Noname manuscript No.
(will be inserted by the editor)

Quantum Implementation and Resource Estimates for RECTANGLE and KNOT

Anubhab Baksi * · **Kyungbae Jang** * · **Gyeongju Song** · **Hwajeong Seo** · **Zejun Xiang**

Received: date / Accepted: date

Abstract With the advancement of the quantum computing technologies, a large body of research work is dedicated to revisit the security claims for ciphers being used. An adversary with access to a quantum computer can employ certain new attacks which would not be possible in the current pre-quantum era. In particular, the Grover's search algorithm is a generic attack against symmetric key cryptographic primitives, that can reduce the search complexity to square root. To apply the Grover's search algorithm, one needs to implement the target cipher as a quantum circuit. Although relatively recent, this field of research has attracted serious attention from the research community, as several ciphers (like AES, GIFT, SPECK, SIMON etc.) are being implemented as quantum circuits. In this work, we target the lightweight block cipher RECTANGLE and the Authenticated Encryption with Associated Data (AEAD) KNOT which is based on RECTANGLE; and implement those in the ProjectQ library (an open-source quantum compatible library designed by researchers from ETH Zurich). AEADs are considerably more complex to implement than a typical block/stream cipher, and ours is among the first works to do this. The implementations reported here are simulated on classical computer (as long as it is feasible).

Keywords Lightweight Cryptography · Quantum Computing · RECTANGLE · KNOT · Grover's search

A. Baksi
Nanyang Technological University, Singapore
E-mail: anubhab001@e.ntu.edu.sg
Corresponding author

K. Jang
Division of IT Convergence Engineering, Hansung University, Seoul, South Korea
E-mail: starj1023@gmail.com
Corresponding author

G. Song
Division of IT Convergence Engineering, Hansung University, Seoul, South Korea
E-mail: thdrudwn98@gmail.com

H. Seo
Division of IT Convergence Engineering, Hansung University, Seoul, South Korea
E-mail: hwajeong84@gmail.com

Z. Xiang
Faculty of Mathematics and Statistics, Hubei Key Laboratory of Applied Mathematics, Hubei University, Wuhan, PR China
E-mail: xiangzejun@hubu.edu.cn

1 Introduction

Quantum computing [19] makes use of the properties of the fundamental particles for the computation. Quantum computing has been advancing in leaps and bounds over the last few years. Technological breakthroughs are being reported frequently. This leads to serious consequences in several research fields, including cryptography [17].

One such prominent algorithm in quantum computing which has found its application in private key cryptography is the Grover's search algorithm [10]. In summary, the Grover's search can find the secret key for a private key cipher at the square root search space of the classical search space. For example, for a 128-bit cipher like `AES`, the search complexity for the Grover's search would be of the order of 2^{64}. In other words, with the help of a powerful enough quantum computer, the attacker would be able to find (with a high probability) the key with a search complexity of around 2^{64}.

Computation in the quantum realm follows the so-called *reversible computing* paradigm[1] [5]. Under this

[1] Except for measurement, see Section 2.3.

paradigm, the entropy is preserved throughout every step of computation, hence certain design considerations are to be made (see Section 2.2).

The objective of this paper is to show a quantum-compatible implementation of two lightweight ciphers, namely RECTANGLE [23] and KNOT [24]. This enables us to estimate the cost that would be required to run the ciphers on an actual quantum computer. Further, in order to apply the Grover's search, certain other circuitry are required on top of the cipher implementation. We explore this direction too and estimate the related costs. In this way, we are able to evaluate the security of those ciphers even against an adversary with access to a quantum computer.

1.1 Our Contribution

The contribution in this paper can be summarised as follows:

1. We report the first quantum implementation of the lightweight block cipher RECTANGLE (which has two variants, namely, RECTANGLE-80 and RECTANGLE-128) [23]. Further, we report the first implementation of the lightweight AEAD KNOT (multiple variants of the cipher) [24].
2. Also, we show analysis on the circuit complexity required to mount the Grover's search for both RECTANGLE and KNOT. In the process, we discover that multiple versions of KNOT fail to meet the NIST recommended threshold for quantum security.
3. We use an optimised implementation to keep the number of qubits low.
4. We also use optimisation to keep the number of gates low.

Thus, we combine quantum implementation and cost estimate for two ciphers at one go. Save for [4], our work is the first to consider an AEAD (which is arguably more complex than a block or stream cipher as the state size is typically larger, the number of initialisation rounds is typically higher, not to mention a higher number of input lines) in the scope, to the best of our knowledge. Our source code is available as open-source[2]. We validate our implementation by simulating it on classical computer, as long as it is practicable.

1.2 Previous Works

Several reports of quantum implementation of symmetric key ciphers as well as estimation of quantum resources required for the Grover's search are published. One may refer to, for example, AES [9, 15, 16, 25], SIMON [3], SPECK [13], GIFT [12], PRESENT [14]. In case of AEADs, the authors of [4] report quantum implementation GRAIN-128-AEAD and TINYJAMBU.

On the other hand, there have been attempts to construct tools that enable efficient implementation of ciphers in reversible/quantum realm. For example, the LIGHTER-R [8] can find reversible implementation of 4×4 SBoxes. The implementation works in-place, thus no ancilla or garbage line is created. Similarly, the sequential XOR implementation from [22] enables implementation for binary non-singular matrices (which are used as the linear layer for ciphers like AES) as reversible circuits.

2 Background

2.1 Quantum Bit (Qubit)

A classical computer uses a bit as the fundamental unit, the analogy to that in a quantum computer would be a *quantum bit* or *qubit* for short. A qubit has a two dimensional state. However, unlike a classical bit, which can only take logic 0 or logic 1; a qubit can be at any state $\langle\psi| = a\langle 0| + b\langle 1|$, where a and b are complex numbers with $|a|^2+|b|^2= 1$, and $\langle 0|$ and $\langle 1|$ respectively correspond to logic 0 and logic 1 states of the qubit. This is known as *superposition* and is not available in a classical computer. In other words, whereas a classical bit can take one of the two discrete states (logic 0 or logic 1), a qubit state can take any point in the unit sphere through superposition. For instance, the polarization of a single photon can represent a qubit (here the state can be described as a superposition of the vertical polarization and horizontal polarization). The expression $a\langle 0| + b\langle 1|$ is often written in the matrix form, $\begin{bmatrix} a \\ b \end{bmatrix}$. Here we call a and b as the *amplitudes* of $\langle 0|$ and $\langle 1|$, respectively.

2.2 Reversible Computing

In the reversible computing paradigm, computation is described as a bijection. Therefore, the entropy is preserved. Put in other words, the input to a reversible computer can be uniquely computed backwards given the output.

The NOT gate is reversible, since the input can be uniquely computed given its output. In contrast, the (2-input) XOR gate is not a reversible circuit; since it is not possible to compute both the inputs given its output. In order to get a reversible XOR gate, the minimum requirement is to insert one extra output line, such

[2] https://github.com/starj1023/KNOT-QC/.

that this output line directly gives one of the inputs. This is what is known as the CNOT gate (see Figure 1(b)). Any such extra output line is called a *garbage*. There may arise a situation where an extra input line is to be considered to make the circuit reversible, which (typically initialized with a constant) is referred to as an *ancilla*. It may be noted that a reversible circuit does not have fan-out (feed-forward), nor fan-in (feed-backward). One of the consequences is that, a reversible circuit cannot have iterations (has to be loop-unrolled).

The common reversible gates are described next. The circuit representation of those gates can be found in Figure 1.

- *NOT/X* gate: $\text{NOT}(a) = \overline{a}$.
- *CNOT (Controlled NOT)/CX/Feynman*: $\text{CNOT}(a, b) = (a, a \oplus b)$. This gate flips one output (known as the target) line if and only if the the other input line (known as the control) is at logic 1.
- *CCNOT/CCX/Toffoli* gate: $\text{CCNOT}(a, b, c) = (a, b, ab \oplus c)$. This gate can be generalized with Tof_n gate, where first $n-1$ variables are used as control lines; i.e., $\text{Tof}_n(a_0, a_1, \ldots, a_{n-2}, a_{n-1}) = (a_0, a_1, \ldots, a_{n-2}, a_0 a_1 \cdots a_{n-2} \oplus a_{n-1})$. The NOT and CNOT gates are sometimes denoted as Tof_1 and Tof_2, respectively.
- *SWAP gate*: $\text{SWAP}(a, b) = (b, a)$.

