Improved Quantum Analysis of SPECK and LowMC
(Full Version)

Kyungbae Jang[1], Anubhab Baksi[2], 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, khj1594012@gmail.com, hwajeong84@gmail.com, anupam@ntu.edu.sg

Abstract. As the prevalence of quantum computing is growing in leaps and bounds over the past few years, there is an ever-growing need to analyze the symmetric-key ciphers against the upcoming threat. Indeed, we have seen a number of research works dedicated to this. Our work delves into this aspect of block ciphers, with respect to the SPECK family and LowMC family.

The SPECK family received two quantum analysis till date (Jang et al., Applied Sciences, 2020; Anand et al., Indocrypt, 2020). We revisit these two works, and present improved benchmarks SPECK (all 10 variants). Our implementations incur lower full depth compared to the previous works.

On the other hand, the quantum circuit of LowMC was explored earlier in Jaques et al.'s Eurocrypt 2020 paper. However, there is an already known bug in their paper, which we patch. On top of that, we present two versions of LowMC (on L1, L3 and L5 variants) in quantum, both of which incur significantly less full depth than the bug-fixed implementation.

Keywords: Quantum Implementation · Grover's Search · SPECK · LowMC

1 Introduction

Among the major progress in the computational science in recent times, the quantum computing is certainly included in the topmost contenders. While a massive race of research is underway to build a functional quantum computer, it stands to reason that we should investigate how such a device can undermine the current security notions. As a matter of fact, it is well-known that certain public-key systems would face major problem [37,10,33,19,20,16] against an adversary equipped with a quantum computer. Going further, one may also notice that the symmetric-key counterpart would also be affected, mostly due to the so-called Grover's search algorithm [18].

Due to the power of the quantum properties of matter (namely, superposition and entanglement), quantum algorithms can find (with a high probability) the solution to certain types of problems faster than the best-known classical algorithms. In this case, the Grover's search algorithm can find the secret key of a symmetric-key cipher with about square-root search of what would be required for a classical computer, roughly speaking.

Therefore, it is not surprising that the research community in the symmetric-key cryptography as well would take interest in figuring out the possible impact a functional quantum computer can have — see Section 2.2 for a collection of related works. This work, too, makes a humble attempt to evaluate the quantum security of the block cipher families, SPECK [12] and LowMC [1].

Contribution

In brief, we present the followings in this work:

Hyunji Kim and Hwajeong Seo were supported by the Institute for Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (⟨Q|Crypton⟩, number 2019-0-00033, Study on Quantum Security Evaluation of Cryptography based on Computational Quantum Complexity); and Kyungbae Jang was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2022R1A6A3A13062701) of the Korean government. Anupam Chattopadhyay was partly supported by the NRF Grant Award, number NRF2021-QEP2-02-P05 by the Singaporean government. Further, we thank Da Lin (Hubei University, Wuhan, PR China) for the kind support during preparation of the manuscript. This is the full version of the paper with the same title accepted at Indocrypt 2022.

1. **SPECK Family** (10 variants; Section 4). We improve the quantum implementations of the SPECK family from the Indocrypt 2020 paper by Anand, Maitra and Mukhopadhyay, [6]; and the same from the Applied Sciences paper by Jang, Choi, Kwon, Kim, Park and Seo [23] in terms lower depth (though the X gate count is higher in our case). By improving the quantum adders and parallelization in the architecture, we show noticeable reduction of depth[3].
2. **LowMC Family** (L1, L3 and L5; Section 5). We observe that the implementation (LowMC) by [30] contains some programming related issue, which probably resulted in underestimating the resources for non-linear components (similar issue with respect to AES was reported by the Asiacrypt'20 authors [42]; and later in [22,25]); although the linear components (Sections 5.2, 5.3) were 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 they reported in Section 5.5.
   Independent to that, we present two versions of three LowMC variants, which we refer to as, the *regular* (⌑) and the *shallow* (⋈) versions. Both the regular and the shallow versions provide high parallelism as the linear layer and key schedule work simultaneously. The regular (respectively, shallow) version uses the S-box implementation that has the Toffoli depth of 3 (respectively, 1), as described in Figure 5. Further, we show some improvement in the implementation of the linear layer, key schedule, and also in the parallelization of both.

Table 1 shows the benchmarks for the SPECK cipher family, including the results from [6,23]. The proposed SPECK quantum circuits require a higher number of X gates than previous works. This is due to the nature of the quantum adder used in our implementation (detailed in Section 4.1). Similarly, a summary of results of on LowMC can be found in Table 2, where we consolidate results from the bug-fixed implementation of [30]. The T-depth of the shallow version of LowMC is higher than the bug-fixed implementation of [30], but actually, this is derived from the difference in the decomposition method of the Toffoli gate although the Toffoli depth is the same (see Section 2.3 for details).

When the basic implementation of the ciphers is available, in Section 6, we elaborate the estimated cost of running the Grover's search algorithm. We estimate only the cost of oracle in the Grover's search algorithm with the proposed quantum circuits. There is a module called diffusion operator that amplifies the amplitude of the solution returned by oracle, but the overhead is negligible, so it is excluded from the cost estimation. Lastly, the parallel operation of the Grover's search algorithm required according to the block and key size of the cipher is reflected in the cost estimation. We also comment on the quantum security level proposed in [32].

Our source codes are written in ProjectQ[4]. Developed by the researchers from ETH Zurich, it is a Python-based open-source framework for quantum computing, and offers a support for IBM's quantum chips. The variable `resource_check` is set to 0 in `ClassicalSimulator` to check the test vectors and set to 1 in `ResourceCounter` to decompose Toffoli gates in our codes. All relevant codes, along with a toy version of SPECK (where it is possible to simulate the Grover's search), are released in public[5].

2 Prerequisite

2.1 Backdrop and Motivation

The Grover's search algorithm is a quantum algorithm that can find a solution in an n-qubit search space with $\lfloor \frac{\pi}{4}\sqrt{2^n} \rfloor$ (about $\sqrt{2^n}$) serial application. Theoretically, this algorithm can reduce symmetric-key ciphers (having an n-bit key) with n-bit security on a classical computer to $n/2$-bit security on a quantum computer.

An abridged description of the algorithm is given as follows. The Grover's search algorithm operates on n-qubits in the superposition state and finds a solution by iterating the set of oracle and diffusion operators about n times. First, n Hadamard gates are used to prepare n-qubits in superposition state. This causes 2^n queries to coexist as probabilities in n-qubit. In the oracle, the logic to find a solution is implemented as quantum gates. For the quantum key search, quantum encryption of the target cipher must be implemented as logic in the oracle. The oracle finds a solution (i.e., the secret key), but the measurement probabilities with non-solutions are still the same. So, the diffusion operator amplifies the amplitude of the solution returned

[3] However the reduction of full depth is less prominent (ranging from 10 percent to 12 percent depending on the variant of SPECK), still our implementation takes less quantum resource. See Table 4 for the benchmark.

[4] Homepage: https://projectq.ch/. Code: https://github.com/ProjectQ-Framework/ProjectQ. Documentation: https://projectq.readthedocs.io/en/latest/.

[5] https://github.com/starj1023/SPECK_LowMC_QC.

Table 1: Comparison of quantum resources required for variants of SPECK.

SPECK		#Toffoli ☆	#CNOT ✻	#NOT ✱	#qubits ❂	Depth	Full depth ❉
32/64	JCKKPS [23]	1,290	3,706	42	97	3,313	N/A
	AMM [6]	1,290	4,222	42	96	1,694	5,873
	This work	1,247	4,179	1,160	98	814	5,258
48/72	JCKKPS [23]	1,982	5,606	42	121	4,969	N/A
	AMM [6]	1,978	6,462	42	120	2,574	9,153
	This work	1,935	6,419	1,848	122	1,166	8,075
48/96	JCKKPS [23]	2,074	5,866	45	145	5,203	N/A
	AMM [6]	2,070	6,762	45	144	2,691	9,541
	This work	2,025	6,717	1,935	146	1,219	8,441
64/96	JCKKPS [23]	3,162	8,890	54	161	8,009	N/A
	AMM [6]	3,162	10,318	54	160	4,082	14,563
	This work	3,111	10,267	3,012	162	1,794	12,870
64/128	JCKKPS [23]	3,286	9,238	57	193	8,323	N/A
	AMM [6]	3,286	10,722	57	192	4,239	15,181
	This work	3,233	10,669	3,131	194	1,863	13,365
96/96	JCKKPS [23]	5,172	14,436	60	193	12,923	N/A
	AMM [6]	5,170	16,854	60	192	6,636	23,657
	This work	5,115	16,799	5,010	194	2,828	21,028
96/144	JCKKPS [23]	5,360	14,960	64	241	13,397	N/A
	AMM [6]	5,358	17,466	64	240	6,873	23,657
	This work	5,301	17,409	5,194	242	2,929	21,779
128/128	JCKKPS [23]	7,942	22,086	75	257	19,797	N/A
	AMM [6]	7,938	25,862	75	256	10,144	36,358
	This work	7,875	25,799	7,761	256	4,256	32,224
128/192	JCKKPS [23]	8,192	22,784	80	321	20,427	N/A
	AMM [6]	8,190	26,682	80	320	10,461	37,381
	This work	8,125	26,617	8,010	322	4,389	33,231
128/256	JCKKPS [23]	8,444	23,484	81	385	21,061	N/A
	AMM [6]	8,442	27,502	81	384	10,778	38,431
	This work	8,375	27,435	8,255	386	4,522	34,238

by the oracle. After increasing the amplitude of the solution sufficiently by repeating the oracle and diffusion operators, n-qubits are finally measured.

