-128, there are less keywords generated. By the time, keyword k_{48} needs to be computed, k_{11} and smaller keys are no longer needed, thus k_{11} can be reversed to k_6 and, then, removed using an inverted S-box, thus saving 32 qubits for an additional 4 S-boxes.

Also, the "zig-zag" method used in [9] used the same amount of qubits for AES-256 as it did for AES-192. This means there is a room for additional rounds or space savings. While we did not reduce the number of qubits for the round generation, we were able to use some of this additional space for the key expansion. Instead of placing Round 12 on the remaining 128 qubits, we can reverse part of Round 10 and

TABLE 1. Quantum resources to implement AES.

	#NOT	#CNOT	#Toffoli	S-box Depth	Toffoli Depth	#Qubits
AES-128	1,507	107,960	16,940	47	1,880	864
AES-192	1,692	125,580	19,580	41	1,640	896
AES-256	1,992	151,011	23,760	54	2,160	1,232

TABLE 2. Resource estimates for AES using designs from prior literature.

	Grassl et al. [9]					Almazrooie et al. [1]				
	#NOT	#CNOT	#Toffoli	Toffoli Depth	#qubits	#NOT	#CNOT	#Toffoli	Toffoli Depth	#qubits
AES-128	1,456	166,548	151,552	12,672	984	1,370	192,832	150,528	*(not reported)*	976
AES-192	1,608	189,432	172,032	11,088	1,112	*(not reported)*				
AES-256	1,943	233,836	215,040	14,976	1,336	*(not reported)*				

reuse those qubits to store part of Round 12, thus gaining enough qubits to store round keys. Thus, when keyword k_{42} is generated, it is generated below where Round 11 is stored, thus saving another 32 qubits for the cost of another 4 inverted S-boxes. Overall, we were able to save 64 qubits over the results in [9]. The total depth of this circuit uses 41 S-boxes, 18 `MixColumns`, and a depth of 208 to apply `AddRoundKey` to each round.

C. SAVINGS IN AES-256

The methods for AES-128 in Section IV-A above apply equally here, but since the key constructions in AES-256 require more previous keys, the removal of keys is not as simple. However, after Round 11 and key k_{47} is constructed, keys for rounds three and two (k_{15} and k_{11}) can be removed in the same fashion as above, and keys for Rounds 12 and 13 (k_{51} and k_{55}) can be stored in their place. Also, after Round 13, key k_{23} can be removed and replaced with key material for Round 14 (k_{59}). This is a total saving of 96 qubits for the increased costs of 12 S-boxes with a total additional depth of 3 S-boxes. The total depth of this circuit uses 54 S-boxes (see Figure 5), 22 `MixColumns`, and a depth of 267 to apply `AddRoundKey` to each round.

This method of computing the keywords during the round generation and only storing k_{4n+3} for AES-128 and AES-256, and k_{4n+5} for AES-192 means the keys between k_{4n} and this key need to be computed several times; however, this method is comparable to other methods of producing the additional keys directly in the rounds.

TABLE 1 summarizes the resources needed to implement AES with the approach suggested here. For comparison, we also recall resource counts for designs proposed in [1] and [9]. Comparing TABLE 2 with TABLE 1, we see that the revised S-box design in combination with the changes to handling the key expansion enables attractive resource savings.

V. EXHAUSTIVE KEY SEARCH WITH GROVER'S ALGORITHM

For our discussion of leveraging Grover's algorithm for an exhaustive key search, we follow the approach in [9], i.e., we assume a straightforward application of Grover's algorithm, using our AES design to implement the pertinent Grover operator. We leave it for future work to explore possible time–space tradeoffs in the spirit of Kim *et al.*'s work [13]. For Grover's algorithm [10], we need a quantum circuit $U_f : |x\rangle\,|y\rangle \longmapsto |x\rangle\,|y \oplus f(x)\rangle$, where $x \in \{0,1\}^k$ represents a candidate key, and $f(x) = 1$ if the key x matches all given plaintext–ciphertext pairs, and $f(x) = 0$, otherwise. Following Amento-Adelmann *et al.* [2], we assume that $r_k = \lceil k/128 \rceil$ known plaintext–ciphertext pairs are sufficient to avoid false positives in an exhaustive key search for AES-k ($k \in \{128, 192, 256\}$). Thus, taking into account "cleaning up" of wires, we need to implement the following.