We present an example of the half adder in the reversible computing paradigm in Figure 2 for more clarity. It can be seen that one input line is set as logic 0 (ancilla), and one output line is not used (garbage).

Quantum computing is inherently reversible, this appears from the very nature of physical principles it is based on. However, there can be some technology which follows the reversible computing paradigm, but not quantum (although no prominent non-quantum reversible technology exists as of yet, to the best of our knowledge).

2.3 Quantum Computing

As the quantum computing is reversible, it uses the reversible gates (common reversible gates are covered in Section 2.2). Further, quantum computing allows certain other gates specialised for quantum application.

The relevant gates are described next, the corresponding circuit diagrams can be found in Figure 3.

- *Hadamard gate*: The Hadamard gate changes the superposition of the state. It can be described by pre-multiplication with the matrix, $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. It may be noted that, by applying the Hadamard gate twice, the state of a qubit returns to its original superposition.
- *Measurement gate*: With the measurement gate, a qubit collapses to a classical bit. Given the state, $\langle\psi| = a\langle 0| + b\langle 1|$, the measurement gate will return a classical bit, either logic 0 with probability $|a|^2$ or logic 1 with probability $|b|^2$. Thus, the property of reversibility is lost. As it can be seen from Figure 3(b), the classical bit is indicated by two lines.

For example, when the Hadamard gate is applied to a qubit with state $\langle 0|$, it enters a new state given by the superposition, $\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \langle 0| + \frac{1}{\sqrt{2}} \langle 1|$. Thus, the probability of measuring logic 0 is $\frac{1}{2}$, and that of logic 1 is $\frac{1}{2}$ too. Similarly, when the Hadamard gate is applied to the qubit with state $\langle 1|$, it enters the state $\frac{1}{\sqrt{2}} \langle 1| + \frac{1}{\sqrt{2}} \langle -1|$. Again in this case, it can be shown that the probability of measuring logic 0 as well as logic 1 are both $\frac{1}{2}$.

2.4 Grover's Search

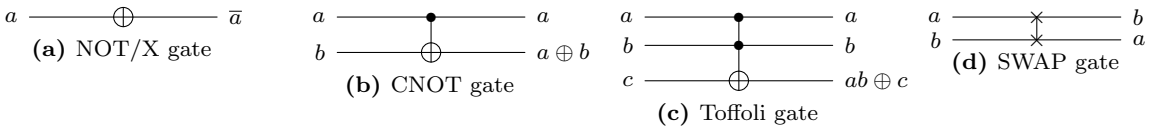
Introduced in [10], the Grover's search is a well-known algorithm in quantum computing. In a nutshell, it takes a function $f(\cdot)$, searches the implicit list of its possible inputs, and returns the input that for which the function returns a particular output (say, y) with high probability. Given N such inputs, it finds the desired input (with high probability) with around $\lfloor \frac{\pi}{4} \cdot \sqrt{N} \rfloor$ searches. Thus, compared to the classical algorithm which searches through the list of inputs one-by-one (and hence is of complexity of N), the Grover's search offers quadratic improvement.

In summary, it works by amplifying the amplitude of the state $\langle\psi|$ for which $f(\hat{x}) = y$. This way, the probability of measuring the state, $\langle\psi| = \hat{x}$, is higher than that of any other $\langle\psi| \neq \hat{x}$. Grover's search consists of oracle and diffusion operator. First, before performing the oracle and diffusion operator, the N inputs are made superposition state by applying Hadamard gates. The oracle reverses the sign by searching for the correct input of the superposition state. The diffusion operator increases the probability of measurement of the correct input by amplifying the amplitude of the inverted sign in the oracle. Grover's search repeats the oracle and diffusion operator to sufficiently increase the probability that the answer will be measured.

3 Description of the Ciphers

3.1 `RECTANGLE`

`RECTANGLE` [23] is a bit-slice lightweight block cipher designed by Zhang et al., which is suitable for low-



(a) NOT/X gate **(b)** CNOT gate **(c)** Toffoli gate **(d)** SWAP gate

Fig. 1: Circuit diagrams of common reversible gates

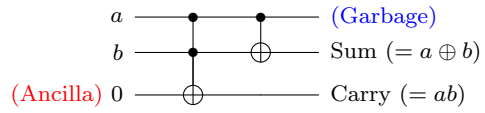


Fig. 2: Reversible circuit of a half adder

cost implementation on multiple platforms. It adopts an Substitution Permutation Network (SPN) structure. The block size of `RECTANGLE` is 64 bits, and the key size has two versions, i.e., 80 bits and 128 bits respectively. The 64 bits state of `RECTANGLE` are arranged as a 4×16 rectangle. At each round, each of the 16 columns of the state is substituted by a 4-bit SBox S, then followed by a circular shift for each row with different shift parameters (refer to Figure 4). The SBox, S, is given by `65CA1E79B03D8F42`[3]. The coordinate functions of S in algebraic normal form (ANF) are given by:

$$
\begin{aligned}
y_0 &= x_0x_1 \oplus x_0 \oplus x_2 \oplus x_3,\\
y_1 &= x_0 \oplus x_1x_3 \oplus x_1 \oplus x_2 \oplus 1,\\
y_2 &= x_0x_1x_2 \oplus x_0x_1 \oplus x_0x_2 \oplus x_1x_2 \oplus x_2x_3\\
&\quad \oplus x_2 \oplus x_3 \oplus 1,\\
y_3 &= x_0x_2 \oplus x_0x_3 \oplus x_1x_2x_3 \oplus x_1x_2 \oplus x_1 \oplus x_3.
\end{aligned}
$$

For the 80-bit key version, the key bits are loaded into a 5×16 array. The first four rows are extracted as the sub-key of the current round, then the array is updated by applying SBox to the bits intersected at the four uppermost rows and the four rightmost columns, followed by a 1-round generalized Feistel transformation and a round constant addition (see Figure 5(a) for a pictorial description). For the 128-bit key version, the key bits are loaded into a 4×32 array. The rightmost 16 columns are extracted as the sub-key, then the array is updated by applying SBox to the rightmost eight columns, followed by a 1-round generalised Feistel transformation and a round constant addition (see Figure 5(b)).

[3]We indicate the SBoxes by the shorthand string-based notation, as opposed to the more common table-based notation, to save space.

3.2 `KNOT`

`KNOT` [24] is one of the 32 candidates in the second round of the NIST Lightweight Cryptography (LWC)[4] project. The `KNOT` family consists of lightweight authenticated encryption algorithms and hash functions. In this paper, we focus only on its authenticated encryption algorithms, which are based on the MonkeyDuplex structure. There are four members of authenticated encryption denoted as `KNOT-AEAD`(k, b, r), where k, b, r are the key size (the nonce and tag sizes are equal to the key size), state width and bit rate, respectively. For different versions, the parameters can be found in Table 1[5]. The `KNOT-AEAD` family uses a 6-bit (resp., a 7-bit) linear feedback shift registers to generate round constants CONST_6 (resp., CONST_7) for different versions, and the 6-bit (resp., 7-bit) constant is added into the first 6 (resp., 7) bits of the state. The encryption process is consist of four steps.

3.2.1 Padding

Padding of the given associated data AD and plaintext P may be needed before encryption. First, a single 1 is inserted as the most significant bit followed with a minimal number of 0's to make the length of associated data and plaintext multiple of r. If the associated data or plaintext is the empty string, nothing is done.

