Quantum Implementation of ASCON Linear Layer

Soham Roy[1], Anubhab Baksi[2], and Anupam Chattopadhyay[2]

[1] Indian Institute of Technology, Madras, India
[2] Nanyang Technological University, Singapore

sroy@smail.iitm.ac.in, anubhab001@e.ntu.edu.sg, anupam@ntu.edu.sg

Abstract. In this paper, we show an in-place implementation of the ASCON linear layer. An in-place implementation is important in the context of quantum computing, we expect our work will be useful in quantum implementation of ASCON. In order to get the implementation, we first write the ASCON linear layer as a binary matrix; then apply two legacy algorithms (Gauss-Jordan elimination and PLU factorization) as well as our modified version of Xiang et al.'s algorithm/source-code (published in ToSC/FSE'20). Our in-place implementation takes 1595 CNOT gates and 119 quantum depth; and this is the first in-place implementation of the ASCON linear layer, to the best of our knowledge.

Keywords: ASCON · Quantum Implementation · Linear Layer · In-place Implementation

1 Introduction

ASCON [DEMS19] is a lightweight cryptographic primitive that provides confidentiality, integrity, and authenticity to data transmission. It is recently selected as the winner of the LWC project by NIST[1].

In recent years, quantum computing has become an active research area for improving the performance and security of cryptographic primitives. The potential benefits of quantum computing include increased processing speeds and the ability to perform certain operations more efficiently than classical computers. Therefore, exploring the potential of quantum computing to improve the performance of ASCON is of significant interest to the cryptography community. With the increasing popularity of quantum computing, an efficient implementation of ciphers on quantum logic is becoming increasingly important. Therefore, it comes as no surprise that the researchers have been working on finding improved quantum implementation for the ciphers (see, e.g., [JBB+22, JBK+22a, JSK+22]).

Contribution

In this work, we present possibly the first-ever in-place implementation of the ASCON linear layer. The related implementations are available as an open-source project[2]. The linear layer is described in terms of rotation and XOR of five 64-bit registers. Thus, it can be equivalently expressed as a 320×320 binary matrix.

To the best of our finding, the only work to deal with the quantum implementation of ASCON is done in [LJS+22]. However, this work does not implement the linear layer explicitly (implements the state update function as one module). Internally, the naïve quantum implementation of the linear layer (Section 3) is invoked, which results in a out-of-place implementation with doubled qubit count.

When in comes to in-place implementation, we know about the PLU factorization, Gauss-Jordan elimination and a recent development by XZLBZ [XZL+20]. All the methods have been used in some capacity in the literature (Section 3). In total, we show 3 in-place implementations of the ASCON linear layer.

In this process, we would like to note that the authors of [XZL+20] explicitly state that their algorithm/source-code works efficiently up to 32×32 binary matrix. Notably, the task of finding good in-place implementation of 64×64 binary matrices and beyond is mostly kept out-of-scope. We contribute to the algorithm as well as the source-code of XZLBZ [XZL+20] so that now it can work even with the ASCON binary matrix (which is of dimension 320×320).

As noted in Section 3, the naïve implementation in quantum doubles the number of qubits and requires a total of (number of qubits + number of XOR operation) CNOT gates in general. When the qubit count $\times$ CNOT count metric is considered, the implementation reported by us is 16.93% cheaper. The benchmarks corresponding to these implementations, along with two more in-place implementations (obtained by applying the Gauss-Jordan elimination and PLU factorization) are given in Section 4.