1) 2 AES instances (for $r_{128} = 1$ plaintext–ciphertext pair) for AES-128.
2) 4 AES instances (for $r_{192} = 2$ plaintext–ciphertext pairs) for AES-192.
3) 4 AES instances (for $r_{256} = 2$ plaintext–ciphertext pairs) for AES 256.

Here, we do not distinguish between implementing encryption or decryption, as the latter can be obtained by running encryption backward, thereby not affecting the cost parameters we are looking at.

Remark 1: The above choices for r_{128}, r_{192}, and r_{256} are smaller than the ones used by Grassl *et al.* [9], but Amento-Adelmann *et al.*'s [2] reasoning could be applied to argue for a smaller number of AES instances in [9], too. We are not offering anything novel here—Our contribution only affects the quantum circuit for AES.

A. NUMBER OF QUBITS

As noted in [2], multiple plaintext–ciphertext pairs can be tested sequentially or in parallel, trading gates and circuit depth for the number of qubits. Prioritizing a smaller T-depth, here we choose the parallel option, as in [9], which leads to a total qubit count of $r_k \cdot q_k + 1$, where q_k is the number of qubits needed to implement AES-k according to TABLE 1.

1) $1 \cdot 864 + 1 = 865$ qubits for a Grover-based key search in AES-128.
2) $2 \cdot 896 + 1 = 1793$ qubits for a Grover-based key search in AES-192.
3) $2 \cdot 1232 + 1 = 2465$ qubits for a Grover-based key search in AES-256.

B. GATE COUNTS

1) OPERATOR U_f

Inside the operator U_f, we need to compare the 128-b outputs of the AES instances with r_k given ciphertexts. For this, we can use a $128 \cdot r_k$-controlled NOT [plus some NOT gates, which we neglect and that depend on the given ciphertext(s).] We also budget $2 \cdot (r_k - 1) \cdot k$ CNOT gates to make the input key available to all r_k parallel AES instances (and uncomputing this operation at the end). And, of course, we need to implement the actual AES instances. From TABLE 1, we obtain the following resource estimates.

1) AES-128: Two AES-instances require $2 \cdot 16\,940 = 33\,880$ Toffoli gates with a Toffoli depth of $2 \cdot 1880 = 3760$. In addition, we need $2 \cdot 1507 = 3014$ NOT gates and $2 \cdot 107\,960 = 215\,920$ CNOT gates.
2) AES-192: Four AES-instances require $4 \cdot 19\,580 = 78\,320$ Toffoli gates with a Toffoli depth of $2 \cdot 1\,640 = 3280$. In addition, we need $4 \cdot 1692 = 6\,768$ NOT gates and $4 \cdot 125\,580 + 2 \cdot 192 = 502\,704$ CNOT gates.
3) AES-256: Four AES-instances require $4 \cdot 23\,760 = 95\,040$ Toffoli gates with a Toffoli depth of $2 \cdot 2160 = 4320$. In addition, we need $4 \cdot 1992 = 7968$ NOT gates and $4 \cdot 151\,011 + 2 \cdot 256 = 604\,556$ CNOT gates.

2) GROVER OPERATOR

Grover's algorithm repeatedly applies the operator

$$G = U_f \cdot \left(\left(H^{\otimes k} \left(2 \left| 0 \right\rangle \left\langle 0 \right| - \mathbf{1}_{2^k} \right) H^{\otimes k} \right) \otimes \mathbf{1}_2 \right)$$

where $|0\rangle$ is the all-zero basis state of appropriate size. So in addition to U_f, further gates are needed. In this article, we do not offer any improvements to those parts of the algorithm. Following [9], for the operator $2 |0\rangle \langle 0| - \mathbf{1}_{2^k}$, we budget a k-fold controlled NOT gate. With $\lfloor \frac{\pi}{4} \cdot 2^{k/2} \rfloor$ Grover iterations being used for AES-k, we can now give estimates in the Clifford+T model and compare our results with prior work.