However, the catch is that the quantum attack using the Grover's algorithm on the symmetric-key cipher requires a lot of quantum resources. Despite much progress though, the state-of-the-art quantum computers have only very limited resources, and consequently cannot afford to run the Grover's algorithm.

If the quantum cost required to attack the cipher is high, it can be expected to provide the desired security (i.e., n-bit security) even in the post-quantum era (without increasing the key size). Thus, it is important to estimate and analyze the cost of quantum attacks on various ciphers.

2.2 Related Works

Estimating the quantum resources required for key recovery using the Grover search algorithm was probably first presented for AES by Grassl, Langenberg, Roetteler, and Steinwand [17]. This work has been followed up by the research community with various implementations of AES [30,42,22,25,2,31]. These papers all focus on the efficient implementation of quantum circuits, thereby reducing the cost for running the Grover's search algorithm with increasingly low resource. Apart from AES, a large number of other ciphers have also received the quantum analysis, SIMON [7], SPECK [6,23], SKINNY [13], PRESENT and GIFT [27], SHA-2 and SHA-3 [3], FSR-based ciphers [5], ChaCha [11], SM3 [38,41], RECTANGLE and KNOT [9], KATAN [34], DEFAULT [24], GIFT–SKINNY–SATURNIN [13], PIPO [29], to name some of those.

2.3 Quantum Gates

There are several commonly used quantum gates to implement ciphers into quantum circuits, such as X (NOT), CNOT, and Toffoli (CCNOT) gates. The X gate inverts the value of a qubit, which can replace the classical

Table 2: Comparison of quantum resources required for variants of LowMC.

LowMC		#CNOT	#1qCliff	#T	T-depth	#qubits	Full depth
L1	❄	344,972	2,466	4,200	20	1,006	49,350
	⌑	498,208	2,466	4,200	240	3,200	4,708
	⋈	500,208	2,466	4,200	80	3,830	4,708
L3	❄	1,135,935	4,699	6,300	30	1,434	159,659
	⌑	1,669,456	4,699	6,300	360	6,720	10,571
	⋈	1,672,456	4,699	6,300	120	7,650	10,571
L5	❄	2,535,162	7,137	7,980	38	1,802	346,736
	⌑	3,754,484	7,137	7,980	456	11,008	17,789
	⋈	3,758,284	7,137	7,980	152	12,178	17,789

❄: Bug-fixed JNRV [30]
⌑: Regular version.
⋈: Shallow version.

NOT operation (i.e., X $(a) = \sim a$). The CNOT gate operates on two qubits, and the value of the target qubit is determined according to the value of the control qubit. If the value of the control qubit is 1, the target qubit is inverted, and if the value of the control qubit is 0, it is maintained (i.e., CNOT $(a, b) = (a, a \oplus b)$. Since this is equivalent to XORing the value of the control qubit to the target qubit, the CNOT gate can replace the classic XOR operation. Toffoli gates operate on three qubits, with two control qubits and one target qubit. The value of the target qubit is reversed only when the values of both control qubits are 1 (i.e., Toffoli $(a, b, c) = (a, b, c \oplus ab)$). Since this is equivalent to XORing the ANDed value of control qubits to the target qubit, Toffoli gate can replace the classic AND operation. We can implement cipher encryption in quantum using these quantum gates, which can replace the classic NOT, XOR, and AND operations.

Among these gates, it is important from an optimization point of view that we need to reduce the number of Toffoli gates. Because the Toffoli gate is implemented as a combination of T gates (determine the T-depth) and Clifford gates (i.e., CNOT, H, or X gate), the cost is relatively high. There are several ways to decompose the Toffoli gate [4,35,21], and the full depth means the depth when the Toffoli gates are decomposed. In our work, when estimating decomposed resources, we adopt the decomposition method of 7 T gates + 8 Clifford gates, T-depth of 4, and full depth of 8 for one Toffoli gate [4].

2.4 NIST Security Levels

In order to describe the security of cipher against a quantum adversary, NIST stated the following security margins for a cipher [32]:

- Level 1: Cipher is at least as hard to break as AES-128.
- Level 2: Cipher is at least as hard to break as SHA-256.
- Level 3: Cipher is at least as hard to break as AES-192.
- Level 4: Cipher is at least as hard to break as SHA-384.
- Level 5: Cipher is at least as hard to break as AES-256.

NIST recommended that ciphers should achieve at least Levels 1, 2 and/or 3, to provide sufficient security in the post-quantum era. The estimates used in [32] were based on the results of AES circuits were taken from that of [17], and are as listed as follows: Level 1: 2^{170}, Level 3: 2^{233}, Level 5: 2^{298}. These figures were calculated as total number of gates × full depth of the quantum key search (as estimated in [17]) respectively for AES-128, 192, and 256 under the Grover's algorithm.

3 Target Ciphers

3.1 SPECK Family (32/64, 48/72, 48/96, 64/96, 64/128, 96/96, 96/144, 128/128, 128/192, 128/256)

SPECK [12] is a family of lightweight block ciphers that was developed by the National Security Agency (NSA) in 2013. The SPECK family adopts a Feistel-like structure and contains 10 variants. The parameters for each variant are specified in Table 3.

Table 3: Parameters for SPECK variants.

Word size (n)	Key words (m)	Block size ($2n$)	Key size (nm)	α	β	Rounds (T)
16	4	32	64	7	2	22
24	3	48	72	8	3	22
	4		96			23
32	3	64	96	8	3	26
	4		128			27
48	2	96	96	8	3	28
	3		144			29
64	2	128	128	8	3	32
	3		192			33
	4		256			34

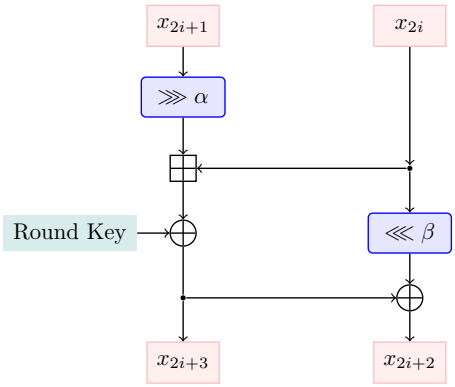


Fig. 1: Round function of SPECK.

Round Function The round function of SPECK consists of modular addition, bit-wise rotation and exclusive-OR (XOR) as shown in Figure 1. Let (x_{2i+1}, x_{2i}) be the $2n$-bit input of the ith round, where x_{2i+1} and x_{2i} are both n-bit words. In each round, the state is updated as follows:

1. Updating x_{2i+1} by cyclically shifting its bits to the right by α bits, and then performing the addition modulo 2^n on x_{2i+1} and x_{2i} via $x_{2i+1} = x_{2i+1} + x_{2i}$.
2. XORing the n-bit round key to x_{2i+1}, and cyclically shifting the bits in x_{2i} to the left by β bits, simultaneously.
3. XORing x_{2i+1} to x_{2i} and finishing the update of round function.

Key Schedule The sub-keys of SPECK are expanded in a similar way as the state in each round. Let us denote by $l_0, l_1, \cdots, l_{m-2}$ the variables for producing the sub-keys of SPECK family. In order to generate the $(i+1)^{\text{th}}$-round sub-key, where $i \in \{0, T\}$, take (l_i, k_i) as the input of round function as shown in Figure 1 with the number i served as round key in key addition step. Denote the output (l_{i+m-1}, k_{i+1}), k_{i+1} is the generated sub-key.

3.2 LowMC Family (L1, L3, L5)

LowMC [1] is a family of SPN-based block ciphers. Motivated by the fact that non-linear gates are costly compared to the linear gates in applications such as Multi-party Computation (MPC), Fully Homomorphic Encryption (FHE) and Zero Knowledge (ZK), the ciphers specific to these niches are designed to have a low small AND gate/depth count. LowMC is flexible in design (some components of it can be determined randomly), the recommended instance of [1] can be characterized by the block size n, the key size k, the number of S-boxes m in the non-linear layer, the allowed data complexity d of attacks and the round r, where $(n, k, m, d, r) \in \{(256, 80, 49, 64, 11), (256, 128, 63, 128, 12)\}$. Note that in the post-quantum digital signature Picnic[6] [40], the adopted variants of LowMC can be characterized by $(n, k, m, r) \in \{(128, 128, 10, 20), (192, 192, 10, 30), (256, 256, 10, 38)\}$. LowMC round consists of SboxLayer, LinearLayer, ConstantAddition and KeyAddition; and in the Key Schedule, round keys are generated through LinearLayer[7].

Round Function The encryption of LowMC starts with a whitening key addition over $\mathbb{F}_2$, followed by r iterations of the round function which is composed as KeyAddition $\circ$ ConstantAddition $\circ$ LinearLayer $\circ$ SboxLayer. Schematic diagrams of LowMC round function and key schedule can be found in Figure 2.

[6] Apart from LowMC, Picnic also uses SHA-3 in some form.

[7] As the exact specification is generated at random, it is suggested in [8] to call LowMC as a 'meta-cipher' (instead of a 'cipher').

SboxLayer. LowMC adopts a 3-bit S-box (in the look-up form, it is given by `01367452`) with the coordinate function representation (in ANF) as $(a \oplus bc, a \oplus b \oplus ac, a \oplus b \oplus c \oplus ab)$ in its substitution layer, where a, b, c are the input bits. For a specific instance of LowMC, only the first $3m$ bits of the state will go through the S-box.

LinearLayer. The linear layer of LowMC is matrix multiplication in $\mathbb{F}_2$.

ConstantAddition. Round constants are XORed to the sate by the operation of addition in $\mathbb{F}_2$.