[1] https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon
[2] https://github.com/sohamroy19/ascon-linear-layer/

2 ASCON Linear Layer

As defined in [DEMS19], the linear layer p_L provides diffusion within each 64-bit register word x_i. It applies a linear function $\Sigma_i(x_i)$ to each word x_i, defined as follows ('$\ggg$' indicates right rotation):

$$\begin{aligned}
x_0 &\leftarrow \Sigma_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)\\
x_1 &\leftarrow \Sigma_1(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)\\
x_2 &\leftarrow \Sigma_2(x_2) = x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6)\\
x_3 &\leftarrow \Sigma_3(x_3) = x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17)\\
x_4 &\leftarrow \Sigma_4(x_4) = x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)
\end{aligned}$$

When simplified, this can be written as a binary non-singular matrix of dimension 320×320. The Hamming weight of each row (and also column) of this matrix is 3, and the multiplicative order of the matrix is 64. A graphical view is given in Figure 1 (blue indicates 1 and yellow indicates 0).

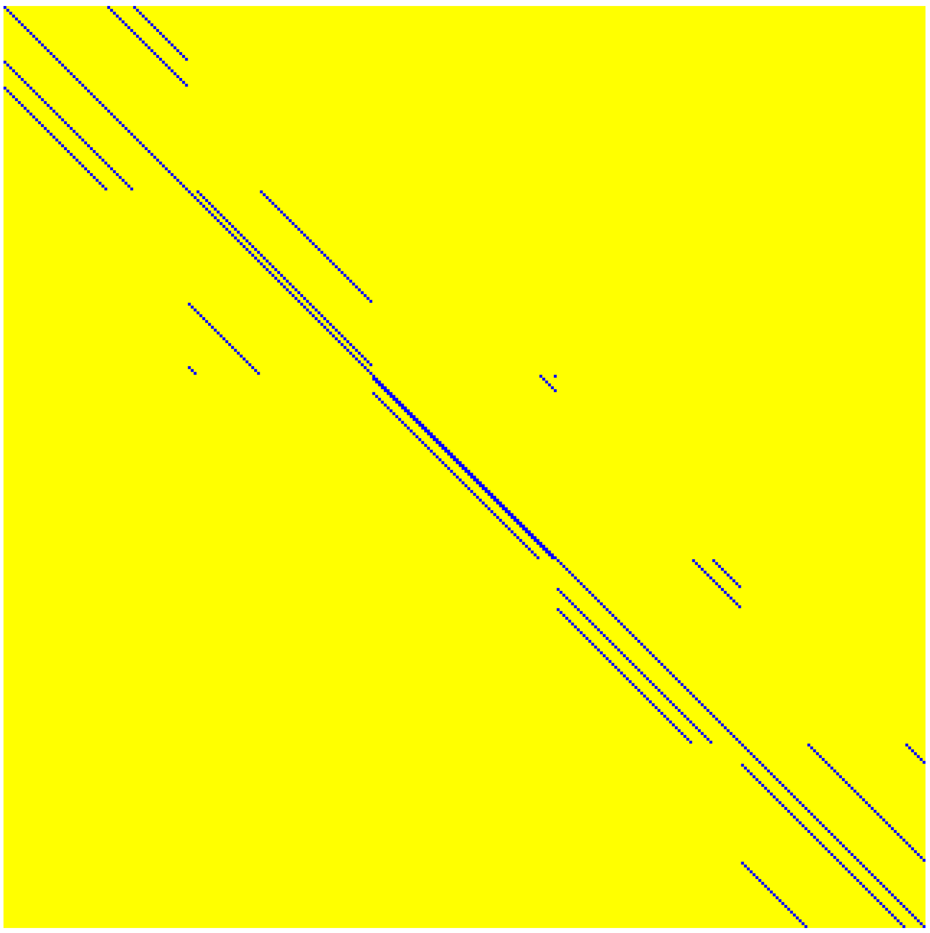

Figure 1: ASCON linear layer (320×320 binary matrix in graphical form)

3 Quantum Implementation of Linear Layer

The problem of optimization of a binary non-singular matrix while implementing for a classical device is well-studied [BP10, LSL⁺19, Köl19, Max19, XZL⁺20, BFI19, BDK⁺21, LWF⁺22]. However, the research works dedicated to find quantum quantum implementation/optimization are few and far between. Here we attempt to collate the major research works. We also cover the basic theory which is relevant for linear layer implementation (for the implementation of the non-linear component, one may refer to, e.g., [DBSC19, CBC23]).