C. OVERALL COST

The previous discussion does not rely on a particular translation from Toffoli gates to T-gates. For a comparison with prior work, we proceed similarly as in [9].

1) The number of T-gates to realize an ℓ-fold controlled NOT ($\ell \geq 5$) is estimated as $32 \cdot \ell - 84$ (see [19]).
2) Toffoli gates are assigned a cost of 7 T-gates plus 8 Clifford gates, a T-depth of 4, and a total depth of 8; this is motivated by the decomposition in [3, FIGURE 7(a)]. Certainly, other choices would be possible here; e. g., in [17], Selinger offers a Toffoli decomposition with T-depth 1, using 7 T-gates, 18 Clifford gates, and 4 ancillas.
3) To estimate the total number of Clifford gates, we count only the Clifford gates in the AES-instances, plus the $2 \cdot (r_k - 1) \cdot k$ CNOT gates inside U_f for the parallel processing of plaintext-ciphertext pairs.

TABLE 3. Revised resource estimates in the Clifford+T model for a Grover-based key search for AES-k.

	Grassl et al. [9]	this paper
AES-128:		
#qubits	$2,953$	865
#T-gates	$1.19 \cdot 2^{86}$	$1.47 \cdot 2^{81}$
T-depth	$1.06 \cdot 2^{80}$	$1.44 \cdot 2^{77}$
#Clifford gates	$1.55 \cdot 2^{86}$	$1.46 \cdot 2^{82}$
overall depth	$1.16 \cdot 2^{81}$	$1.39 \cdot 2^{79}$
AES-192:		
#qubits	$4,449$	$1,793$
#T-gates	$1.81 \cdot 2^{118}$	$1.68 \cdot 2^{114}$
T-depth	$1.21 \cdot 2^{112}$	$1.26 \cdot 2^{109}$
#Clifford gates	$1.17 \cdot 2^{119}$	$1.71 \cdot 2^{115}$
overall depth	$1.33 \cdot 2^{113}$	$1.23 \cdot 2^{111}$
AES-256:		
#qubits	$6,681$	$2,465$
#T-gates	$1.41 \cdot 2^{151}$	$1.02 \cdot 2^{147}$
T-depth	$1.44 \cdot 2^{144}$	$1.66 \cdot 2^{141}$
#Clifford gates	$1.83 \cdot 2^{151}$	$1.03 \cdot 2^{148}$
overall depth	$1.57 \cdot 2^{145}$	$1.61 \cdot 2^{143}$

4) To estimate depth and T-depth we only take the depth and T-depth of AES-k into account (ignoring in particular the two multi-controlled NOT gates). For the S-box used in this article, the (Clifford + T) depth is about 600.

With this, the estimated total cost for a Grover-based attack against AES-k is as follows.