KeyAddition. The n-bit round keys generated by key schedule are XORed to the state after each round. Also, the encryption with LowMC starts with a key whitening.

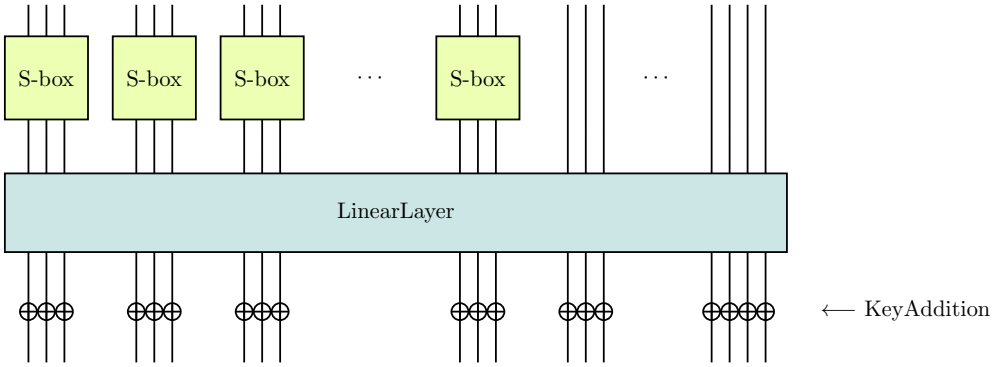


Fig. 2: Round function of LowMC.

Key Schedule LowMC uses a simple method to generate the sub-key for each round. The round keys are derived from the master key via multiplication with a random matrix with full rank.

4 SPECK in Quantum

For implementation of SPECK in quantum, we present a parallel addition implementation for a quantum circuit. We design a parallel addition structure by allocating one more carry qubit. We take an on-the-fly approach to perform round functions and key schedules together. Then the additions of the round function and key schedule are performed in parallel. As a result, compared to the implementation in [6], we save one Toffoli gate per addition and provide a 56% performance improvement in terms of depth.

4.1 Quantum Adder for SPECK

A quantum adder is implemented as a combination of quantum gates. Previous implementations of SPECK [26] used a ripple carry-based quantum adder [15]. The quantum adder in the previous work uses one ancilla qubit, $(2n-2)$ Tofffoli gates, $(4n-2)$ CNOT gates, with a depth of $(5n-3)$. Later, Anand et al. improved performance in terms of depth and saved one ancilla qubit by adopting a different quantum adder (from [39]) in their SPECK quantum circuit implementation [6]. The quantum adder used in their work uses a $(2n-2)$ Tofffoli gate, $(5n-6)$ CNOT gates, with a depth of $(5n-5)$ where no ancilla qubits are used. For this reason, it saves 1 qubit compared to the quantum adder used in [26].

We use an improved quantum adder based on the ripple-carry approach, which is referred to as the improved Cuccaro–Draper–Kutin–Moulton (CDKM) adder [15]. This quantum adder uses one ancilla and more X gates, but reduce the Toffoli gates and circuit depth, significantly. When the condition is $n \geq 4$ for n-bit addition, an improved quantum adder is available. Since the 16-bit addition operator is the smallest unit in SPECK, the improved quantum adder can be applied to all variants of SPECK. In modular addition, one ancilla qubit can be saved (generic addition uses two ancilla), and the quantum gates and circuit depth can also be reduced.

Finally, the quantum adder we used requires one ancilla qubit, $(2n-3)$ Toffoli gates, $(5n-7)$ CNOT gates, $(2n-6)$ X gates, and the circuit depth is $(2n+3)$. We do not know exactly why, but when we implemented in ProjectQ, a depth of $(2n+3)$ was estimated. Details of the implementation can be found in [15]. Algorithm 1 describes the improved CDKM adder used in our implementation of SPECK.

Algorithm 1: Quantum circuit for improved n-bit CDKM adder ($n \geq 6$).

Input: n-qubit operands a, b, carry qubit c $(= 0)$.
Output: $a = a, b = a + b, c = 0$.

for $i = 0$ to $n-3$ **do**
    $b[i+1] \leftarrow \text{CNOT}(a[i+1], b[i+1])$
end for

$c \leftarrow \text{CNOT}(a[1], c)$
$c \leftarrow \text{Toffoli}(a[0], b[0], c)$
$a[1] \leftarrow \text{CNOT}(a[2], a[1])$
$a[1] \leftarrow \text{Toffoli}(c, b[1], a[1])$
$a[2] \leftarrow \text{CNOT}(a[3], a[2])$

for $i = 0$ to $n-6$ **do**
    $a[i+2] \leftarrow \text{Toffoli}(a[i+1], b[i+2], a[i+2])$
    $a[i+3] \leftarrow \text{CNOT}(a[i+4], a[i+3])$
end for

$a[n-3] \leftarrow \text{Toffoli}(a[n-4], b[n-3], a[n-3])$
$b[n-1] \leftarrow \text{CNOT}(a[n-2], b[n-1])$
$b[n-1] \leftarrow \text{CNOT}(a[n-1], b[n-1])$
$b[n-1] \leftarrow \text{Toffoli}(a[n-3], b[n-2], b[n-1])$

for $i = 0$ to $n-4$ **do**
    $b[i+1] \leftarrow \text{X}(b[i+1])$
end for

$b[1] \leftarrow \text{CNOT}(c, b[1])$

for $i = 0$ to $n-4$ **do**
    $b[i+2] \leftarrow \text{CNOT}(a[i+1], b[i+2])$
end for

$a[n-3] \leftarrow \text{Toffoli}(a[n-4], b[n-3], a[n-3])$

for $i = 0$ to $n-6$ **do**
    $a[n-4-i] \leftarrow \text{Toffoli}(a[n-5-i], b[n-4-i], a[n-4-i])$
    $a[n-3-i] \leftarrow \text{CNOT}(a[n-2-i], a[n-3-i])$
    $b[n-3-i] \leftarrow \text{X}(b[n-3-i])$
end for

$a[1] \leftarrow \text{Toffoli}(c, b[1], a[1])$
$a[2] \leftarrow \text{CNOT}(a[3], a[2])$
$b[2] \leftarrow \text{X}(b[2])$
$c \leftarrow \text{Toffoli}(a[0], b[0], c)$
$a[1] \leftarrow \text{CNOT}(a[2], a[1])$
$b[1] \leftarrow \text{X}(b[1])$
$c \leftarrow \text{CNOT}(a[1], c)$

for $i = 0$ to $n-2$ **do**
    $b[i] \leftarrow \text{CNOT}(a[i], b[i])$
end for

return a, b, c

4.2 Quantum Circuit for SPECK using Parallel Addition

We briefly reiterate the round function of SPECK and the key schedule process (refer to Section 3.1) for better clarity. The round function of SPECK uses an n-bit round key (k) for a $2n$-bit (x, y) block, and the process is shown in Equation (1). Notations $\lll$ and $\ggg$ mean left and right rotation, respectively.

$$R_k(x, y) = ((x \ggg \alpha) + y) \oplus k, (y \lll \beta) \oplus ((x \ggg \alpha) + y) \oplus k) \tag{1}$$

The initial key is $K = k_0, l_0, \ldots, l_{m-2}$, and the generated $RK_i = k_0, k_1, \ldots, k_{r-1}$ are used as the i^{th} round key ($0 \leq i \leq r-1$, r being the total number of rounds). The key schedule process is given in Equation (2).

$$l_{i+m-1} = (k_i + (l_i \ggg \alpha)) \oplus i, k_{i+1} = (k_i \lll \beta) \oplus l_{i+m-1}. \tag{2}$$

In this part, we explore where the parallel addition is available in the implementation of SPECK as a quantum circuit. We use the initial k_0 in the first round, then update k_0 to k_i to use it as the round key in the i^{th} round ($0 \leq i \leq r-1$). By taking this on-the-fly approach, there is no need to allocate qubits for the key schedule. For each round, the round function and key schedule are executed together. Due to this, addition $(x \ggg \alpha) + y$ in the round function and addition $k_i + (l_i \ggg \alpha)$ in the key schedule can be performed in parallel. In the previous implementation, the key schedule is performed after the round function in the i-th round by adopting the same on-the-fly approach. And only one carry qubit c_0 for addition is allocated. We take two different approaches for parallel addition.

First, k should not be updated in the key schedule until the round key k is used in the round function. In general, parallel addition is impossible because the round function and the key schedule are performed sequentially. We present the procedure for each round as round function (1/2) → key schedule (1/2) → round function (2/2) → key schedule (2/2) instead of round function (1/1) → key schedule (1/1).

Then, Round function (1/2) is $x = (x \ggg \alpha) + y$ and key schedule (1/2) is $l_i = k_i + (l_i \ggg \alpha)$, which are parallel addition targets. Since k_i should not be updated (required in round function (2/2)), the result of addition in key schedule (1/2) is stored in l_i. Round function (2/2) is $x = x \oplus k_i$, $y = (y \lll \beta) \oplus x$ and key schedule (2/2) is $k_i = (k_i \lll \beta) \oplus (l_i \oplus i)$.

Algorithm 2: Quantum circuit implementation of SPECK-32/64.