3.2.2 Initialisation

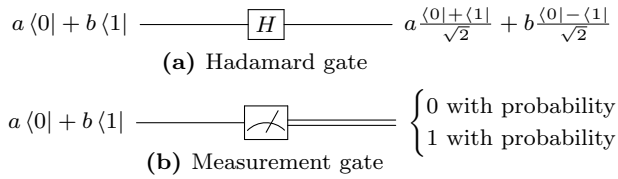
This step loads the key and nonce into the state, i.e., the b-bit state is initialised as:

$$
S = \begin{cases} (0^{128}||K||N) \oplus (1||0^{383}) \\ \qquad \text{for } \texttt{KNOT-AEAD}(128, 384, 192), \\ K||N \\ \qquad \text{for other versions.} \end{cases}
$$

Then, the state is updated by a public permutation with nr_0 iterated rounds. The public permutation will be described later.

[4]https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates.

[5]It may be noted that, $c = b - r$ is the *capacity*.



(a) Hadamard gate

(b) Measurement gate

Fig. 3: Circuit diagrams of common quantum gates

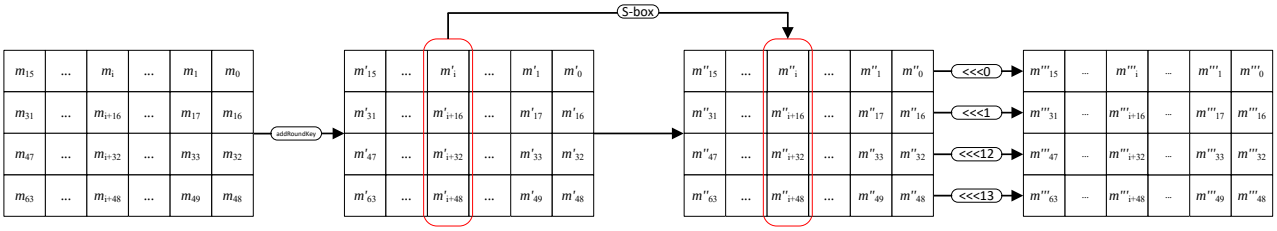
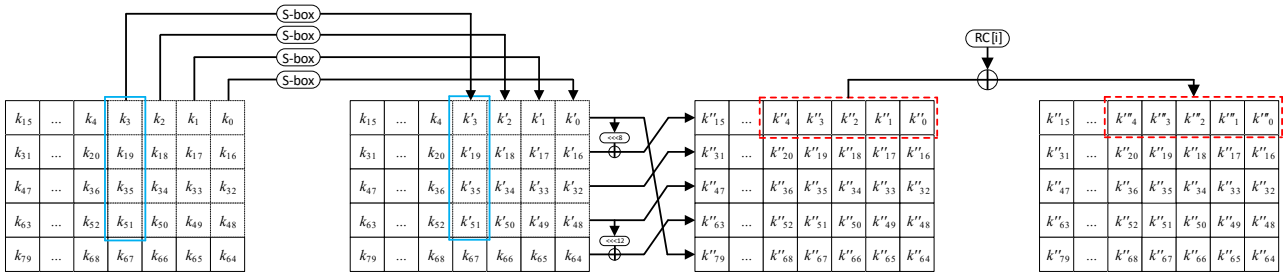
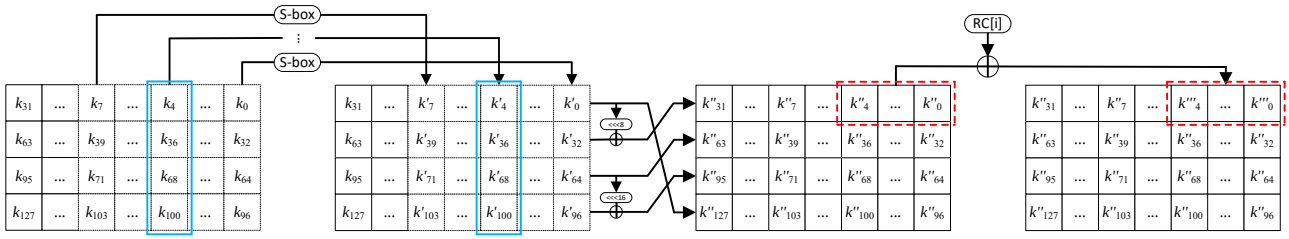


Fig. 4: Schematic for round function of RECTANGLE



(a) RECTANGLE-80



(b) RECTANGLE-128

Fig. 5: Schematic for key schedule of RECTANGLE

Table 1: Parameters for the four members of authenticated encryption

KNOT-AEAD Variant	Parameters				Constants	Rounds		
	k	b	r	$c\ (=b-r)$		nr_0	nr	nr_f
KNOT-AEAD (128,256,64)	128	256	64	192	CONST_6	52	28	32
KNOT-AEAD (128,384,192)	128	384	192	192	CONST_7	76	28	32
KNOT-AEAD (192,384,96)	192	384	96	288	CONST_7	76	40	44
KNOT-AEAD (256,512,128)	256	512	128	384	CONST_7	100	52	56

3.2.3 Processing Associated Data

The padded associated data is first cut into r-bit blocks, and each block is processed as follows. The r-bit block is added to the first r bits of the state, then update the state with an nr-round permutation. After all the associated data blocks are processed or if the associated

data is empty, a domain separation bit 1 is added to the last bit of the state.

3.2.4 Encryption

The padded plaintext is first cut into r-bit blocks, and each block is processed by adding it to the first r bits of the state, then these r bits are extracted as a ciphertext block. If the current block is not the last block, the state is updated by an nr-round permutation. Otherwise, the state is not updated and this last ciphertext block is truncated to its real size, i.e., the size of the last block before padding.

3.2.5 Finalisation

After all plaintext blocks being processed, the state is updated by a nr_f-round permutation and the first k bits is returned as the tag. In the encryption process, the tag is returned as part of the output. However, we have to compare this tag with the user received tag in the decryption process, and the plaintext is returned only if these two tags are the same.

Note that the `KNOT-AEAD` encryption and decryption are quite similar, except for the third step. Thus, we just specifies the third step of `KNOT-AEAD` decryption as follows.

3.2.6 Decryption

The padded ciphertext is first cut into r-bit blocks, and each block is processed as follows. The first r bits of the state is extracted as the key stream, that is added with the current ciphertext block to generate the corresponding plaintext block. Then, the first r bits of the state is replaced by the current ciphertext block. If the current block is not the last block, the state is updated by an nr-round permutation. Otherwise, the state is not updated and this last plaintext block is truncated to its real size.

3.3 `KNOT-PERMUTATION`

The `KNOT-PERMUTATION` can be characterised by the width parameter b, where $b \in \{256, 384, 512\}$. Therefore it has a state size of b-bits, which is organized in a two dimensional $4 \times \frac{b}{4}$ matrix, which can be found in [24, Chapter 2.1]. At each round, an SPN round transformation (denoted by, p_b) is applied over iterations. Each p_b consists of the 3 respective steps: $\texttt{AddRoundConstant}_b$, $\texttt{SubColumn}_b$, $\texttt{ShiftRow}_b$.

The $\texttt{AddRoundConstant}_b$ subroutine adds round constants CONST_6 (or CONST_7) to the state, more description can be found at [24, Chapter 2.2]. Next, the $\texttt{SubColumn}_b$ step updates the state by applying a 4-bit SBox, which is given by `40A7BE1D9F6852C3`, in column-major fashion (similar to `RECTANGLE`). The coordinate functions of this SBox in ANF are given by:

$$y_0 = x_0x_1x_3 \oplus x_0x_1 \oplus x_0x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_2 \oplus x_3,$$
$$y_1 = x_0x_3 \oplus x_1x_2x_3 \oplus x_1 \oplus x_2x_3 \oplus x_2,$$
$$y_2 = x_0 \oplus x_1x_2 \oplus x_1 \oplus x_2 \oplus x_3 \oplus 1,$$
$$y_3 = x_0x_1 \oplus x_1 \oplus x_2 \oplus x_3.$$

Lastly, the $\texttt{ShiftRow}_b$ step left-rotates the state in a row-major fashion. The i^{th} row of the state matrix is rotated c_i bits, $i = 0, 1, 2, 3$; where $c_0 = 0$ and $c_1 = 1$ for all versions, $c_2 = 8$ and $c_3 = 25$ for $b = 256$, $c_2 = 8$ and $c_3 = 55$ for $b = 384$, $c_2 = 16$ and $c_3 = 25$ for $b = 512$. The designers' recommend to use 28-rounds for $b = 256$ and 52-rounds for $b = 512$ [24, Table 2].