1) AES-128:
   a) T-gates: $\lfloor \frac{\pi}{4} \cdot 2^{64} \rfloor \cdot (7 \cdot 33\,880 + 32 \cdot 128 - 84 + 32 \cdot 128 - 84) \approx 1.47 \cdot 2^{81}$ T-gates with a T-depth of $\lfloor \frac{\pi}{4} \cdot 2^{64} \rfloor \cdot 4 \cdot 3760 \approx 1.44 \cdot 2^{77}$.
   b) Clifford gates: $\lfloor \frac{\pi}{4} \cdot 2^{64} \rfloor \cdot (8 \cdot 33\,880 + 3014 + 215\,920) \approx 1.46 \cdot 2^{82}$.
   c) Circuit depth: $\lfloor \frac{\pi}{4} \cdot 2^{64} \rfloor \cdot 2 \cdot (47 \cdot 600 + 15 \cdot 39 + 142) \approx 1.39 \cdot 2^{79}$.
2) AES-192:
   a) T-gates: $\lfloor \frac{\pi}{4} \cdot 2^{96} \rfloor \cdot 7 \cdot (78\,320 + 32 \cdot 192 - 84 + 32 \cdot 256 - 84) \approx 1.68 \cdot 2^{114}$ T-gates with a T-depth of $\lfloor \frac{\pi}{4} \cdot 2^{96} \rfloor \cdot 4 \cdot 3280 \approx 1.26 \cdot 2^{109}$.
   b) Clifford gates: $\lfloor \frac{\pi}{4} \cdot 2^{96} \rfloor \cdot (8 \cdot 78\,320 + 6768 + 502\,704) \approx 1.71 \cdot 2^{115}$.
   c) Circuit depth: $\lfloor \frac{\pi}{4} \cdot 2^{96} \rfloor \cdot 2 \cdot (41 \cdot 600 + 18 \cdot 39 + 254) \approx 1.23 \cdot 2^{111}$
3) AES-256:
   a) T-gates: $\lfloor \frac{\pi}{4} \cdot 2^{128} \rfloor \cdot 7 \cdot (95\,040 + 32 \cdot 256 - 84 + 32 \cdot 256 - 84) \approx 1.02 \cdot 2^{147}$ T-gates with a T-depth of $\lfloor \frac{\pi}{4} \cdot 2^{128} \rfloor \cdot 4 \cdot 4320 \approx 1.66 \cdot 2^{141}$.
   b) Clifford gates: $\lfloor \frac{\pi}{4} \cdot 2^{128} \rfloor \cdot (8 \cdot 95\,040 + 7968 + 604\,556) \approx 1.03 \cdot 2^{148}$.
   c) Circuit depth: $\lfloor \frac{\pi}{4} \cdot 2^{128} \rfloor \cdot 2 \cdot (54 \cdot 600 + 22 \cdot 39 + 267) \approx 1.61 \cdot 2^{143}$.

TABLE 3 summarizes the main resource counts; for comparison, we also include the estimates reported in [9].

VI. CONCLUSION

The previous discussion establishes that fewer quantum resources for an exhaustive key search in AES are required than previously reported. In particular, the number of Toffoli—and there with the number of costly T-gates—can be reduced. These savings can be achieved in tandem with reducing the T-depth, the number of Clifford gates, and the number of qubits needed. Even for AES-128, the established quantum resource estimates remain well beyond currently available technology and a Grover-attack stays well beyond current feasibility. However, for a quantitative interpretation of the security categories offered by NIST in [15], it may be helpful to take the revised resource estimates into account.

Follow-up work: Since the submission of this article, additional work on implementing AES as a quantum circuit has been published. Specifically, we would like to mention Jaques *et al.*'s work [12], which offers depth optimized circuits and leverages AND-gates, along with measurement-based uncomputation, to reduce the number of T-gates and T-depth.

ACKNOWLEDGMENT

The authors would like to thank M. Soeken for making us aware of the work in [8] during a discussion with one of the authors at a Dagstuhl Seminar on quantum cryptanalysis. The work of R. Steinwandt was supported by the North Atlantic Treaty Organization's Science for Peace and Security Project G5448 and in part by the National Institute of Standards and Technology under Grants 60NANB18D216 and 60NANB18D217.