Input: 32-qubit block (x, y), 64-qubit keywords (k_0, l_0, l_1, l_2),
carry qubits c_0 $(=0)$, c_1 $(=0)$.
Output: 32-qubit ciphertext (x, y).

for $i = 0$ to $r - 2$ **do**
  Round function (1/2) :
    $x \leftarrow x \ggg 7$
    $x \leftarrow \mathrm{ADD}(y, x, c_0)$
  Key schedule (1/2) :
    $l_{i\%3} \leftarrow l_{i\%3} \ggg 7$
    $l_{i\%3} \leftarrow \mathrm{ADD}(k_0, l_{i\%3}, c_1)$
  Round function (2/2) :
    $x \leftarrow \mathrm{CNOT16}(k_0, x)$
    $y \leftarrow y \lll 2$
    $y \leftarrow \mathrm{CNOT16}(x, y)$
  Key schedule (2/2) :
    for $j = 0$ to 5 **do** ▷ Constant XOR
      if $(i \gg j)\&1$ **then**
        $l_{i\%3}[j] \leftarrow \mathrm{X}(l_{i\%3}[j])$
      end if
    end for
    $k_0 \leftarrow k_0 \lll 2$
    $k_0 \leftarrow \mathrm{CNOT16}(l_{i\%3}, k_0)$
end for

Round function (1/2) : ▷ Last round
  $x \leftarrow x \ggg 7$
  $x \leftarrow \mathrm{ADD}(y, x, c_0)$
Round function (2/2) :
  $x \leftarrow \mathrm{CNOT16}(k_0, x)$
  $y \leftarrow y \lll 2$
  $y \leftarrow \mathrm{CNOT16}(x, y)$
return (x, y)

Second, now that parallel addition is possible, we need one more carry qubit for this. A ripple-carry quantum adder requires a carry qubit with an initial value of 0, and when the addition is completed, the carry qubit is reset to 0 again. The previous implementation takes advantage of this to allocate only one carry qubit c_0 and reuse it in all additions. However, in order to reuse c_0, the next addition cannot be performed until the addition is finished. Therefore, since we will perform two additions in parallel, we allocate two carry qubits c_0 and c_1 and use them in each addition.

Finally, the proposed quantum circuit implementation provides a 56% performance improvement in terms of depth. Algorithm 2 describes the quantum circuit implementation for SPECK-32/64.

This technique is applied to all SPECK versions, only the parameters are changed. Implementations for other versions can be found in our code. Rotations (i.e., $\lll, \ggg$) can be implemented with the swap gates, but we do not use quantum resources by implementing a logical swap that changes the index of the qubits. CNOT16 means CNOT gate operation of a 16-qubit array. Figure 3 shows the quantum circuit of SPECK-32/64 operating for 3 rounds.

4.3 Architecture and Resource Requirement

As shown in Table 1, the quantum resources required to implement our SPECK quantum circuits are much cheaper compared to the previous SPECK quantum circuits. In [26,23], Jang et al. used a ripple carry-based quantum adder and did not take into account the room for parallel addition. In [6], Anand et al. improved the performance by using a different quantum adder than that of the previous implementation. The quantum circuit they implemented does not use additional qubits and offers performance improvements in terms of depth. We use two more carry qubits and X gates, but parallel addition using an improved ripple carry quantum adder provides performance improvement in terms of circuit depth and a reduction in the number of Toffoli gates.

In the case of SPECK, which is based on an ARX structure, it is important which quantum adder is used. In this work, a quantum circuit is designed so that the additions of the round function of SPECK and the key

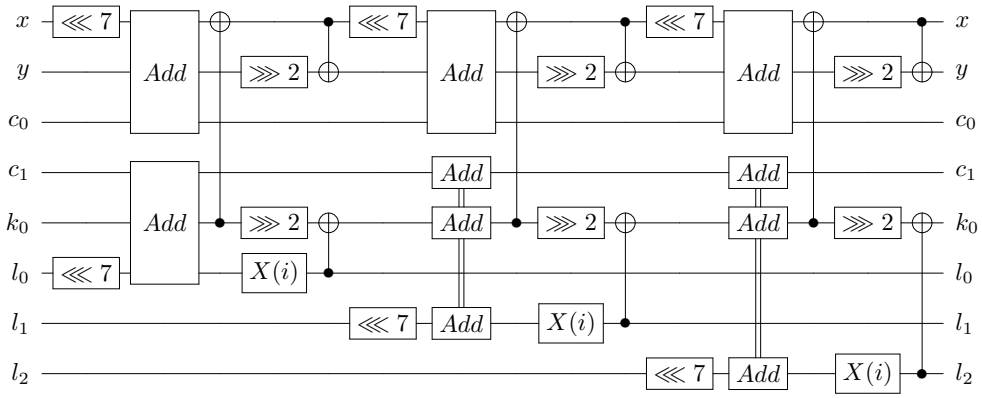


Fig. 3: Quantum circuit for SPECK-32/64 (3 rounds only).

schedule are performed in parallel, and a few ancilla qubits are allocated accordingly. Also, this approach is expandable because it works even if it is changed to another quantum adder.

In Table 1, quantum resources are reported when the Toffoli gates are not decomposed for simplicity of comparison. However, the Toffoli gate is decomposed into several quantum gates. For detailed resource estimation in this paper, we follow the Toffoli gate decomposition in [4]. One Toffoli gate is decomposed into 7 T gates + 8 Clifford gates (T-depth is 4 and full depth is 8). Table 4 shows the detailed quantum resources required for our SPECK quantum circuits. A birds-eye-view of the relative performance of the relevant works can be seen from Figure 4.

Table 4: Quantum resources (decomposed gates) required for variants of SPECK (this work).

SPECK	#CNOT ✻	#1qCliff ❁	#T ✢	T-depth ✤	#qubits ❂	Full depth ❋
32/64	11,661	3,654	8,729	2,552	98	5,258
48/72	18,029	5,718	13,545	3,960	122	8,074
48/96	18,867	5,985	14,175	4,140	146	8,441
64/96	28,933	9,234	21,777	6,344	162	12,870
64/128	30,067	9,597	22,631	6,588	194	13,365
96/96	47,489	15,240	35,805	10,416	194	21,028
96/144	49,215	15,796	37,107	10,788	242	21,779
128/128	73,049	23,511	55,125	16,000	258	32,224
128/192	75,367	24,260	56,875	16,500	322	33,231
128/256	77,685	25,005	58,625	17,000	386	34,238

5 LowMC in Quantum

Regular and Shallow Versions

As mentioned earlier, our quantum circuits of LowMC are divided into regular (⌑) and shallow (⋈) versions. The regular version offers high parallelism while taking into account the trade-off of qubit-depth. Both the regular and the shallow versions provide high parallelism as the linear layer and key schedule work simultaneously. The difference is that the regular version of the S-box has a Toffoli depth of 3 and the shallow version of the S-box has a Toffoli depth of 1, as detailed in Section 5.1.

5.1 Implementation of S-box

In [30], two quantum circuit implementation for the 3-bit S-box of LowMC were described as shown in Figure 5. The in-place S-box (Figure 5(a)) stores the output value in the input, and the shallow S-box (Figure 5(b))

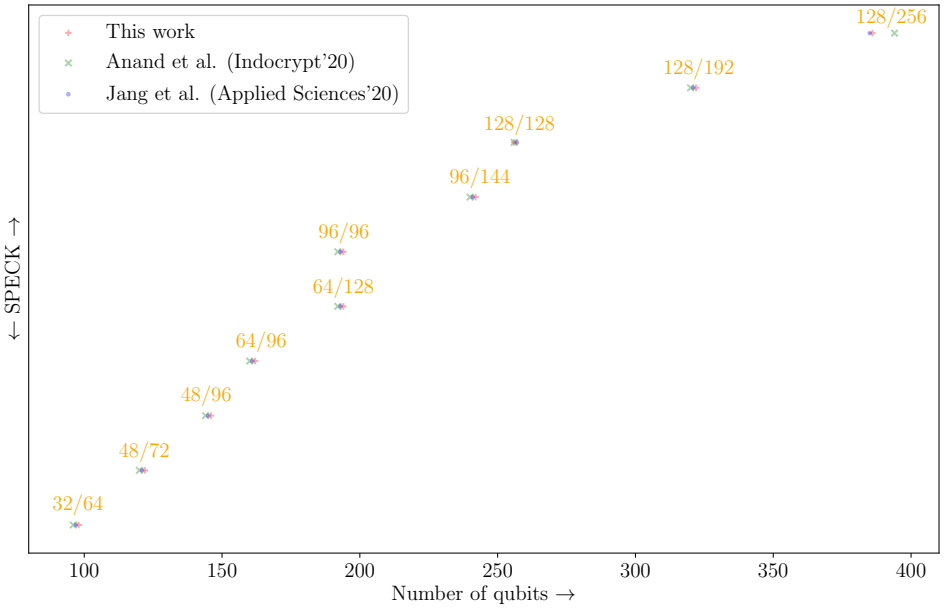
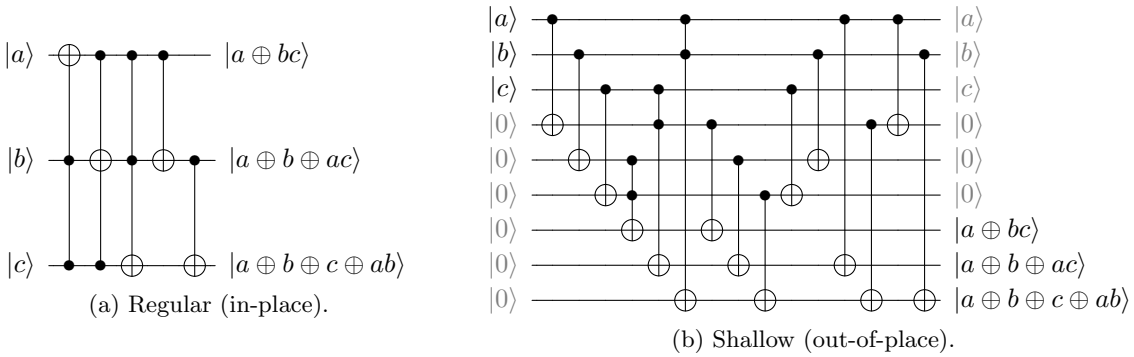


Fig. 4: Comparison of qubit requirement for SPECK variants.

additionally uses 3 output qubits and 3 ancilla qubits, but the Toffoli depth can be reduced and the shallow S-box is adopted in their implementation. Notice that the 3 garbage lines are reset, this is because those are reused in the next S-box (save for the last one). When the Toffoli gate is decomposed in the case of the in-place S-box, the full depth is 23, and the shallow S-box is lower at 12. Table 5(a) shows the quantum resources required for the two implementations of the 3-bit S-box.