3.1 Metrics

Qubit Analogous to the concept of bit in the classical computing, we use *quantum bits* (*qubits* for short). It is customary to write the qubits using the *Dirac's ket notation*, e.g., $|0\rangle$ or $|1\rangle$.

CNOT and SWAP Gates The *Controlled NOT* (*CNOT* for short) and *SWAP* gates form the basis of linear operations. The CNOT gate basically works like an in-place XOR gate. The circuit diagrams for the gates are as shown in Figure 2.

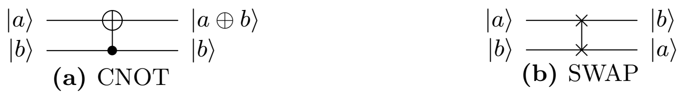

Figure 2: Basic quantum gates

A SWAP gate can be implemented using 3 CNOT gates (see, e.g., [DBSC19, Figure 3]). It may not be possible to swap two qubits if those are not adjacent, see [Ser17, Page 6] or [WFH11].

Quantum Circuit and Depth The quantum circuits are composed of the quantum gates and those operate on the qubits. For instance, the quantum circuit corresponding to the implementation stated in Example 4 is given in Figure 3.

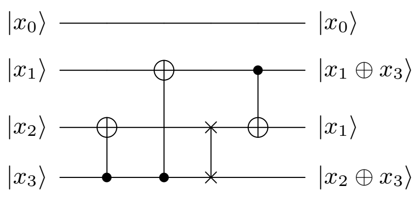

Figure 3: A quantum circuit (with CNOT and SWAP gates)

The depth of a logical circuit can be defined as the number of combinational logic gates along the longest path of the circuit. Unlike the classical depth, the quantum depth is generally not simple to calculated (noted in [JBK$^+$22b]), due to the fact fact a quantum gate can have only 1 fan-out. More relevant discussion on depth can be found in [ZH23]; and apparently this is the only paper that attempts to optimize for quantum depth by using a greedy algorithm.

3.2 Naïve Quantum Implementation

The so-called *d-XOR* implementation is proposed to indicate the naïve implementation in a classical circuit [KPPY14]. The d-XOR count for the ASCON linear layer is 640, as each row has two XOR operations. However, due to fan-out restriction in a quantum circuit, this implementation becomes incompatible.

For the naïve quantum implementation of a $n \times n$ binary matrix, we first need to introduce (up to) n ancilla qubits (effectively doubling the qubit count) initialized at $|0\rangle$. Then the ancilla qubits are updated with the initial values of the first n qubits. This helps retain the original matrix while the n qubits are overwritten.

For the lack of a better terminology, we call this the *naïve quantum* implementation or *direct-CNOT* (*d-CNOT* for short) implementation. An example can be seen from Example 1. Here and in the subsequent Examples, we use the variable x with subscript starting from 0 to indicate the matrix; if needed the variable y with subscript starting from 0 is also used to indicate each row of the matrix (thus, x variables act as the input and y variables act as the output for the implementation).

Example 1 (Naïve quantum). Consider the binary matrix $M^{4\times4}$:

$$\begin{pmatrix} 1\,1\,0\,0 \\ 0\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,0\,0\,1 \end{pmatrix}$$

Our original qubits are $\{x_0, x_1, x_2, x_3\}$. In the first step, 4 ancilla qubits are created and initialized to 0:

$$\begin{aligned} x_4 &\leftarrow |0\rangle \\ x_5 &\leftarrow |0\rangle \\ x_6 &\leftarrow |0\rangle \\ x_7 &\leftarrow |0\rangle \end{aligned}$$

After this, the ancilla qubits are updated in-place, so that each copies one original qubit:

$$\begin{aligned} x_4 &\leftarrow x_4 \oplus x_0 \\ x_5 &\leftarrow x_5 \oplus x_1 \\ x_6 &\leftarrow x_6 \oplus x_2 \\ x_7 &\leftarrow x_7 \oplus x_3 \end{aligned}$$