APPENDIX

```
import math
from projectq.ops import CNOT, Measure, X, Toffoli
from projectq import MainEngine
from projectq.meta import Compute, Uncompute
from projectq.backends import CircuitDrawer,
 ResourceCounter, ClassicalSimulator
import projectq.libs.math

drawing_engine = CircuitDrawer()
resource_counter = ResourceCounter()
sim = ClassicalSimulator()
eng = MainEngine(sim)

def aes-box(eng):

  U = eng.allocate_qureg(8)

  T = eng.allocate_qureg(15)

  Z = eng.allocate_qureg(1)

  S = eng.allocate_qureg(8)

input_m = [0]*(8)
output_m = [0]*(8)

with Compute(eng):
  CNOT | (U[0], U[5])
  CNOT | (U[3], U[5])
  CNOT | (U[6], U[5])
  CNOT | (U[0], U[4])
  CNOT | (U[3], U[4])
  CNOT | (U[6], U[4])
  Toffoli | (U[5], U[4], T[0])  #t2
  CNOT | (T[0], T[5])

  CNOT | (U[1], U[3])
  CNOT | (U[2], U[3])
  CNOT | (U[7], U[3])
  Toffoli | (U[3], U[7], T[0])  #t6
  CNOT | (U[0], U[6])
  CNOT | (U[0], U[2])
  CNOT | (U[4], U[2])
  CNOT | (U[5], U[2])
  CNOT | (U[6], U[2])
  Toffoli | (U[6], U[2], T[1])  #t7
  CNOT | (T[1], T[2])

  CNOT | (U[2], U[1])
  CNOT | (U[4], U[1])
  CNOT | (U[5], U[1])
  CNOT | (U[7], U[1])
  CNOT | (U[1], U[0])
  CNOT | (U[6], U[0])
  Toffoli | (U[1], U[0], T[1])  #t9

  CNOT | (U[1], U[6])
  CNOT | (U[0], U[2])
  Toffoli | (U[6], U[2], T[2])  #t11

  CNOT | (U[6], U[3])
  CNOT | (U[7], U[2])
  Toffoli | (U[3], U[2], T[3])  #t12
  CNOT | (T[3], T[4])

  CNOT | (U[1], U[6])
  CNOT | (U[5], U[6])
  CNOT | (U[2], U[0])
  CNOT | (U[4], U[0])
  CNOT | (U[7], U[0])
  Toffoli | (U[6], U[0], T[3])  #t14

  CNOT | (U[6], U[3])
  CNOT | (U[2], U[0])
  Toffoli | (U[3], U[0], T[4])  #t16

  CNOT | (T[3], T[1])       #t19

  CNOT | (U[1], U[3])
```

```
CNOT | (U[7], U[4])
Toffoli | (U[3], U[4], T[5])  #t4

CNOT | (T[5], T[3])          #t17

CNOT | (T[4], T[0])          #t18

CNOT | (T[2], T[4])          #t20

CNOT | (U[1], U[6])
CNOT | (U[2], U[6])
CNOT | (U[3], U[6])
CNOT | (U[6], T[3])          #t21

CNOT | (U[0], U[1])
CNOT | (U[3], U[1])
CNOT | (U[1], T[0])          #t22

CNOT | (U[1], U[5])
CNOT | (U[4], U[5])
CNOT | (U[6], U[5])
CNOT | (U[7], U[5])
CNOT | (U[5], T[1])          #t23

CNOT | (U[1], U[4])
CNOT | (U[3], U[4])
CNOT | (U[5], U[4])
CNOT | (U[4], T[4])          #t24


Toffoli | (T[3], T[1], T[6])  #t26
CNOT | (T[0], T[3])          #t25

CNOT | (T[4], T[7])
CNOT | (T[6], T[7])          #t27

CNOT | (T[0], T[6])          #t31
Toffoli | (T[3], T[7], T[0])  #t29

CNOT | (T[1], T[8])
CNOT | (T[4], T[8])          #t30

Toffoli | (T[6], T[8], T[9])  #t32

#clean up T[8]:
CNOT | (T[4], T[8])
CNOT | (T[1], T[8])
#t[8] is free to reuse

CNOT | (T[4], T[9])          #t33
CNOT | (T[9], T[1])          #t34

CNOT | (T[7], T[8])
CNOT | (T[9], T[8])          #t35

Toffoli | (T[4], T[8], T[10]) #t36

#clean up T[8] again:
CNOT | (T[9], T[8])
CNOT | (T[7], T[8])
#t[8] is free to reuse

CNOT | (T[10], T[1])         #t37
CNOT | (T[10], T[7])         #t38

Toffoli | (T[0], T[7], T[3])  #t40

CNOT | (T[3], T[8])
CNOT | (T[1], T[8])          #t41

CNOT | (T[0], T[11])


CNOT | (T[9], T[11])         #t42

CNOT | (T[0], T[12])
CNOT | (T[3], T[12])         #t43

CNOT | (T[9], T[13])
CNOT | (T[1], T[13])         #t44

CNOT | (T[11], T[14])
CNOT | (T[8], T[14])         #t45

CNOT | (U[0], U[2])
CNOT | (U[1], U[2])
CNOT | (U[6], U[2])          #for z16