(a) Regular (in-place).

(b) Shallow (out-of-place).

Fig. 5: Quantum circuit for LowMC S-box.

Several trade-offs are to be considered when choosing the quantum S-box implementation. The Toffoli depth of the in-place S-box is 3 and that of the shallow S-box is 1. This is definitely an advantage for the shallow S-box. However, we found that the full depth of the S-box does not affect the full depth of the LowMC when using 10 S-boxes. This is because the depth for S-box is covered by the key schedule and the linear layer. One thing to note is that in-place S-box can be operated in parallel without additional cost, but shallow S-box requires additional ancilla qubits for parallel operation, and qubits for output are allocated every round. Considering these trade-offs, we adopt and compare both S-boxes in our implementations. The regular version of LowMC (⌑) uses the regular/in-place S-box implementation and the shallow version (⋈) uses the shallow/out-of-place S-box implementation.

5.2 Implementation of Linear Layer and Key Schedule

In the linear layer, the pseudo-randomly generated matrix over GF(2) of dimension $n \times n$ in LowMC instantiation is multiplied by an n-bit block. In [30], it is possible to implement an in-place implementation in which CNOT gates are used only in an n-qubit block due to PLU factorization (i.e., internal mixing). In contrast, in our quantum circuit implementation, CNOT gate is performed depending on where the bit value of the matrix is 1. In the CNOT gate, the n-qubit block acts as a control, and a newly allocated n-qubit acts as a target. Finally, the matrix product is stored in the newly allocated n-qubit. Although n-qubit to store the output of the linear layer is newly allocated every round, our approach can obtain a compact quantum circuit. Because we allocate new n-qubits for matrix multiplication, it frees up space and allows for parallelism. Table 5(b) shows the quantum resources required to implement quantum circuits for the linear layer. Since the CNOT gates and depths required for a round are slightly different according to the pseudo-randomly generated matrices, our results in Table 5(b) show the average for all rounds.

In the key schedule, round keys are generated by multiplying the k-bit input key with the matrix of dimension $k \times k$ of each round in the same way. Unlike the linear layer, we can save qubits by using the reverse operation in the key schedule. Only in the first key schedule, a new k-qubit for storing the round key is allocated. After KeyAddition, the reverse operation of the key schedule is performed to return the round key (k-qubit) to a clean state, and it is reused in the next key schedule. Due to the reverse operation, the CNOT gates are doubled. However, in terms of depth, we perform the reverse operation of the key schedule in parallel with the linear layers for the n-qubit block by using two input keys and round key qubits. Figure 6 shows our LowMC quantum circuit operating fully in parallel by operating two input keys (reverse operation of Key Schedule is denoted as Key schedule†[8]). We initially allocate additional $2 \cdot k$ qubits (k_1 and rk_1) and use them alternately in rounds. Although it is omitted in Figure 6, the input key k_0 is copied to k_1 through the CNOT gates and then the circuit is executed. Through this, the key schedule and the reverse operation of the key schedule can be executed simultaneously with the linear layer. Table 5(c) shows the quantum resources required to implement quantum circuits for the key schedule. It should be pointed out that in Table 5(c), our result excludes the initially allocated $3 \cdot k$-qubit (rk_0, k_1 and rk_1 in Figure 6). The regular version of Figure 6(a) and the shallow version of Figure 6(b) differ in whether the output qubits for the S-boxes are allocated or not, and the Toffoli depth.

5.3 Implementation of KeyAddition and ConstantAddition

KeyAddition and ConstantAddition are implemented the same as in SPECK. KeyAddition is simply implemented using k CNOT gates. In ConstantAddition, since the constants are already known, the X gates are performed where the bit value of the constant is 1.

5.4 Architecture and Resource Requirement

As already presented in Table 2, one may find the quantum resources required to implement our LowMC quantum circuits. In LowMC quantum circuits, the most quantum resources are used for matrix multiplication in the key schedule and linear layer. In [30], an in-place implementation was presented through matrix multiplication using the PLU factorization. On the other hand, we design with a general structure, using more qubits, but more compact quantum circuits are obtained. Lastly, our quantum circuit design using two input keys simultaneously executes the linear layer, key schedule, and reverse operation of the key schedule.

5.5 Corrected LowMC implementation from Eurocrypt'20 (JNRV)

For a clearer context, here we give a brief description of the situations where Q#'s `ResourcesEstimator` issues arise and how those issues affect the quantum benchmarks given in the Eurocrypt'20 paper [30]. This was discovered when we tried to cross-check their publicly available source codes[9]. Indeed, this was also noted in [42] as a bug; and this apparently led to underestimation of gate count, qubit count and depth reported in [30] for the non-linear components (and also the S-box of LowMC).

[8] Key Schedule in quantum (of LowMC) denotes the product of the matrix of the round and the input key, and the product is stored in qubits for the round key. The reverse operation (i.e., uncompute) of Key Schedule is defined as Key Schedule†, and cleans the qubits for the round key.

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

Table 5: Comparison of quantum resources (decomposed gates) required for LowMC.

(a) S-box.

Method	#CNOT ✻	#1qCliff ❁	#T ✢	Toffoli depth ✦	#qubits ❂	Full depth ❋
⌑ S-box [30]	20	6	21	3	3	26
⋈ S-box [30]	30	6	21	1	9	12

⌑: Regular version.
⋈: Shallow version.

(b) Linear layer.

Method	#CNOT ✻	#1qCliff ❁	#qubits ❂	Full depth ❋
Linear layer L1 [30]	8,093	60	128	2,365
Linear layer L3 [30]	18,080	90	192	5,301
Linear layer L5 [30]	32,714	137	256	8,603
Linear layer L1	8,205	0	256	225
Linear layer L3	18,418	0	384	339
Linear layer L5	32,793	0	512	455

(c) Key schedule.

Method	#CNOT ✻	#1qCliff ❁	#qubits ❂	Full depth ❋
Key schedule L1 [30]	8,104	0	128	2,438
Key schedule L3 [30]	18,242	0	192	4,896
Key schedule L5 [30]	32,525	0	256	9,358
Key schedule L1	8,183	0	128	224
Key schedule L3	18,418	0	192	340
Key schedule L5	32,772	0	256	456

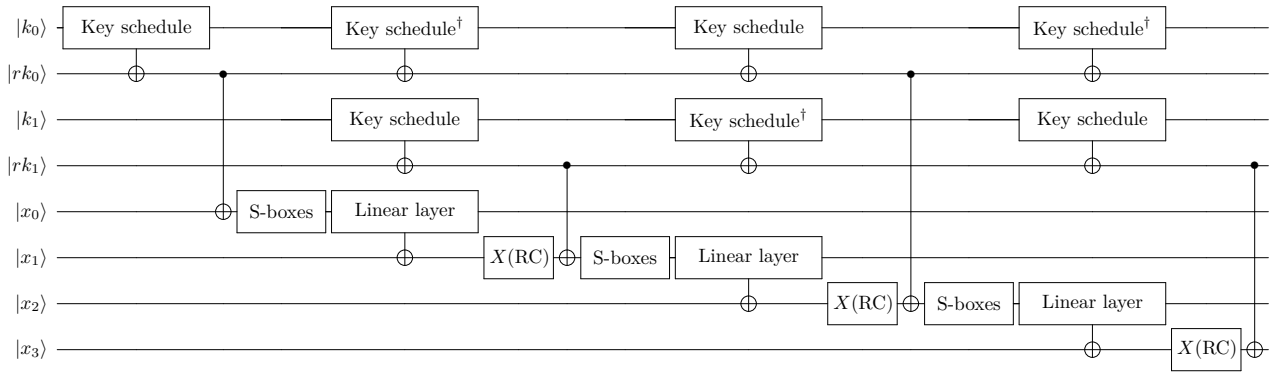
To our understanding, some problems arise if the qubits are allocated by the `using` command in Q# (and it affects the non-linear components). However more experiments are to be carried out in order to be completely certain about it. The `using` command automatically disposes when the function ends. If ancilla qubits to implement LowMC S-box are allocated with the `using` command, the consistency between depth and qubits is lost. When 10 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 10 S-boxes simultaneously. To be modified, the number of qubits must be increased or the 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.

For LowMC quantum circuits in [30], the key schedule and the linear layer are in-place implementations, so only the shallow S-box is reported as lower-bound. We correct the number of qubits so that $10 \times$ S-boxes can be operated in parallel. In LowMC, CCNOT implementation with T-depth of one in [36] is adopted rather than AND gate. This CCNOT implementation requires 4 ancilla qubits (see [36] for details). We correct the number of qubits while keeping the CCNOT implementation they adopted.