4 Quantum Circuit Design Methodology

4.1 `RECTANGLE`

We designed a `RECTANGLE` quantum circuit that allocates only qubits for plaintext and key. 144 qubits were allocated to the `RECTANGLE`-80 quantum circuit implementation, and 192 qubits were allocated to the `RECTANGLE`-128 quantum circuit implementation. Also, many of the quantum gates of the proposed algorithms to implement `AddRoundkey`, `SubColumn`, `ShiftRow`, and `Key Schedule` are performed in parallel. As a result, our `RECTANGLE`-80 and `RECTANGLE`-128 quantum circuits have a low depth of 266.

4.1.1 AddRoundkey

`AddRoundKey`, which XORs the 64-qubit round key RK to the 64-qubit block B, is implemented using only CNOT gates. Round key RK and block B are the inputs of the CNOT gate, and $RK \oplus B$ is stored in block B. The quantum circuit design for `AddRoundKey` is described in Algorithm 1.

4.1.2 SubColumn

`RECTANGLE` is a lightweight block code optimised for bit-slice, with SBox applied to each column. When implementing `SubColumn` on quantum computers, it is very inefficient to use a table type SBox that derives a specific output according to the input. Since qubits in the superposition state represent all values, in order to use the

Algorithm 1: `AddRoundKey` quantum circuit implementation

Input: 64-qubit block B $(b_{63}, \ldots, b_0)$, 64-qubit round key RK $(rk_{63}, \ldots, rk_0)$
Output: 64-qubit block B $(b_{63}, \ldots, b_0)$
1: **for** $i = 0$ to 63 **do**
2: $\quad b_i \leftarrow \text{CNOT}(rk_i, b_i)$
3: **return** $B(b_{63}, \ldots, b_0)$

SBox table, it is necessary to implement a quantum circuit that checks all input values and generates outputs. On the other hand, if the SBox operation in algebraic normal form is implemented as a quantum circuit, outputs for all inputs can be generated at once. However, in order to implement the `RECTANGLE` SBox operation in algebraic normal form as a quantum circuit, temp qubits are required to store the input X (x_3, x_2, x_1, x_0).

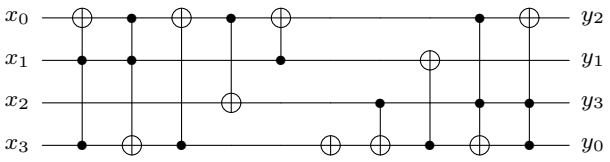


Fig. 6: Quantum implementation of `RECTANGLE` SBox (`65CA1E79B03D8F42`, using LIGHTER-R)

We propose a 4-qubit quantum SBox implementation that generates the output of the SBox table no matter what input comes in without temp qubits. To design the proposed 4-qubit quantum SBox, we use the LIGHTER-R tool [8] (with the MCT_gc library; and the cost incurred is 4 Toffoli gates, 5 CNOT gates, and 1 NOT gate). As a result, we obtain the optimal quantum SBox in terms of qubits and quantum gates. The implementation using LIGHTER-R is described in Algorithm 2/Figure 6.

4.1.3 ShiftRow

In `ShiftRow`, each row of 64-qubit block B is rotated in units of 16 qubits. Changing the position between qubits is performed with the SWAP gates, but this can be replaced by relabeling the qubits like the output of the `RECTANGLE` SBox (i.e., $(y_0, y_1, y_2, y_3) \leftarrow (x_2, x_1, x_3, x_0)$). We relabel all the qubits so that none of the quantum resources are used for these qubit rotation operations.

4.1.4 Key Schedule

`RECTANGLE` supports two versions with 80-bit key and 128-bit key. In `RECTANGLE`-80, the 80-qubit key K is

Algorithm 2: Quantum implementation of `RECTANGLE` SBox (`65CA1E79B03D8F42`, using LIGHTER-R)

Input: (x_3, x_2, x_1, x_0) to the SBox
Output: (y_3, y_2, y_1, y_0) from the SBox
1: $x_0 \leftarrow \text{Toffoli}(x_1, x_3, x_0)$
2: $x_3 \leftarrow \text{Toffoli}(x_0, x_1, x_3)$
3: $x_0 \leftarrow \text{CNOT}(x_3, x_0)$
4: $x_2 \leftarrow \text{CNOT}(x_0, x_2)$
5: $x_0 \leftarrow \text{CNOT}(x_1, x_0)$
6: $x_3 \leftarrow \text{X}(x_3)$
7: $x_3 \leftarrow \text{CNOT}(x_2, x_3)$
8: $x_1 \leftarrow \text{CNOT}(x_3, x_1)$
9: $x_3 \leftarrow \text{Toffoli}(x_0, x_2, x_3)$
10: $x_0 \leftarrow \text{Toffoli}(x_2, x_3, x_0)$
11: $(y_0, y_1, y_2, y_3) \leftarrow (x_2, x_1, x_3, x_0)$
12: **return** (y_3, y_2, y_1, y_0)

arranged in five 16-qubit rows, $K = Row_4 \| Row_3 \| Row_2 \| Row_1 \| Row_0$. In the first of the `Key Schedule`, the quantum `RECTANGLE` SBox of Algorithm 2 is utilised for each column. Then, the Feistel transformation is applied. CNOT gates are used for the XOR operation between rows (i.e., $Row'_0 = (Row_0 \lll 8) \oplus Row_1, Row'_3 = (Row_3 \lll 12) \oplus Row_4$). At this time, since Row_0 and Row_3 must be used as the result later (Row'_2, Row'_4), we store the XOR result in Row_1 and Row_4 for optimisation. Qubit rotations are used together, qubits are input to the CNOT gates according to the qubit index after rotation(e.g., $\text{CNOT}(k_8, k_{16})$). Swap operations between rows are also relabeled, so quantum resources are not used. Lastly, the round constants RC_i generated by the 5-bit LFSR are XORed to the key. Since all of the round constants RC_i can be known in advance, we optimised using X gates according to the round constant and denoted as $AddRoundConsant(RC_i, K)$. For example, when $i = 4$, $RC_4 = 12$. In 5-bit RC_4 $(rc_4, rc_3, rc_2, rc_1, rc_0)$, the bits with value 1 are rc_4 and rc_1. Therefore, XORing RC_4 to K can be replaced by performing X gate on k_1 and k_4 qubits, and X gate consumes less resources than CNOT gate. The quantum circuit design for `Key Schedule` of `RECTANGLE`-80 is described in Algorithm 3.

In `RECTANGLE`-128, the 128-qubit key K is arranged in four 32-qubit rows, $K = Row_3 \| Row_2 \| Row_1 \| Row_0$. The `Key Schedule` subroutine of `RECTANGLE`-128 is similar to `RECTANGLE`-80 and is shown in Algorithm 4.