CNOT | (U[1], U[4])
CNOT | (U[3], U[4])
CNOT | (U[5], U[4])          #for z1

CNOT | (U[1], U[6])
CNOT | (U[3], U[6])
CNOT | (U[4], U[6])
CNOT | (U[5], U[6])
CNOT | (U[7], U[6])          #for z11

CNOT | (U[1], U[0])
CNOT | (U[3], U[0])          #for z13

CNOT | (U[0], U[3])
CNOT | (U[2], U[3])
CNOT | (U[6], U[3])          #for z14


Toffoli | (T[0], U[3], S[2])   #z14
CNOT | (S[2], S[5])

CNOT | (U[0], U[3])
Toffoli | (T[12], U[3], S[6])   #z12
```

```
CNOT | (S[6], S[2])
CNOT | (S[6], S[5])
CNOT | (U[0], U[3])

Toffoli | (T[1], U[4], S[1])      #z1
CNOT | (S[1], S[3])
CNOT | (S[1], S[4])

CNOT | (U[7], U[4])
Toffoli | (T[13], U[4], S[7])     #z0
CNOT | (S[7], S[1])
CNOT | (S[7], S[2])
CNOT | (S[7], S[3])
CNOT | (S[7], S[5])
CNOT | (U[7], U[4])

Toffoli | (T[3], U[0], S[6])      #z13
CNOT | (S[6], S[7])

CNOT | (U[3], U[6])
Toffoli | (T[11], U[6], S[0])     #z15
CNOT | (S[0], S[2])
CNOT | (U[3], U[6])

Toffoli | (T[14], U[2], S[0])     #z16
CNOT | (S[0], S[1])
CNOT | (S[0], S[3])
CNOT | (S[0], S[4])
CNOT | (S[0], S[5])
CNOT | (S[0], S[6])
CNOT | (S[0], S[7])

Toffoli | (T[9], U[7], Z[0])      #z2
CNOT | (Z[0], S[2])
CNOT | (Z[0], S[4])
CNOT | (Z[0], S[5])
CNOT | (Z[0], S[7])
Toffoli | (T[9], U[7], Z[0])

with Compute(eng):
   CNOT | (U[0], U[5])
   CNOT | (U[3], U[5])
   Toffoli | (T[12], U[5], Z[0]) #z3
CNOT | (Z[0], S[0])
CNOT | (Z[0], S[3])
CNOT | (Z[0], S[5])
CNOT | (Z[0], S[7])
Uncompute(eng)

with Compute(eng):
   CNOT | (U[1], U[6])
   CNOT | (U[2], U[6])
   CNOT | (U[3], U[6])
   CNOT | (U[4], U[6])
   Toffoli | (T[3], U[6], Z[0])  #z4
CNOT | (Z[0], S[0])
CNOT | (Z[0], S[3])
CNOT | (Z[0], S[4])
CNOT | (Z[0], S[5])
CNOT | (Z[0], S[6])
Uncompute(eng)

with Compute(eng):
   CNOT | (U[0], U[6])
   CNOT | (U[1], U[6])
   CNOT | (U[2], U[6])
   CNOT | (U[4], U[6])
   CNOT | (U[5], U[6])
   Toffoli | (T[0], U[6], Z[0])  #z5
CNOT | (Z[0], S[4])
CNOT | (Z[0], S[6])
CNOT | (Z[0], S[7])
Uncompute(eng)

with Compute(eng):
   CNOT | (U[0], U[7])
   CNOT | (U[1], U[7])
   CNOT | (U[2], U[7])
   CNOT | (U[4], U[7])
   CNOT | (U[5], U[7])
   CNOT | (U[6], U[7])
   Toffoli | (T[11], U[7], Z[0]) #z6
CNOT | (Z[0], S[0])
CNOT | (Z[0], S[1])
CNOT | (Z[0], S[2])
Uncompute(eng)

with Compute(eng):
   CNOT | (U[0], U[7])
   CNOT | (U[3], U[7])
   CNOT | (U[4], U[7])
   CNOT | (U[5], U[7])
   Toffoli | (T[14], U[7], Z[0]) #z7
CNOT | (Z[0], S[0])
CNOT | (Z[0], S[1])
CNOT | (Z[0], S[5])
CNOT | (Z[0], S[6])
Uncompute(eng)

with Compute(eng):
   CNOT | (U[1], U[6])
   CNOT | (U[2], U[6])
   CNOT | (U[3], U[6])
   Toffoli | (T[8], U[6], Z[0])  #z8
CNOT | (Z[0], S[2])
CNOT | (Z[0], S[5])
CNOT | (Z[0], S[6])
Uncompute(eng)

with Compute(eng):
   CNOT | (U[0], U[3])
```

```
    CNOT | (U[2], U[3])
    Toffoli | (T[13], U[3], Z[0]) #z9
CNOT | (Z[0], S[0])
CNOT | (Z[0], S[1])
CNOT | (Z[0], S[3])
CNOT | (Z[0], S[4])
Uncompute(eng)

with Compute(eng):
    CNOT | (U[0], U[6])
    CNOT | (U[2], U[6])
    CNOT | (U[3], U[6])
    Toffoli | (T[1], U[6], Z[0])  #z10
CNOT | (Z[0], S[0])
CNOT | (Z[0], S[1])
CNOT | (Z[0], S[3])
CNOT | (Z[0], S[4])
CNOT | (Z[0], S[5])
Uncompute(eng)

CNOT | (U[2], U[6])
CNOT | (U[3], U[6])
Toffoli | (T[8], U[6], S[2])      #z17
CNOT | (U[3], U[6])
CNOT | (U[2], U[6])

Toffoli | (T[9], U[6], S[5])      #z11

X | S[1]
X | S[2]
X | S[6]
X | S[7]

Uncompute(eng)
```