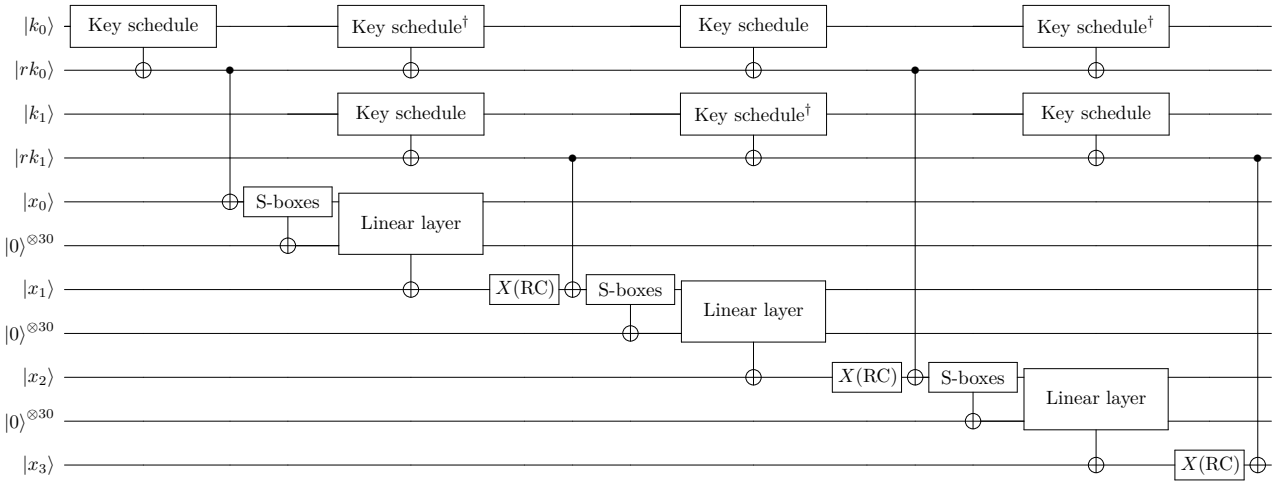
They count the qubits for ($10 \times$ S-boxes $\times$ number of rounds) as follows: ($10 \times 3 \times$ number of rounds) qubits for the output of S-boxes, 3 ancilla qubits for all shallow S-boxes, and 4 ancilla qubits for all CCNOT implementations. As a result, 607, 907 and 1,147 ancilla qubits are counted for LowMC L1, L3, and L5, respectively[10].

Now, we correct the number of ancilla qubits estimated as lower-bound. To operate 10 shallow S-boxes in parallel, 10×3 ancilla qubits are required (rather than 3 ancilla qubits). For parallel operation of three CCNOT gates in a shallow S-box, 3×4 ancilla qubits are required. Furthermore, for parallel operation of all CCNOT gates in $10 \times$ S-boxes, $10 \times 3 \times 4$ ancilla qubits are required (rather than 4 ancilla qubits). Since these ancilla qubits are initialized to zero after the operation, there is no need to clean the ancilla qubits (i.e., no need to reverse). So the count for ancilla qubits for S-boxes is correct as follows: ($10 \times 3 \times$ number of rounds)

[10] https://github.com/microsoft/grover-blocks/blob/master/numbers/lowmc.csv.



(a) Regular version.



(b) Shallow version.

Fig. 6: Architecture of LowMC quantum circuit.

ancilla qubits for the output of S-boxes, 10×3 ancilla qubits for parallel shallow S-boxes, and $10 \times 3 \times 4$ ancilla qubits for parallel CCNOT implementation. As a result, corrected 750, 1,050, and 1,290 ancilla qubits are counted for LowMC L1, L3, and L5, respectively.

For the linear layer and key schedule, there is no need for ancilla qubits as they are in-place implementations. So only (block size+key size) qubits are initially set. However, 384, 576, and 768 qubits are reported for LowMC L1, L3, and L5 respectively. We believe that only 256, 384 and 512 qubits need be set for LowMC L1, L3 and L5 respectively.

Finally, the corrected 1,056, 1,434 and 1,802 qubits are counted for LowMC L1, L3 and L5, respectively. We correct the number of qubits while maintaining their gates and circuit depth.

6 Estimating Cost of Grover's Key Search

In this part, we evaluate the performance (quantum resources required) of the proposed quantum circuits (i.e., SPECK and LowMC). Our quantum implementation results from a quantum simulator on a classical computer, not on a real quantum computer. Due to the difficulty in accessing real quantum computers (and there is also no large-scale quantum computer), most studies report quantum implementations and resource analysis on quantum simulators [7,25,6,34,28,30]. In our work, we use the quantum programming tool ProjectQ to implement and simulate quantum circuits. We use two internal libraries (`ClassicalSimulator` and `ResourceCounter`) of ProjectQ to verify the test vector and then estimate the required quantum resources. `ClassicalSimulator` can simulate large-scale quantum circuits by limiting only quantum gates with Boolean functions (i.e., that have analogy with classical gates) such as X, CNOT, and CCNOT gates. We use `ResourceCounter` to check

the number of qubits, the number of quantum gates, and the depth of our quantum circuits. Tables 2 and 4 show the quantum resources required to implement our SPECK and LowMC quantum circuits.

We estimate the cost of the Grover's key search for SPECK and LowMC based on the proposed quantum circuits. The Grover's search algorithm operates by iteration of oracle and diffusion operator. Commonly, the cost of the diffusion operator is ignored in the estimation [17,25,24,30]. The diffusion operator operates on key qubits, and has very little overhead compared to oracle. For this reason, in most studies, the cost of iterating the oracle is estimated as the final cost of the Grover's key search.

In the Grover's oracle, the quantum circuit for the target cipher is operated twice. The first operation encrypts the known plaintext using the key in superposition. Then, we need to check that the (n-bit) ciphertext in the superposition state matches the ciphertext we know. An n-Controlled X gate is used for this. This single gate (i.e., n-Controlled X gate) is also excluded from resource estimation for simplicity because it is a negligible overhead in oracle. Therefore, the cost of the oracle is calculated as the quantum resources required for the encryption circuit (Table 2 or 4) to operate twice sequentially.

As mentioned earlier, the Grover's search algorithm operates as an iteration of oracle and diffusion operator, and we exclude the cost of diffusion operator from resource estimation. Then, the final cost of the Grover's key search is calculated as (oracle × number of iterations). The number of times the Grover's oracle is applied is in turn determined by the key size. For a k-bit key (i.e., k-bit search space), the number of iterations to get the solution key is $\lfloor \frac{\pi}{4}\sqrt{2^k} \rfloor$ [14] (about $\sqrt{2^k}$). That is, the Grover's search algorithm reduces the security (by the square root) of symmetric key ciphers. Lastly, the Grover's key search on r (plaintext, ciphertext) pairs must be performed (which can be done in parallel) to exclude spurious keys. In [31,30], $r = \lceil k/n \rceil$ (plaintext, ciphertext) pairs are used for Grover's key search for ciphers using n-bit blocks and k-bit keys, and we also follow this structure.

Table 6 shows the Grover's key search cost for SPECK variants. According to the block and key size of SPECK variants, r (plaintext, ciphertext) pairs are required. However, since r (plaintext, ciphertext) pairs are operated in parallel, the depth is not affected. Table 6 and 7 show the Grover's key search cost for SPECK and LowMC variants, respectively. Table 6 and 7 are calculated as (Table 4 and 2, respectively) $\times\ 2 \times \lfloor \frac{\pi}{4}\sqrt{2^k} \rfloor \times r$ (the number of qubits is not needed in the calculation, owing to the sequential nature of the quantum circuits).

Table 6: Quantum resources required for key search for SPECK (this work).

SPECK	r	#qubits ❂	Total gates ✻	Full depth ❉	Cost ✻ × ❉	Level of security		
						NIST [32]	G$^+$ [17]	J$^+$ [25]
32/64	2	133	$1.749 \cdot 2^{47}$	$1.008 \cdot 2^{45}$	$1.747 \cdot 2^{92}$	Not achieved ($< 2^{170}$)	Not achieved ($< 2^{169}$)	Not achieved ($< 2^{157}$)
48/72	2	173	$1.357 \cdot 2^{52}$	$1.548 \cdot 2^{49}$	$1.05 \cdot 2^{102}$			
48/96	2	197	$1.419 \cdot 2^{64}$	$1.619 \cdot 2^{61}$	$1.149 \cdot 2^{126}$			
64/96	2	229	$1.089 \cdot 2^{65}$	$1.234 \cdot 2^{62}$	$1.344 \cdot 2^{127}$			
96/96	1	195	$1.181 \cdot 2^{65}$	$1.008 \cdot 2^{63}$	$1.19 \cdot 2^{128}$			
64/128	2	261	$1.132 \cdot 2^{81}$	$1.281 \cdot 2^{78}$	$1.45 \cdot 2^{159}$			Level 1 ($< 2^{222}$)
96/144	2	341	$1.854 \cdot 2^{89}$	$1.044 \cdot 2^{87}$	$1.936 \cdot 2^{176}$	Level 1 ($< 2^{233}$)	Level 1 ($< 2^{233}$)	Level 1 ($< 2^{222}$)
128/128	1	259	$1.818 \cdot 2^{81}$	$1.545 \cdot 2^{79}$	$1.404 \cdot 2^{161}$	Not achieved ($< 2^{170}$)	Not achieved ($< 2^{169}$)	Level 1 ($< 2^{222}$)
128/192	2	453	$1.42 \cdot 2^{114}$	$1.593 \cdot 2^{111}$	$1.131 \cdot 2^{226}$	Level 1 ($< 2^{233}$)	Level 1 ($< 2^{233}$)	Level 3 ($< 2^{287}$)
128/256	2	517	$1.463 \cdot 2^{146}$	$1.641 \cdot 2^{143}$	$1.201 \cdot 2^{290}$	Level 3 ($< 2^{298}$)	Level 3 ($< 2^{298}$)	Level 5 ($> 2^{287}$)

Now, we evaluate the post-quantum security levels of SPECK and LowMC based on NIST's post-quantum security requirements [32]. NIST defined the post-quantum security level as the Grover's key search cost of AES variants calculated in [17], as stated already in Section 2.4. For instance, if the complexity to mount a quantum attack for a given cipher is comparable to or more difficult to that of AES-128 (i.e., 2^{170}), then Level

Table 7: Quantum resources required for key search for LowMC (this work).