4.2 `KNOT-PERMUTATION`

`KNOT-PERMUTATION` consists of the following subroutine (characterised by parameter, b), $\texttt{AddRoundConstant}_b$, $\texttt{SubColumn}_b$ and $\texttt{ShiftRow}_b$. Since the structure of `KNOT-PERMUTATION` is similar to `RECTANGLE`, the design of quantum circuit is also similar. $\texttt{AddRoundConstant}_b$,

Algorithm 3: `Key Schedule` of `RECTANGLE-80` as a quantum circuit

Input: 80-qubit key K $(k_{79}, \ldots, k_0)$
Output: 64-qubit round key RK $(rk_{63}, \ldots, rk_0)$

$k_{48}, k_{32}, k_{16}, k_0 \leftarrow \text{SBox}\ (k_{48}, k_{32}, k_{16}, k_0)$
$k_{49}, k_{33}, k_{17}, k_1 \leftarrow \text{SBox}\ (k_{49}, k_{33}, k_{17}, k_1)$
$k_{50}, k_{34}, k_{18}, k_2 \leftarrow \text{SBox}\ (k_{50}, k_{34}, k_{18}, k_2)$
$k_{51}, k_{35}, k_{19}, k_3 \leftarrow \text{SBox}\ (k_{51}, k_{35}, k_{19}, k_3)$
$Row_1 \leftarrow \text{CNOT}((Row_0 \lll 8), Row_1)$ ▷ Row unit operation (16-qubit)
$Row'_0 \leftarrow Row_1$ ▷ Row unit operation (16-qubit)
$Row_4 \leftarrow \text{CNOT}((Row_3 \lll 12), Row_4)$
$Row'_3 \leftarrow Row_4$
$Row'_1 \leftarrow Row_2$
$Row'_2 \leftarrow Row_3$
$Row'_4 \leftarrow Row_0$
▷ `AddRoundConsant`(RC_i, K)
$RK \leftarrow Row'_3 \| Row'_2 \| Row'_1 \| Row'_0$
return $RK(rk_{63}, \ldots, rk_0)$

Algorithm 4: `Key Schedule` of `RECTANGLE-128` as a quantum circuit

Input: 128-qubit key K $(k_{127}, \ldots, k_0)$
Output: 64-qubit round key RK $(rk_{63}, \ldots, rk_0)$

$k_{96}, k_{64}, k_{32}, k_0 \leftarrow \text{SBox}\ (k_{96}, k_{64}, k_{32}, k_0)$
$k_{97}, k_{65}, k_{33}, k_1 \leftarrow \text{SBox}\ (k_{97}, k_{65}, k_{33}, k_1)$
$k_{98}, k_{66}, k_{34}, k_2 \leftarrow \text{SBox}\ (k_{98}, k_{66}, k_{34}, k_2)$
$k_{99}, k_{67}, k_{35}, k_3 \leftarrow \text{SBox}\ (k_{99}, k_{67}, k_{35}, k_3)$
$k_{100}, k_{68}, k_{36}, k_4 \leftarrow \text{SBox}\ (k_{100}, k_{68}, k_{36}, k_4)$
$k_{101}, k_{69}, k_{37}, k_5 \leftarrow \text{SBox}\ (k_{101}, k_{69}, k_{37}, k_5)$
$k_{102}, k_{70}, k_{38}, k_6 \leftarrow \text{SBox}\ (k_{102}, k_{70}, k_{38}, k_6)$
$k_{103}, k_{71}, k_{39}, k_7 \leftarrow \text{SBox}\ (k_{103}, k_{71}, k_{39}, k_7)$
$Row_1 \leftarrow \text{CNOT}((Row_0 \lll 8), Row_1)$ ▷ Row unit operation (32-qubit)
$Row'_0 \leftarrow Row_1$ ▷ Row unit operation (32-qubit)
$Row_3 \leftarrow \text{CNOT}((Row_2 \lll 16), Row_3)$
$Row'_2 \leftarrow Row_3$
$Row'_1 \leftarrow Row_2$
$Row'_3 \leftarrow Row_0$
▷ `AddRoundConsant`(RC_i, K)
$RK \leftarrow$
$Row'_3(k_{111}, \ldots, k_{96}) \| Row'_2(k_{79}, \ldots, k_{64})$
$\| Row'_1(k_{47}, \ldots, k_{32}) \| Row'_0(k_{15}, \ldots, k_0)$
return $RK(rk_{63}, \ldots, rk_0)$

`SubColumn`$_b$ and `ShiftRow`$_b$ are optimised in quantun circuits as follows.

In `AddRoundConstant`$_b$, round constants RC_i generated by $d(6, 7, 8)$-bit LFSR are XORed to the first d-qubit of B. Since we know the RC_i value used in each round in advance, we optimise it by performing X gates according to RC_i.

In `SubColumn`, B consists of four $\frac{b}{4}$-qubit rows (i.e., $B = Row_3 \| Row_2 \| Row_1 \| Row_0$) and a 4-qubit SBox is applied to each column. We implemented SBox as a quantum circuit using LIGHTER-R (with the MCT_gc library; and the cost incurred is 4 Toffoli gates, 3 CNOT gates, and 1 NOT gate). As a result, no additional qubit is allocated, and low-cost quantum resources are used. Details of the SBox quantum circuit implementation is shown in Algorithm 5 and Figure 7.

Algorithm 5: Quantum implementation of `KNOT` SBox (`40A7BE1D9F6852C3`, using LIGHTER-R)

Input: (x_3, x_2, x_1, x_0) to the SBox
Output: (y_3, y_2, y_1, y_0) from the SBox

$x_0 \leftarrow \text{X}(x_0)$
$x_2 \leftarrow \text{Toffoli}(x_0, x_1, x_2)$
$x_0 \leftarrow \text{Toffoli}(x_1, x_2, x_0)$
$x_3 \leftarrow \text{CNOT}(x_2, x_3)$
$x_1 \leftarrow \text{CNOT}(x_3, x_1)$
$x_0 \leftarrow \text{CNOT}(x_1, x_0)$
$x_1 \leftarrow \text{Toffoli}(x_0, x_2, x_1)$
$x_2 \leftarrow \text{Toffoli}(x_0, x_1, x_2)$
$(y_0, y_1, y_2, y_3) \leftarrow (x_1, x_2, x_0, x_3)$
return (y_3, y_2, y_1, y_0)

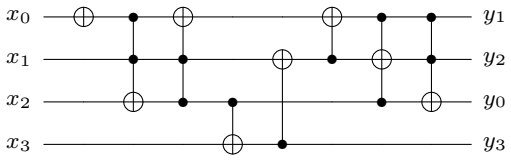


Fig. 7: Quantum implementation of `KNOT` SBox (`40A7BE1D9F6852C3`, using LIGHTER-R)

In `ShifRow`, for B ($B = Row_3 \| Row_2 \| Row_1 \| Row_0$), (c_1, c_2, c_3) qubit left rotation operation is performed on each (Row_1, Row_2, Row_3). The values of (c_1, c_2, c_3) depend on the width of b. No matter what rotation operations are performed in `ShiftRow`, we do not use any quantum resources as we relabel the qubits without using SWAP gates as in `ShiftRow` of `RECTANGLE`. Our optimised `KNOT-PERMUTATION` quantum circuit implementation is utilised as a round transformation of `KNOT-AEAD` in Section 4.3.

4.3 `KNOT-AEAD`

We implemented all versions of `KNOT-AEAD` as a quantum circuit. The previously mentioned `KNOT-PERMUTATION` is utilised, and all other operations are optimised in quantum circuits. The proposed `KNOT-AEAD` quantum circuits are optimised in terms of qubits, and the use of quantum gates is also minimised. Only qubits for input and output are allocated, and no additional qubit is used during the operation.

4.3.1 Padding

Before authenticated encryption, `Padding` is performed on associated data AD and plaintext P, except when AD and P are $\emptyset$. First, a single 1 is added as the most significant bit and filled with the minimum number of 0's to satisfy multiples of the r-bit block. However, we optimised `Padding` by not using quantum resources at all. That is, we do not perform `Padding`. Since the padded P and AD are used only as intermediate values for the result value in authenticated encryption, we assume that the original P and AD (i.e., not padded) are padded and perform authenticated encryption. This is described in detail in Sections `Processing Associated Data` and `Encryption`.

4.3.2 Initialisation

The key K and nonce N are initialised differently according to parameters. In `KNOT-AEAD` (128,384,192), it is initialised to $S = (0^{128}\|K\|N) \oplus (1\|0^{383})$. Since S is continuously updated and used as the output; 128 additional qubits for 0^{128} are allocated for this purpose. On the other hand, the other three parameters are initialised to $S = (K\|N)$. Because K and N are just attached, quantum resources are not used. Lastly, the quantum version of `KNOT-PERMUTATION` $p_b[nr_0]$ is performed on the initial state S (i.e., $S \leftarrow p_b[nr_0](S)$).