REFERENCES

[1] M. Almazrooie, A. Samsudin, R. Abdullah, and K. N. Mutter, "Quantum reversible circuit of AES-128," *Quantum Inf. Process.*, vol. 17, no. 5, 2018, Art. no. 112, doi: 10.1007/s11128-018-1864-3.

[2] B. Amento-Adelmann, M. Grassl, B. Langenberg, Y.-K. Liu, E. Schoute, and R. Steinwandt, "Quantum cryptanalysis of block ciphers: A case study," Poster at Quantum Information Processing, 2018.

[3] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 32, no. 6, pp. 818–830, Jun. 2013, doi: 10.1109/TCAD.2013.2244643.

[4] G. Banegas, Personal communication, Aug. 2019.

[5] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Develop.*, vol. 17, no. 6, pp. 525–532, 1973, doi: 10.1147/rd.176.0525.

[6] W. Bosma, J. Cannon, and C. Playoust, "The Magma algebra system. I. The user language," *J. Symbolic Comput.*, vol. 24, no. 3/4, pp. 235–265, 1997.

[7] J. Boyar and R. Peralta, "A new combinational logic minimization technique with applications to cryptology," in *Proc. Int. Symp. on Exp. Algorithms*, P. Festa, Ed. Berlin, Germany: Springer-Verlag, 2010, pp. 178–189, doi: 10.1007/978-3-642-13193-6_16.