Equivalently, M is augmented with the 4×4 identity matrix to create this 8×4 matrix, which we name M':

$$M = \begin{pmatrix} 1\,1\,0\,0 \\ 0\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,0\,0\,1 \end{pmatrix} \longrightarrow \begin{pmatrix} 1\,1\,0\,0 \\ 0\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,0\,0\,1 \\ \hdashline 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{pmatrix} = M'$$

Finally, we update the original qubits in-place (in M') with the help of the ancilla qubits:

$$\begin{aligned} x_0 &\leftarrow x_0 \oplus x_5 \\ x_1 &\leftarrow x_1 \oplus x_6 \\ x_1 &\leftarrow x_1 \oplus x_7 \\ x_2 &\leftarrow x_2 \oplus x_4 \\ x_2 &\leftarrow x_2 \oplus x_5 \\ x_2 &\leftarrow x_2 \oplus x_7 \\ x_3 &\leftarrow x_3 \oplus x_4 \end{aligned}$$

Therefore, the naïve quantum implementation takes 8 qubits and 11 CNOT operations in total, with 3 quantum depth.

□

It may not always be necessary to double the number of qubits in the naïve quantum implementation. This can manifest in two ways (see Examples 2 and 3):

1. Not all original qubits are updated (happens when the i^{th} row of the matrix is identical to the i^{th} row of an identity matrix).
2. Not all original qubits are needed to update other qubits (happens when the i^{th} column of the matrix is identical to the i^{th} column of an identity matrix).

Example 2 (Not all qubits are updated). The naïve quantum implementation of the following matrix is possible without copying x_3 to an ancilla qubit, as the 3rd row has not been modified:

$$\begin{pmatrix} 1&1&0&0\\ 0&1&0&1\\ 0&1&1&1\\ 0&0&0&1 \end{pmatrix}$$

□

Example 3 (Not all original qubits are needed). The naïve quantum implementation of the following matrix is possible without copying x_0 and x_2 to ancilla qubits, as the 0th and 2nd columns have not been modified:

$$\begin{pmatrix} 1&1&0&0\\ 0&1&0&1\\ 0&1&1&1\\ 0&0&0&1 \end{pmatrix}$$

Thus, it can be implemented using only 1 ancilla qubit to store a copy of the original x_1.

□

There is also another notion that captures the essence of in-place implementation. This is the so-called s_ϵ-*XOR* implementation [BDK+21, BKD+21b]. In particular, when $\epsilon = 1$, this implementation reduces to factorization of a binary non-singular matrix into matrices for which the quantum implementation is apparent.

3.3 In-place Implementations

Legacy Algorithms Two legacy algorithms are known to have a quantum-friendly outcome:

1. Gauss-Jordan elimination (mentioned in [Köl19]);
2. PLU factorization (used in some form in [ASR12, GLRS16, ASAM18, vH19, ZWS+20, JNRV20]).

As these two algorithms are well-known, we briefly describe those in Examples 4 and 5 for the sake of completeness (similar to Example 1, we use the variable x with subscript starting from 0). For the PLU factorization, we use the Sage[3] implementation.

Example 4 (Gauss-Jordan elimination). Consider the binary matrix $M^{4\times4}$:

$$\begin{pmatrix} 1&0&0&0\\ 0&1&0&1\\ 0&1&0&0\\ 0&0&1&1 \end{pmatrix}$$

By applying the Gauss-Jordan elimination we obtain the following factorization of the matrix:

$$\begin{pmatrix} 1&0&0&0\\ 0&1&0&1\\ 0&1&0&0\\ 0&0&1&1 \end{pmatrix} = \begin{pmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&1&1&0\\ 0&0&0&1 \end{pmatrix} \begin{pmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&0&1\\ 0&0&1&0 \end{pmatrix} \begin{pmatrix} 1&0&0&0\\ 0&1&0&1\\ 0&0&1&0\\ 0&0&0&1 \end{pmatrix} \begin{pmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&1&1\\ 0&0&0&1 \end{pmatrix}$$