4.3.3 Processing Associated Data

As mentioned earlier, we minimise the use of qubits by using the input as it is without performing padding for the associated data AD. Suppose AD is padded to be AD_i $(i = 0, \ldots, u-1)$ composed of u blocks. Only the last block AD_{u-1} is padded with a single 1 and 0's. Therefore, if it is not the last block, the following general process is performed. Block unit operation($S \leftarrow S \oplus AD_i$) is performed using CNOT gates, and the `KNOT-PERMUTATION` $p_b[nr]$ is performed on S. In the last block, block unit operation is not performed. In AD_{u-1}, only the unpadded input AD_{u-1} is XORed to S using CNOT gates.

Originally, in `Padding`, input AD is appended with a single 1 and 0s to fill the r-bit block. The XOR operation of 0s does not change anything, so this does not need to be implemented. Only a single X gate is used for a single 1 XOR operation and `KNOT-PERMUTATION` $p_b[nr]$ is performed. Lastly, 1 is XORed by performing a single X gate on the last qubit of S. Through these, `Processing Associated Data` is implemented as a quantum circuit with minimal quantum gates and no additional qubits. Details of the quantum circuit implementation are shown in Algorithm 6.

Algorithm 6: `Processing Associated Data` of `KNOT-AEAD` as a quantum circuit

Input: $S = (s_{b-1}, \ldots, s_0)$, x-qubit AD_i $(i = 0, \ldots, u-1)$ ▷ Assuming padded
Output: $S = (s_{b-1}, \ldots, s_0)$
$i \leftarrow 0$
while $i \neq (u-1)$ **do**
  $S \leftarrow \mathrm{CNOT}(AD_i, S)$ ▷ Block unit operation (r-qubit)
  $S \leftarrow p_b[nr](S)$
  $i \leftarrow i + 1$
$x = x \bmod r$ ▷ Last block
for $j = 0$ to $x - 1$ **do**
  $s_j \leftarrow \mathrm{CNOT}(AD_{i(j)}, s_j)$
$s_x \leftarrow \mathrm{X}(s_x)$
$S \leftarrow p_b[nr](S)$
$s_{b-1} \leftarrow \mathrm{X}(s_{b-1})$
return $S(s_{b-1}, \ldots, s_0)$

4.3.4 Encryption

In `Encryption`, the same techniques used in `Processing Associated Data` are utilised. In the last block P_{v-1}, only the non-padded input P_{v-1} is XORed to S using CNOT gates. For the padded part, no additional qubits are allocated and only a single X gate is used. The difference from `Processing Associated Data` is that `KNOT-PERMUTATION` $p_b[nr]$ is not performed in the last block, and y qubits of the same size as P_i are newly allocated to store the ciphertext C_i. The implementation of quantum circuit for `Encryption` is described in detail in Algorithm 7.

Algorithm 7: `Encryption` of `KNOT-AEAD` as a quantum circuit

Input: $S = (s_{b-1}, \ldots, S_0)$, y-qubit P_i $(i = 0, \ldots, v-1)$ ▷ Assuming padded
Output: Ciphertext C_i $(i = 0, \ldots, v-1)$
$i = 0$
while $i \neq v - 1$ **do**
  $S \leftarrow \mathrm{CNOT}(P_i, S)$ ▷ Block unit operation (r-qubit)
  $C_i \leftarrow \mathrm{CNOT}(S, C_i)$ ▷ Generate ciphertext
  $S \leftarrow p_b[nr](S)$
  $i \leftarrow i + 1$
$y = y \bmod r$ ▷ Last block
for $j = 0$ to $y - 1$ **do**
  $s_j \leftarrow \mathrm{CNOT}(P_{i(j)}, s_j)$
  $C_{i(j)} \leftarrow \mathrm{CNOT}(s_j, C_{i(j)})$
$s_y \leftarrow \mathrm{X}(s_y)$
return C_i $(i = 0, \ldots, v-1)$

4.3.5 Finalisation

In `Finalisation`, the tag T is generated and appended to the ciphertext C to generate the output $T||C$. A `KNOT-PERMUTATION` $pb_b[nr_f]$ is performed on S, and the

lower k-qubits are used as tag T:

$$S = pb_b[nr_f](S),$$
$$T = s_{k-1}||\ldots||s_0.$$

5 Resource Estimates/Evaluation

5.1 Resources for Implementation of RECTANGLE and KNOT-AEAD

We used ProjectQ, a quantum programming tool provided by IBM, to implement RECTANGLE and KNOT-AEAD as quantum circuits. We simulated RECTANGLE and KNOT-AEAD quantum circuits using the ClassicalSimulator and estimated the used quantum resources using the ResourceCounter. Quantum resources required to implement quantum circuits of RECTANGLE and other block ciphers are shown in Table 2. Based on the quantum resources in Table 2, we compare and check the important factors in implementing quantum circuits.

The number of qubits is an important factor in quantum circuit implementation. Since large-scale qubit quantum computers have not yet been developed, the number of qubits used in quantum circuits is related to the timing at which they actually operate in quantum computers. In RECTANGLE, only the qubits for the initial input and key are allocated and no more additional qubits are allocated.

For a block cipher, particularly which are based on the Substitution Permutation Network (SPN) family, how the SBox is implemented plays an important role. Since the input is in a superposition state, an inefficient SBox quantum circuit must be implemented that checks all inputs one by one and allocates additional qubits for the output. We optimise the RECTANGLE SBox operation using LIGHTER-R rather than implementing it naïvely from its coordinate functions. With this, input qubits become output qubits and the cost of quantum gates was also minimized. In Key Schedule, no additional qubits are allocated by storing the result of the operation in rows that are not used as result values when performing an operation between rows. As a result, an ideal number of qubits were used in the quantum circuits of RECTANGLE.

Many operations such as AddRoundkey, SubColumn, and Key Schedule of RECTANGLE are performed in parallel. As a result, the depth of the quantum circuit is 266, the lowest among the ciphers given in Table 2. The depths of SPECK and CHAM are not indicated, but those are much higher than the depth of SPECK. As a result of measuring the resources from the source codes of CHAM [11] and SPECK [13], we get the depth of CHAM-64/128 is 7807 and that of SPECK-64/128 is 8323. In the latest implementation of SPECK [2], it is reduced to 4239, which is still high. Depth represents the longest path from beginning till the end of the circuit, which is an important factor related to execution time [6].

In KNOT-AEAD except KNOT-AEAD (128, 384, 192), a fixed length key K and nonce N are used, and K and N are appended to use the internal state S (i.e., $S = K||N$). Therefore, $K_{size} + N_{size}$ qubits are allocated. In KNOT-AEAD (128, 384, 192), since 128 additional 0's are appended (i.e., $S = 0^{128}||K||N$), $K_{size}+N_{size}+128$ qubits are allocated. The qubits of S are fixed, and the final number of qubits depends on the length of the associated data AD and plaintext P. Originally, Padding, in which additional bits are appended to AD and P, is performed; but we do not allocate any qubits for Padding. Finally, the optimal number of qubits, $|S|+|AD|+2\cdot|P|$, is used in KNOT-AEAD quantum circuits. The depth of the KNOT quantum circuits and the required quantum gates depend on the block lengths of the padded AD and P (i.e., u and v). This is because the number of KNOT-PERMUTATION depends on the block length. KNOT-PERMUTATION occupies the most proportion in KNOT-AEAD. Since KNOT-PERMUTATION is very similar to RECTANGLE, the same optimisation technique is applied. In other words, quantum resources also differ according to $|AD|$ and $|P|$, but are ignored in Table 3, because they are small changes compared to u and v.

A summary of results on KNOT-AEAD is given in Table 3. For the depth figures in Table 3, we consider AD and P are $\emptyset$.

5.2 Resources for Grover's Search Oracle

In Grover's search, the main module is oracle, and the core of oracle in this key search is KNOT-AEAD quantum circuit. We estimate the attack resource based on the previously optimized implementation of KNOT-AEAD. For standardized evaluation of all parameters, encryption is based when associated data AD and plaintext P are 32 bits, and the resource estimates with the X/NOT, CNOT and Toffoli gates (NCT gates) are as shown in Table 4.