[8] J. Boyar and R. Peralta, "A depth-16 circuit for the AES S-box," Cryptology ePrint Archive: Report 2011/332, Jun. 2011. [Online]. Available: https://eprint.iacr.org/2011/332

[9] M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt, "Applying Grover's algorithm to AES: Quantum resource estimates," in *Proc. Post-Quantum Cryptography*, T. Takagi, Ed. Berlin, Germany: Springer-Verlag, 2016, pp. 29–43, doi: 10.1007/978-3-319-29360-8_3.

[10] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annu. ACM Symp. Theory Comput.*, 1996, pp. 212–219, doi: 10.1145/237814.237866.

[11] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, "A software methodology for compiling quantum programs," *Quantum Sci. Technol*, vol. 3, 2018, doi: 10.1088/2058-9565/aaa5cc.

[12] S. Jaques, M. Naehrig, M. Roetteler, and F. Virdia, "Implementing Grover oracles for quantum key search on AES and LowMC," Cryptology ePrint Archive: Report 2019/1146, 2019. [Online]. Available: https://eprint.iacr.org/2019/1146

[13] P. Kim, D. Han, and K. C. Jeong, "Time–space complexity of quantum search algorithms in symmetric cryptanalysis: Applying to AES and SHA-2," *Quantum Inf. Process.*, vol. 17, 2018, Art. no. 339, doi: 10.1007/s11128-018-2107-3.

[14] NIST, *Advanced Encryption Standard*, Federal Information Processing Standards Publication 197, Nov. 2001.

[15] NIST, *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*, 2017. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

[16] P. Saravanan and P. Kalpana, "Novel reversible design of advanced encryption standard cryptographic algorithm for wireless sensor networks," *Wireless Personal Commun.*, vol. 100, no. 4, pp. 1427–1458, 2018, doi: 10.1007/s11277-018-5647-z.

[17] P. Selinger, "Quantum circuits of T-depth one," *Phys. Rev. A*, vol. 87, no. 4, 2013, Art. no. 042302, doi: 10.1103/PhysRevA.87.042302.

[18] D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: An open source software framework for quantum computing," 2016, *arXiv:1612.08091*.

[19] N. Wiebe and M. Roetteler, "Quantum arithmetic and numerical analysis using repeat-until-success circuits," *Quantum Inf. Comput.*, vol. 16, no. 1/2, pp. 0134–0178, 2016, doi: 10.26421/QIC16.1-2.

BRANDON LANGENBERG received the M.S. and Ph.D. degrees in mathematics from Florida Atlantic University, Boca Raton, FL, USA, in 2014 and 2018, respectively, with a focus on cryptography and quantum cryptanalysis.

He is currently a Senior Researcher with PQSecure Technologies, Boca Raton, FL, USA, and a Postdoctoral Researcher with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University. His research has been funded through the Air Force Research Laboratory and the German Federal Office for Information Security and is also the Principal Investigator on an NSF SBIR Phase II grant awarded to PQSecure for the commercialization of postquantum cryptography. His research interests include cryptography, quantum cryptanalysis, and postquantum cryptography.

HAI PHAM received the Graduate degree from Valencia College West Campus, Orlando, FL, USA, and the M.S. and Ph.D. degrees in mathematics with a concentration in cryptology from Florida Atlantic University, Boca Raton, FL, USA, in 2015 and 2019, respectively.

He is currently teaching with Valencia College West Campus. His research interests include cryptography, hybrid quantum protocol, and quantum cryptanalysis.

RAINER STEINWANDT received the M.S. and Ph.D. degrees in computer science, researching topics in computer algebra, from the University of Karlsruhe, Karlsruhe, Germany, in 1998 and 2000.

He is currently the Chair of the Department of Mathematical Sciences, Florida Atlantic University (FAU), Boca Raton, FL, USA. He is also the Director of the Center for Cryptology and Information Security, FAU. Before joining FAU, he was with the University of Karlsruhe. His research has been funded through the Air Force Research Laboratory, the German Federal Office for Information Security, the National Institute of Standards and Technology, the National Science Foundation, and the NATO Science for Peace and Security program. His research interest focuses on cryptology, including quantum cryptanalysis and quantum-safe cryptography.