In this factorization each matrix is necessarily an elementary matrix, corresponding to an elementary operation. Thus, the sequence of gates given by the above factorization is as follows:

$$\begin{aligned} x_2 &\leftarrow x_2 \oplus x_3 \\ x_1 &\leftarrow x_1 \oplus x_3 \\ x_2, x_3 &\leftarrow x_3, x_2 \\ x_2 &\leftarrow x_2 \oplus x_1 \end{aligned}$$

This implementation incurs 3 CNOT and 1 SWAP gates with 4 quantum depth (can be seen from Figure 3).

□

[3] https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix2.html#sage.matrix.matrix2.Matrix.LU

Example 5 (PLU factorization). Consider the binary matrix $M^{4\times4}$:

$$\begin{pmatrix} 1\,0\,1\,0 \\ 0\,1\,0\,1 \\ 0\,1\,0\,0 \\ 1\,0\,0\,1 \end{pmatrix}$$

By applying the PLU factorization, we obtain the following:

$$\begin{pmatrix} 1\,0\,1\,0 \\ 0\,1\,0\,1 \\ 0\,1\,0\,0 \\ 1\,0\,0\,1 \end{pmatrix} = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,1\,0 \end{pmatrix} \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 1\,0\,1\,0 \\ 0\,1\,0\,1 \end{pmatrix} \begin{pmatrix} 1\,0\,1\,0 \\ 0\,1\,0\,1 \\ 0\,0\,1\,1 \\ 0\,0\,0\,1 \end{pmatrix}$$

These three matrices are respectively a permutation matrix, a lower triangular matrix and an upper triangular matrix; and can be implemented following [vH19, Algorithm 1]. The implementation is given as follows (that incurs 5 CNOT and 1 SWAP gates, with 4 quantum depth):

$$\begin{aligned} x_0 &\leftarrow x_0 \oplus x_2 \\ x_1 &\leftarrow x_1 \oplus x_3 \\ x_2 &\leftarrow x_2 \oplus x_3 \\ x_3 &\leftarrow x_3 \oplus x_1 \\ x_2 &\leftarrow x_2 \oplus x_0 \\ x_2, x_3 &\leftarrow x_3, x_2 \end{aligned}$$

□

XZLBZ and Its Modification A new idea for in-place implementation of binary matrices is proposed by XZLBZ [XZL+20]. This is subsequently used by various research works such as [JBK+22b, HS22, YJBS23].

The details of the algorithm used by XZLBZ [XZL+20] is omitted here for brevity though, we present a summarized view for the sake of completeness. The algorithm runs it two phases.

1. In the first phase, an initial factorization is done. In this process, the first priority is to use A* search. However, if there is no resolution from the A* search, then the algorithm falls back to Gauss-Jordan elimination.
2. In the second phase, the algorithm tries to find an alternate representation by replacing gap number of CNOT operations with a reduced number of CNOT operations. The gap variable is iterated over, initially being set to the length of the sequence. The program terminates if the gap hits a certain lower bound (which is hard-coded as 4 in their source-code).

In this work, we take a closer look and make some in-house modifications to XZLBZ. The main objective for such change is to accommodate the algorithm/source-code run with larger (i.e., $> 32 \times 32$) binary matrices.

In another direction, the work of Zhu-Huang [ZH23] deals with taking the output from XZLBZ, then post-process it to find a lower quantum depth. A greedy algorithm is used for this purpose. This is among the first, if not the first, work to purposefully optimize for quantum depth. One notable feature in the output is that, unlike XZLBZ [XZL+20], there is no *final relabel* in [ZH23]. In other words, in a typical XZLBZ output, there are assignments to the y variables. In contrast, Zhu-Huang's implementation does not need any special y-variable assignment (as a y-variable corresponds to its x-variables). We also make changes to XZLBZ so that our modified version optionally returns with/without final relabel[4]. See Example 6 for an implementation with final relabel and another implementation without final relabel of the same matrix.