LowMC		r	#qubits ❂	Total gates ✳	Full depth ❇	Cost ✳ × ❇	Level of security		
							NIST [32]	G^+ [17]	J^+ [25]
L1	⌑	1	3,201	$1.513 \cdot 2^{83}$	$1.806 \cdot 2^{76}$	$1.366 \cdot 2^{160}$	Not achieved	Not achieved	Level 1
	⋈		3,831	$1.519 \cdot 2^{83}$	$1.806 \cdot 2^{76}$	$1.371 \cdot 2^{160}$	($< 2^{170}$)	($< 2^{169}$)	($< 2^{222}$)
L3	⌑	1	6,721	$1.259 \cdot 2^{117}$	$1.013 \cdot 2^{110}$	$1.276 \cdot 2^{227}$	Level 1	Level 1	Level 3
	⋈		7,651	$1.261 \cdot 2^{117}$	$1.013 \cdot 2^{110}$	$1.278 \cdot 2^{227}$	($< 2^{233}$)	($< 2^{233}$)	($< 2^{287}$)
L5	⌑	1	11,009	$1.412 \cdot 2^{150}$	$1.706 \cdot 2^{142}$	$1.204 \cdot 2^{293}$	Level 3	Level 3	Level 5
	⋈		12,179	$1.413 \cdot 2^{150}$	$1.706 \cdot 2^{142}$	$1.205 \cdot 2^{293}$	($< 2^{298}$)	($< 2^{298}$)	($> 2^{287}$)

⌑: Regular version.
⋈: Shallow version.

1 is said to be achieved; since the estimate of Level 1 was taken as 2^{170} in [32]. It may be stated that, the count of qubit was not directly included in computing the security levels (i.e., high full depth was allowed).

Following the security levels stated in [32], the cost of running the Grover's key search on SPECK and LowMC, for the variants of $\leq$ 128-bit sized keys, none achieves Level 1 security. When the key size is increased, SPECK using 144-bit key achieves Level 1 security; similarly the variants with 192-bit and 256-bit sized keys respectively achieve Level 1 and Level 3 security. On the other hand, the bounds that were actually computed based on the circuits presented in [17] are quite close, but not exactly the same as that of [32] for Level 1 ($< 2^{169}$ from [17], but 2^{170} in [32]).

That said, one may note that the bounds stated in [32] or [17], in some sense overestimated the cost for the respective levels. With each newer implementation, the quantum costs is reduced. In other words, as the quantum costs for the AES variants are reduced, the security levels are to be adjusted accordingly. As far as we know, the best-known implementation (i.e., with the least cost) of AES-128, 192 and 256 as quantum circuits were presented in [25]; were calculated as Level 1: $\approx 2^{157}$, Level 3: $\approx 2^{222}$, Level 5: $\approx 2^{287}$. When adjusted with these newly computed figures, we observe that SPECK and LowMC achieve Level 1 for 128-bit keys, Level 3 for 192-bit keys, and Level 5 (highest) for 256-bit keys.

Apart from the cost itself, there is another requirement from NIST in terms of full depth. The quantum circuits should have less full depth than the so-called "MAXDEPTH" limit [32]. No clear boundary for MAXDEPTH was specified; instead 2^{40}, 2^{64} and 2^{96} are to be considered as landmarks. However, as discussed in [25, Section 2.3], this limit is not always respected in the literature. Looking at Tables 6 and 7 that, one may notice that, our implementations overtook the MAXDEPTH boundaries, particularly those with larger key size. As a follow-up work, one may be interested in adopting a proper procedure (see [25, Section 2.3] for three possible options), as those are out-of-scope for this work.

7 Conclusion

In this work, we follow the previous works [6,23,30] where the quantum analysis of the SPECK and LowMC cipher families was conducted. As a synopsis of our work, it can be mentioned that, we manage to find a reduced depth implementation of the 10 SPECK variants (thereby improving from [6,23]) and 3 LowMC variants, on top of bug-fixing the LowMC implementation from [30] (and benchmark those). All in all, our implementations achieve these security bounds (which are defined in terms of the quantum cost of the AES family [32]):

- Variants of SPECK that use $\leq$ 96-bit key: Not achieved ($< 2^{157}$), SPECK-64/128: 2^{159} (Level 1), SPECK-96/144: 2^{176} (Level 1), SPECK-128/128: 2^{161} (Level 1), SPECK-128/192: 2^{226} (Level 3), SPECK-128/256: 2^{290} (Level 5);
- LowMC L1: 2^{160} (Level 1), LowMC L3: 2^{227} (Level 3), LowMC L5: 2^{293} (Level 5);

when the results from [25] are taken into account. We anticipate our work would be useful to the broader community when analyzing the quantum security of ciphers in the coming future. In particular, we anticipate future researcher will take interest in implementing other ARX ciphers (for instance, by utilizing the quantum adder, see Section 4.1) as well as SHA-256 and SHA-384 (those are important milestones to figure out the quantum security levels, see Section 2.4).

A Additional Results

Similar to [42, Table 6], we show the per-round benchmark for our implementations of the target ciphers. Table 8 shows the same for the ten variants of SPECK. Since the number of rounds is higher (and there are a total of 10 variants), we only show the benchmark for a typical (non-last round) in this case. The same for LowMC L1, L3 and L5 are shown in Table 9(a), 9(b) and 9(c), respectively. In the last round, the CNOT, NOT, and Toffoli gate costs are less since the key schedule is not performed.

Table 8: Quantum resources required for a typical round for SPECK (this work).

SPECK	#CNOT ✲	#NOT ✱	#Toffoli ☆	Toffoli depth ✦
32/64	194	52	58	29
48/72	298	84	90	45
48/96	298	84	90	45
64/96	402	116	122	61
64/128	402	116	122	61
96/96	610	180	186	93
96/144	610	180	186	93
128/128	818	244	250	125
128/192	818	244	250	125
128/256	818	244	250	125

References

1. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 430–454. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_17, https://doi.org/10.1007/978-3-662-46800-5_17 1, 5
2. Almazrooie, M., Samsudin, A., Abdullah, R., Mutter, K.N.: Quantum reversible circuit of AES-128. Quantum Information Processing **17**(5), 1–30 (may 2018). https://doi.org/10.1007/s11128-018-1864-3, https://doi.org/10.1007/s11128-018-1864-3 3
3. 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. pp. 317–337. Springer International Publishing, Cham (2017) 3
4. 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), 818–830 (Jun 2013). https://doi.org/10.1109/tcad.2013.2244643, http://dx.doi.org/10.1109/TCAD.2013.2244643 4, 9
5. Anand, R., Maitra, A., Maitra, S., Mukherjee, C.S., Mukhopadhyay, S.: Quantum resource estimation for FSR based symmetric ciphers and related Grover's attacks. In: Adhikari, A., Küsters, R., Preneel, B. (eds.) Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13143, pp. 179–198. Springer (2021). https://doi.org/10.1007/978-3-030-92518-5_9, https://doi.org/10.1007/978-3-030-92518-5_9 3
6. Anand, R., Maitra, A., Mukhopadhyay, S.: Evaluation of quantum cryptanalysis on SPECK. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) Progress in Cryptology – INDOCRYPT 2020. pp. 395–413. Springer International Publishing, Cham (2020) 2, 3, 6, 8, 13, 15
7. Anand, R., Maitra, A., Mukhopadhyay, S.: Grover on SIMON. Quantum Information Processing **19**(9) (Sep 2020). https://doi.org/10.1007/s11128-020-02844-w, http://dx.doi.org/10.1007/s11128-020-02844-w 3, 13
8. Baksi, A., Bhattacharjee, A., Breier, J., Isobe, T., Nandi, M.: Big brother is watching you: A closer look at backdoor construction. Cryptology ePrint Archive, Paper 2022/953 (2022), https://eprint.iacr.org/2022/953, https://eprint.iacr.org/2022/953 5
9. 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). https://doi.org/10.1007/s11128-021-03307-6, https://doi.org/10.1007/s11128-021-03307-6 3

Table 9: Quantum resources required per round for LowMC variants (this work).

(a) L1

Round	#CNOT ✲		#NOT ✱	#Toffoli ☆	Toffoli depth ✦	
	⌑	⋈	⌑ ⋈	⌑ ⋈	⌑	⋈
1≀	41,048	41,148	60	30	3	1
2	24,691	24,791	66	30	3	1
3	24,752	24,852	62	30	3	1
4	24,773	24,873	78	30	3	1
5	24,830	24,930	68	30	3	1
6	24,997	25,097	53	30	3	1
7	24,833	24,933	73	30	3	1
8	24,762	24,862	61	30	3	1
9	24,581	24,681	65	30	3	1
10	24,670	24,770	54	30	3	1
11	24,628	24,728	58	30	3	1
12	24,760	24,860	60	30	3	1
13	24,780	24,880	71	30	3	1
14	24,735	24,835	67	30	3	1
15	24,749	24,849	58	30	3	1
16	24,715	24,815	71	30	3	1
17	24,673	24,773	57	30	3	1
18	24,541	24,641	55	30	3	1
19	16,544	16,644	59	30	3	1
20	16,568	16,668	70	30	3	1

≀: Including initial key XOR.