We analyze KNOT-AEAD with the NCT gates, and analyze Grover's oracle and key search at the T + Clifford gate level. X and CNOT gates count as one Clifford gate, and Toffoli gates count as T + Clifford gates. Toffoli gates can be decomposed in several ways. We choose the method of [1], the Toffoli gate is decomposed into 7 T gates + 8 Clifford gates, and the T depth is 4. In oracle, KNOT-AEAD is executed twice due to encryption + reverse, and a single multi-controlled NOT gate is used

Table 2: Quantum resources required for few lightweight block ciphers

Cipher	Qubits	Toffoli gates	CNOT gates	X gates	Depth
RECTANGLE-80 (Ours)	144	2,000	4,964	567	266
RECTANGLE-128 (Ours)	192	2,400	5,688	668	266
SIMON-64/128 [3]	192	1,408	7,396	1,216	2,643
SPECK-64/128 [13]	193	3,286	9,238	57	N/A
GIFT-64/128 [12]	192	1,792	1,792	3,261	308
CHAM-64/128 [11]	196	2,400	12,285	240	N/A

Table 3: Quantum resources required for KNOT-AEAD implementation

Cipher	Qubits	Toffoli gates	CNOT gates	X gates	Depth
KNOT-AEAD (128, 256, 64)	$256 + \lvert AD\rvert + 2 \cdot \lvert P\rvert$	$21,044 + 7,032 \cdot u + 7,032 \cdot (v-1)$	$16,128 + 5,376 \cdot u + 5,376 \cdot (v-1)$	$5,146 + 1,723 \cdot u + 1,723 \cdot (v-1)$	$672 + 224 \cdot u + 224 \cdot (v-1)$
KNOT-AEAD (128, 384, 192)	$384 + \lvert AD\rvert + 2 \cdot \lvert P\rvert$	$40,846 + 10,620 \cdot u + 10,620 \cdot (v-1)$	$31,104 + 8,072 \cdot u + 8,072 \cdot (v-1)$	$10,056 + 2,621 \cdot u + 2,621 \cdot (v-1)$	$864 + 224 \cdot u + 224 \cdot (v-1)$
KNOT-AEAD (192, 384, 96)	$384 + \lvert AD\rvert + 2 \cdot \lvert P\rvert$	$45,362 + 15,146 \cdot u + 15,146 \cdot (v-1)$	$34,560 + 11,520 \cdot u + 11,520 \cdot (v-1)$	$11,161 + 3,732 \cdot u + 3,732 \cdot (v-1)$	$960 + 320 \cdot u + 320 \cdot (v-1)$
KNOT-AEAD (256, 512, 128)	$512 + \lvert AD\rvert + 2 \cdot \lvert P\rvert$	$78,862 + 26,304 \cdot u + 26,304 \cdot (v-1)$	$59,904 + 19,968 \cdot u + 19,968 \cdot (v-1)$	$19,463 + 6,495 \cdot u + 6,495 \cdot (v-1)$	$1,248 + 416 \cdot u + 416 \cdot (v-1)$

u = length of AD block (after padding), v = length of P block (after padding)

to compare the generated ciphertext with the known ciphertext. Therefore, in oracle, Table 4 $\times$ 2 resources are used, excluding qubits. When the ciphertext length is l-bit, l multi-controlled NOT gate is used and it is decomposed into $32 \cdot l - 84$ T gates [21]. In addition, in a multi controlled NOT gate, one target qubit is allocated, which is flipped when the ciphertext matches. As a result, the resources required for oracle are shown in Table 5.

5.3 Resources for Grover's Key Search

KNOT-AEAD operates with plaintext P, associated data AD, nonce N, and key K as inputs. In [4], the authors estimated the Grover's key search resources for GRAIN-128-AEAD and TINYJAMBU, assuming the plaintext, associated data, and nonce are known. We also assume that the plaintext P, associated data AD, and nonce N are known and estimate the resource for Grover key search. In this case, Grover search is performed on key K in superposition state, and the number of iterations depends on the size of K (i.e. k). The Grover search algorithm is well known for reducing the complexity of N to $\sqrt{N}$. However, in [7], the authors suggested that the optimal number of iterations for N is $\lfloor \frac{\pi}{4}\sqrt{N} \rfloor$ through a tight analysis of the Grover search algorithm. Therefore, in case of 128-bit key, oracle is repeated $\lfloor \frac{\pi}{4} \cdot 2^{64} \rfloor$, in case of 192-bit key and 256-bit key, it is repeated $\lfloor \frac{\pi}{4} \cdot 2^{96} \rfloor$, $\lfloor \frac{\pi}{4} \cdot 2^{128} \rfloor$. We sufficiently increase the probability of measuring the solution key by iterating the oracle and diffusion operator. Finally, the resource for Grover's key search is estimated as shown in Table 5 $\times$ $\lfloor \frac{\pi}{4} \cdot 2^{\frac{k}{2}} \rfloor$, which is shown in Table 6. Grover's search algorithm iterates the oracle and diffusion operators as a set, but we ignore the cost of the diffusion operator. Because the diffusion operator has a standardized structure, there is no special technique to implement. Also, since the oracle accounts for most of the cost, diffusion operator is usually ignored when estimating the cost [9].

5.4 Security Strength Estimated by NIST

It has been established that Grover's search algorithm reduces the security of symmetric key cryptosystems in half. What we need to consider is the necessary cost. If a huge cost is required to apply the Grover search algorithm, it can be judged that it is resistant to attacks by quantum computers. For this reason, NIST has defined the following security requirements based on resources for AES [9], which are listed in order of security strength [20].

- Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit key (e.g., AES-128)

Table 4: Quantum resources required for KNOT-AEAD implementation

Cipher	Qubits	Toffoli gates	CNOT gates	X gates	Depth
KNOT-AEAD (128, 256, 64)	352	28,074	21,600	6,875	899
KNOT-AEAD (128, 384, 192)	480	51,464	39,264	12,683	1,091
KNOT-AEAD (192, 384, 96)	480	60,506	46,176	14,899	1,283
KNOT-AEAD (256, 512, 128)	608	105,164	79,968	25,964	1,667

AD and P are of 32-bits

Table 5: Quantum resources required for KNOT-AEAD to run Grover's oracle

Cipher	Qubits	Clifford gates	T gates	T depth	Depth
KNOT-AEAD (128, 256, 64)	353	506,134	398,072	224,592	1,799
KNOT-AEAD (128, 384, 192)	481	927,318	725,532	411,712	2,183
KNOT-AEAD (192, 384, 96)	481	1,090,246	854,168	484,048	2,567
KNOT-AEAD (256, 512, 128)	609	1,894,488	1,481,428	841,312	3,335

Table 6: Quantum resources required for KNOT-AEAD to run Grover's key search

Cipher	Qubits	Clifford gates	T gates	T depth	Depth	Total gates
KNOT-AEAD (128, 256, 64)	353	$1.516 \cdot 2^{82}$	$1.193 \cdot 2^{82}$	$1.346 \cdot 2^{81}$	$1.378 \cdot 2^{74}$	$1.354 \cdot 2^{83}$
KNOT-AEAD (128, 384, 192)	481	$1.389 \cdot 2^{83}$	$1.087 \cdot 2^{83}$	$1.234 \cdot 2^{82}$	$1.673 \cdot 2^{74}$	$1.238 \cdot 2^{84}$
KNOT-AEAD (192, 384, 96)	481	$1.633 \cdot 2^{115}$	$1.279 \cdot 2^{115}$	$1.450 \cdot 2^{114}$	$1.968 \cdot 2^{106}$	$1.456 \cdot 2^{116}$
KNOT-AEAD (256, 512, 128)	609	$1.419 \cdot 2^{148}$	$1.109 \cdot 2^{148}$	$1.260 \cdot 2^{147}$	$1.278 \cdot 2^{139}$	$1.264 \cdot 2^{149}$

- Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 192-bit key (e.g., AES-192)
- Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 256-bit key (e.g., AES-256)

Based on [9], NIST conservatively estimate D (total gates $\times$ depth) as 2^{170}, 2^{233} and 2^{298} for AES-128, 196 and 256. Now we compare KNOT-AEAD with the security strength of NIST, as shown in Table 7. As a result, resources for Grover key search for KNOT-AEAD using 128-bit key, 196-bit key, and 256-bit key are lower than the security strengths AES-128, AES-192 and AES-256 suggested by NIST.