Example 6 (Final relabel). Consider the following matrix:

$$\begin{pmatrix} 0\,1\,1\,0\,0 \\ 1\,0\,0\,1\,1 \\ 1\,0\,1\,1\,0 \\ 1\,1\,0\,0\,0 \\ 1\,1\,0\,0\,1 \end{pmatrix}$$

[4]Our implementations (given in the repository) follow the without final relabel format, where the initial permutation is declared at the top.

The matrix is implemented with final relabel by XZLBZ [XZL+20] as follows (notice that the y-variables do not directly match the corresponding x-variables at the LHS):

$$\begin{aligned}
x_3 &\leftarrow x_3 \oplus x_0 \\
(x_0 &\leftarrow x_0 \oplus x_1) \rightarrow y_3 \\
(x_1 &\leftarrow x_1 \oplus x_2) \rightarrow y_0 \\
(x_2 &\leftarrow x_2 \oplus x_3) \rightarrow y_2 \\
(x_3 &\leftarrow x_3 \oplus x_4) \rightarrow y_1 \\
(x_4 &\leftarrow x_4 \oplus x_0) \rightarrow y_4
\end{aligned}$$

The matrix can also be implemented without final relabel, by using 2 SWAP gates at the beginning (this is similar to [ZH23]). In this implementation, the last assignment of an x-variable acts as the corresponding y-variable, and is given as follows:

$$\begin{aligned}
x_0, x_1 &\leftarrow x_1, x_0 \\
x_1, x_3 &\leftarrow x_3, x_1 \\
x_1 &\leftarrow x_1 \oplus x_3 \\
(x_3 &\leftarrow x_3 \oplus x_0) \rightarrow y_3 \\
(x_0 &\leftarrow x_0 \oplus x_2) \rightarrow y_0 \\
(x_2 &\leftarrow x_2 \oplus x_1) \rightarrow y_2 \\
(x_1 &\leftarrow x_1 \oplus x_4) \rightarrow y_1 \\
(x_4 &\leftarrow x_4 \oplus x_3) \rightarrow y_4
\end{aligned}$$

The implementation with final relabel can be described by following permutation matrix; and this is termed as the *reduced matrix* by XZLBZ [XZL+20]:

$$\begin{pmatrix}
0&1&0&0&0\\
0&0&0&1&0\\
0&0&1&0&0\\
1&0&0&0&0\\
0&0&0&0&1
\end{pmatrix}$$

□

SMT/MILP Model To the best of our finding, the only other idea for in-place implementation of the linear layer is reported in [BKD+21b, BKD21a]. The idea here is to convert the factorization into an SMT or MILP instance. However, it becomes mostly impractical to deal with $> 8 \times 8$ binary matrices, due to the solver taking long time.

4 Quantum Benchmark

With the background covered in Section 3, the following quantum benchmarks are reported in Table 1:

1. Naïve corresponds to using ancillary qubits to enable computing the new state of a qubit while maintaining the previous one (in a back-up qubit), which is required for the calculation of new state of two other qubits.
2. Gauss-Jordan elimination factorizes any binary matrix using elementary operations (which correspond to CNOT or SWAP gates).
3. PLU factorization outputs a given binary matrix into a permutations matrix, a lower triangular matrix and an upper triangular matrix.
4. Modified XZLBZ is our in-house modification of what is introduced in [XZL+20].

In the naïve quantum implementation, the qubit count $\times$ CNOT count $= 640 \times 960$; but the same metric for the result obtained using modified XZLBZ is 320×1595. Thus, we save about about 16.93%.