⌑: Regular version.

⋈: Shallow version.

10. Banegas, G., Bernstein, D.J., Van Hoof, I., Lange, T.: Concrete quantum cryptanalysis of binary elliptic curves. Cryptology ePrint Archive (2020) 1
11. Bathe, B.N., Anand, R., Dutta, S.: Evaluation of Grover's algorithm toward quantum cryptanalysis on ChaCha. Quantum Inf. Process. **20**(12), 394 (2021). https://doi.org/10.1007/s11128-021-03322-7, https://doi.org/10.1007/s11128-021-03322-7 3
12. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013), https://eprint.iacr.org/2013/404 1, 4
13. Bijwe, S., Chauhan, A.K., Sanadhya, S.K.: Quantum search for lightweight block ciphers: Gift, skinny, saturnin. Cryptology ePrint Archive, Paper 2020/1485 (2020), https://eprint.iacr.org/2020/1485, https://eprint.iacr.org/2020/1485 3
14. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik **46**(4-5), 493–505 (Jun 1998). https://doi.org/10.1002/(sici)1521-3978(199806)46:4/5¡493::aid-prop493¿3.0.co;2-p, http://dx.doi.org/10.1002/(SICI)1521-3978(199806)46:4/5<493::AID-PROP493>3.0.CO;2-P 14
15. Cuccaro, S., Draper, T., Kutin, S., Moulton, D.: A new quantum ripple-carry addition circuit. arXiv (2008), https://arxiv.org/pdf/quant-ph/0410184.pdf 6, 7
16. Gidney, C.: Factoring with n+ 2 clean qubits and n-1 dirty qubits. arXiv preprint arXiv:1706.07884 (2017) 1
17. Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying Grover's algorithm to AES: Quantum resource estimates. In: Takagi, T. (ed.) Post-Quantum Cryptography. pp. 29–43. Springer International Publishing, Cham (2016) 3, 4, 14, 15
18. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 212–219 (1996) 1
19. Häner, T., Jaques, S., Naehrig, M., Roetteler, M., Soeken, M.: Improved quantum circuits for elliptic curve discrete logarithms. In: International Conference on Post-Quantum Cryptography. pp. 425–444. Springer (2020) 1
20. Häner, T., Roetteler, M., Svore, K.M.: Factoring using 2n+ 2 qubits with toffoli based modular multiplication. arXiv preprint arXiv:1611.07995 (2016) 1

(b) L3

Round	#CNOT ✲		#NOT ✱	#Toffoli ☆	Toffoli depth ✦	
	⌑	⋈	⌑ ⋈	⌑ ⋈	⌑	⋈
1≀	92,465	92,565	90	30	3	1
2	55,424	55,524	96	30	3	1
3	56,102	56,202	86	30	3	1
4	55,495	55,595	98	30	3	1
5	55,622	55,722	96	30	3	1
6	55,632	55,732	100	30	3	1
7	55,286	55,386	99	30	3	1
8	55,678	55,778	97	30	3	1
9	55,463	55,563	104	30	3	1
10	55,623	55,723	101	30	3	1
11	55,464	55,564	101	30	3	1
12	55,352	55,452	93	30	3	1
13	55,606	55,706	104	30	3	1
14	55,324	55,424	84	30	3	1
15	55,001	55,101	94	30	3	1
16	55,273	55,373	112	30	3	1
17	55,319	55,419	95	30	3	1
18	55,657	55,757	98	30	3	1
19	55,014	55,114	90	30	3	1
20	55,533	55,633	98	30	3	1
21	55,386	55,486	101	30	3	1
22	55,155	55,255	94	30	3	1
23	55,531	55,631	97	30	3	1
24	55,427	55,527	88	30	3	1
25	55,494	55,594	108	30	3	1
26	55,409	55,509	92	30	3	1
27	55,591	55,691	91	30	3	1
28	55,387	55,487	102	30	3	1
29	37,182	37,282	98	30	3	1
30	37,211	37,311	92	30	3	1

≀: Including initial key XOR.

⌑: Regular version.

⋈: Shallow version.

21. 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), 2350–2361 (2017) 4
22. 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
23. Jang, K., Choi, S., Kwon, H., Kim, H., Park, J., Seo, H.: Grover on Korean block ciphers. Applied Sciences **10**(18) (2020). https://doi.org/10.3390/app10186407, https://www.mdpi.com/2076-3417/10/18/6407 2, 3, 8, 15
24. 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, https://eprint.iacr.org/2022/647 3, 14
25. Jang, K., Baksi, A., Kim, H., Song, G., Seo, H., Chattopadhyay, A.: Quantum analysis of aes. Cryptology ePrint Archive, Paper 2022/683 (2022), https://eprint.iacr.org/2022/683, https://eprint.iacr.org/2022/683 2, 3, 13, 14, 15
26. Jang, K., Choi, S., Kwon, H., Seo, H.: Grover on SPECK: Quantum resource estimates. Cryptology ePrint Archive, Report 2020/640 (2020), https://eprint.iacr.org/2020/640 6, 8
27. 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), https://www.mdpi.com/2076-3417/11/11/4776 3
28. Jang, K., Song, G., Kim, H., Kwon, H., Kim, H., Seo, H.: Parallel quantum addition for Korean block cipher. IACR Cryptol. ePrint Arch. p. 1507 (2021), https://eprint.iacr.org/2021/1507 13

(c) L5

Round	#CNOT ✲		#NOT ✱	#Toffoli ☆	Toffoli depth ✦	
	⌑	⋈	⌑ ⋈	⌑ ⋈	⌑	⋈
1≀	164,178	164,278	137	30	3	1
2	98,782	98,882	127	30	3	1
3	98,975	99,075	119	30	3	1
4	98,442	98,542	122	30	3	1
5	98,619	98,719	127	30	3	1
6	98,496	98,596	131	30	3	1
7	98,331	98,431	130	30	3	1
8	98,554	98,654	129	30	3	1
9	98,889	98,989	112	30	3	1
10	98,153	98,253	139	30	3	1
11	98,592	98,692	122	30	3	1
12	98,608	98,708	135	30	3	1
13	98,521	98,621	123	30	3	1
14	98,547	98,647	121	30	3	1
15	98,350	98,450	132	30	3	1
16	98,841	98,941	122	30	3	1
17	98,619	98,719	120	30	3	1
18	97,781	97,881	132	30	3	1
19	98,796	98,896	133	30	3	1
20	98,307	98,407	145	30	3	1
21	98,671	98,771	124	30	3	1
22	98,957	99,057	127	30	3	1
23	98,798	98,898	135	30	3	1
24	98,347	98,447	146	30	3	1
25	98,993	99,093	130	30	3	1
26	98,927	99,027	125	30	3	1
27	99,137	99,237	139	30	3	1
28	98,486	98,586	122	30	3	1
29	99,272	99,372	127	30	3	1
30	98,674	98,774	126	30	3	1
31	98,273	98,373	123	30	3	1
32	98,832	98,932	125	30	3	1
33	98,598	98,698	126	30	3	1
34	98,781	98,881	127	30	3	1
35	98,494	98,594	130	30	3	1
36	98,454	98,554	123	30	3	1
37	65,565	65,665	121	30	3	1
38	66,034	66,134	123	30	3	1

≀: Including initial key XOR.

⌑: Regular version.

⋈: Shallow version.

29. Jang, K., Song, G., Kwon, H., Uhm, S., Kim, H., Lee, W.K., Seo, H.: Grover on pipo. Electronics **10**(10), 1194 (2021) 3

30. 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. Lecture Notes in Computer Science, vol. 12106, pp. 280–310. Springer (2020). https://doi.org/10.1007/978-3-030-45724-2_10, https://doi.org/10.1007/978-3-030-45724-2_10 2, 3, 4, 9, 11, 12, 13, 14, 15

31. 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**, 1–12 (01 2020). https://doi.org/10.1109/TQE.2020.2965697 3, 14
32. 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, 4, 14, 15
33. Putranto, D.S.C., Wardhani, R.W., Larasati, H.T., Kim, H.: Another concrete quantum cryptanalysis of binary elliptic curves. Cryptology ePrint Archive (2022) 1
34. Rahman, M., Paul, G.: Grover on katan: Quantum resource estimation. IEEE Transactions on Quantum Engineering **3**, 1–9 (2022) 3, 13
35. Selinger, P.: Quantum circuits of t-depth one. Physical Review A **87**(4), 042302 (2013) 4
36. Selinger, P.: Quantum circuits of *t*-depth one. Phys. Rev. A **87**, 042302 (Apr 2013). https://doi.org/10.1103/PhysRevA.87.042302, https://link.aps.org/doi/10.1103/PhysRevA.87.042302 12
37. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science. pp. 124–134. Ieee (1994) 1
38. 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 3
39. Takahashi, Y., Tani, S., Kunihiro, N.: Quantum addition circuits and unbounded fan-out (2009), https://arxiv.org/abs/0910.2530 6
40. Zaverucha, G., Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Katz, J., Wang, X., Kolesnikov, V., Kales, D.: The Picnic signature algorithm. Submission to PQC third round (2020), https://github.com/microsoft/Picnic/blob/master/spec/spec-v3.0.pdf 5
41. 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), 1–38 (2022) 3
42. 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. pp. 697–726. Springer International Publishing, Cham (2020) 2, 3, 11, 16