For estimation of resources, both the plaintext P and associated data AD are assumed to be 32-bit; but even if the size is increased to 128-bit and 256-bit, it does not satisfy the security strength of NIST (i.e., less than 2^{170}, 2^{233} and 2^{298}). The effect of increasing the size of P and AD is not significant. We believe that in order to satisfy the security strengths of 2^{170}, 2^{233} and 2^{298}, it is necessary to increase the parameters of the permutation based on RECTANGLE used in KNOT-AEAD.

6 Conclusion and Future Work

In this work, we implement two lightweight ciphers (namely, the block cipher RECTANGLE [23] and the AEAD KNOT [24]) as quantum circuits, using the ProjectQ library. This is the first such (public) implementation for the ciphers, and among the first implementation for an AEAD. Multiple optimisations are used to keep the cost as minimal as possible. Further, we estimate the cost of the circuit that would be required to run the Grover's search algorithm [10]. Thus, our work constitutes the basis for analysis of the two target ciphers by a quantum computer.

In the KNOT implementations, it is possible to reduce the depth slightly with a sharp increase in the qubit complexity, but we ignore this as it appears to be highly inefficient. Instead, we attempt to reduce the qubit complexity while keeping the depth within a tolerable bound. In other words, the KNOT implementations presented here do not let the depth increase unbounded to reduce the qubit complexity.

We note two potential research directions that could be interesting for the future researchers to pursue. First, other ciphers (such as AEADs from the NIST LWC project or the CAESAR[6] project) can be analysed in an analogous way. In the process, one may be interested in a generalised framework to compare multiple ciphers;

[6] https://competitions.cr.yp.to/caesar.html.

Table 7: Comparison of NIST's security strength of KNOT-AEAD with AES

Cipher	Total gates	Depth	D	NIST security
KNOT-AEAD $(128, 256, 64)$	$1.354 \cdot 2^{83}$	$1.378 \cdot 2^{74}$	$1.866 \cdot 2^{157}$	2^{170} (AES-128)
KNOT-AEAD $(128, 384, 192)$	$1.238 \cdot 2^{84}$	$1.673 \cdot 2^{74}$	$1.036 \cdot 2^{159}$	
KNOT-AEAD $(192, 384, 96)$	$1.456 \cdot 2^{116}$	$1.968 \cdot 2^{106}$	$1.433 \cdot 2^{223}$	2^{233} (AES-192)
KNOT-AEAD $(256, 512, 128)$	$1.264 \cdot 2^{149}$	$1.278 \cdot 2^{139}$	$1.615 \cdot 2^{288}$	2^{298} (AES-256)

D = Total gates $\times$ Total depth

for example, as done in context of FPGA benchmarking [18]. Second, tools can be designed/improved for (more) efficient implementation. For example, support for bigger SBoxes (such as, 8×8) or non-zero cost for the SWAP gate can be incorporated in LIGHTER-R.

References

1. Amy, M., Maslov, D., Mosca, M., Roetteler, M.: A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **32**(6) (Jun 2013) 818–830 10
2. Anand, R., Maitra, A., Mukhopadhyay, S.: Evaluation of quantum cryptanalysis on speck. In Bhargavan, K., Oswald, E., Prabhakaran, M., eds.: Progress in Cryptology – INDOCRYPT 2020, Cham, Springer International Publishing (2020) 395–413 10
3. Anand, R., Maitra, A., Mukhopadhyay, S.: Grover on SIMON. Quantum Information Processing **19**(9) (Sep 2020) 2, 11
4. Anand, R., Maitra, S., Maitra, A., Mukherjee, C.S., Mukhopadhyay, S.: Resource estimation of grovers-kind quantum cryptanalysis against fsr based symmetric ciphers. Cryptology ePrint Archive, Report 2020/1438 (2020) https://eprint.iacr.org/2020/1438. 2, 11
5. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**(6) (November 1973) 525–532 1
6. Bhattacharjee, D., Chattopadhyay, A.: Depth-optimal quantum circuit placement for arbitrary topologies. CoRR **abs/1703.08540** (2017) 10
7. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik **46**(4-5) (Jun 1998) 493–505 11
8. Dasu, V.A., Baksi, A., Sarkar, S., Chattopadhyay, A.: LIGHTER-R: optimized reversible circuit implementation for sboxes. In: 32nd IEEE International System-on-Chip Conference, SOCC 2019, Singapore, September 3-6, 2019. (2019) 260–265 2, 7
9. Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying grover's algorithm to AES: quantum resource estimates. CoRR **abs/1512.04965** (2015) 2, 11, 12
10. Grover, L.K.: A fast quantum mechanical algorithm for database search. In Miller, G.L., ed.: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996, ACM (1996) 212–219 1, 3, 12
11. Jang, K., Choi, S., Kwon, H., Kim, H., Park, J., Seo, H.: Grover on Korean block ciphers. Applied Sciences **10**(18) (2020) 6407 10, 11
12. Jang, K., Kim, H., Eum, S., Seo, H.: Grover on GIFT. Cryptology ePrint Archive, Report 2020/1405 (2020) https://eprint.iacr.org/2020/1405. 2, 11
13. 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. 2, 10, 11
14. 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) 2
15. Jaques, S., Naehrig, M., Roetteler, M., Virdia, F.: Implementing grover oracles for quantum key search on aes and lowmc. Cryptology ePrint Archive, Report 2019/1146 (2019) https://eprint.iacr.org/2019/1146. 2
16. Langenberg, B., Pham, H., Steinwandt, R.: Reducing the cost of implementing aes as a quantum circuit. Cryptology ePrint Archive, Report 2019/854 (2019) https://eprint.iacr.org/2019/854. 2
17. Loepp, S., Wootters, W.K.: Protecting Information: From Classical Error Correction to Quantum Cryptography. Cambridge University Press (2012) 1
18. Mohajerani, K., Haeussler, R., Nagpal, R., Farahmand, F., Abdulgadir, A., Kaps, J.P., Gaj, K.: Fpga benchmarking of round 2 candidates in the nist lightweight cryptography standardization process: Methodology, metrics, tools, and results. Cryptology ePrint Archive, Report 2020/1207 (2020) https://eprint.iacr.org/2020/1207. 13
19. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. University Press, Cambridge (2010) 1
20. 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. 11
21. Wiebe, N., Roetteler, M.: Quantum arithmetic and numerical analysis using repeat-until-success circuits (2014) 11
22. Xiang, Z., Zeng, X., Lin, D., Bao, Z., Zhang, S.: Optimizing implementations of linear layers. Cryptology ePrint Archive, Report 2020/903 (2020) https://eprint.iacr.org/2020/903. 2
23. Zhang, W., Bao, Z., Lin, D., Rijmen, V., Yang, B., Verbauwhede, I.: RECTANGLE: A bit-slice lightweight block cipher suitable for multiple platforms. Cryptology ePrint Archive, Report 2014/084 (2014) https://eprint.iacr.org/2014/084. 2, 3, 12
24. Zhang, W., Ding, T., Yang, B., Bao, Z., Xiang, Z., Ji, F., Zhao, X.: Knot: Algorithm specifications and supporting document. Submission to NIST (2019) https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/knot-spec-round.pdf. 2, 4, 6, 12
25. Zou, J., Wei, Z., Sun, S., Liu, X., Wu, W.: Quantum circuit implementations of aes with fewer qubits. In Moriai, S., Wang, H., eds.: Advances in Cryptology – ASIACRYPT 2020, Cham, Springer International Publishing (2020) 697–726 2