Table 1: Quantum benchmarks of ASCON linear layer

Method	Qubit count	CNOT count	Quantum depth
Naïve	640 (out-of-place)	960	26
Gauss-Jordan	320 (in-place)	2413	358
PLU		2413	288
Modified XZLBZ		1595	119

5 Conclusion

In this work, we extract the ASCON linear layer (which is a 320×320 binary non-singular matrix) and show its quantum benchmarks. To the best of our knowledge, this is the first-of-its-kind analysis on ASCON. We show 4 in-place implementations of the matrix, which is useful in finding a suitable quantum implementation (as in-place implementations do not require any ancilla/garbage qubit).

The best in-place implementation (in terms of either CNOT count or quantum depth) that we obtain is from a modified version of the XZLBZ [XZL+20]. We contribute in respective parts of the algorithm and the source-code of XZLBZ [XZL+20], so that now our version of XZLBZ works well even with the ASCON linear layer.

The naïve quantum implementation doubles the qubit count, whereas the in-place implementations maintain the same qubit count. Thus, one may be interested in a trade-off where the resulting circuit incurs relatively low CNOT count and/or quantum depth, but not all qubits are duplicated. Put in other words, some of the qubits are updated in-place, but the rest are duplicated and updated out-of-place.

References

ASAM18. Mishal Almazrooie, Azman Samsudin, Rosni Abdullah, and Kussay N. Mutter. Quantum reversible circuit of AES-128. *Quantum Information Processing*, 17(5):1–30, may 2018. 5

ASR12. Brittanney Amento, Rainer Steinwandt, and Martin Roetteler. Efficient quantum circuits for binary elliptic curve arithmetic: reducing t-gate complexity, 2012. 5

BDK+21. Anubhab Baksi, Vishnu Asutosh Dasu, Banashri Karmakar, Anupam Chattopadhyay, and Takanori Isobe. Three input exclusive-or gate support for boyar-peralta's algorithm. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*, volume 13143 of *Lecture Notes in Computer Science*, pages 141–158. Springer, 2021. 2, 5

BFI19. Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. More results on shortest linear programs. Cryptology ePrint Archive, Report 2019/856, 2019. https://eprint.iacr.org/2019/856. 2

BKD21a. Anubhab Baksi, Banashri Karmakar, and Vishnu Asutosh Dasu. POSTER: optimizing device implementation of linear layers with automated tools. In *Applied Cryptography and Network Security Workshops - ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, June 21-24, 2021, Proceedings*, volume 12809 of *Lecture Notes in Computer Science*, pages 500–504. Springer, 2021. 7

BKD+21b. Anubhab Baksi, Banashri Karmakar, Vishnu Asutosh Dasu, Dhiman Saha, and Anupam Chattopadhyay. Further insights on implementation of the linear layer. *SILC Workshop – Security and Implementation of Lightweight Cryptography*, 2021. https://www.esat.kuleuven.be/cosic/events/silc2020/wp-content/uploads/sites/4/2020/10/Submission1.pdf. 5, 7

BP10. Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings*, pages 178–189, 2010. 2

CBC23. Matthew Chun, Anubhab Baksi, and Anupam Chattopadhyay. Dorcis: Depth optimized quantum implementation of substitution boxes. Cryptology ePrint Archive, Paper 2023/286, 2023. https://eprint.iacr.org/2023/286. 2

DBSC19. Vishnu Asutosh Dasu, Anubhab Baksi, Sumanta Sarkar, and Anupam Chattopadhyay. LIGHTER-R: optimized reversible circuit implementation for sboxes. In *32nd IEEE International System-on-Chip Conference, SOCC 2019, Singapore, September 3-6, 2019*, pages 260–265, 2019. 2, 3

DEMS19. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to NIST, 2019. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf. 1, 2

GLRS16. Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying Grover's algorithm to AES: Quantum resource estimates. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography*, pages 29–43, Cham, 2016. Springer International Publishing. 5

HS22. Zhenyu Huang and Siwei Sun. 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. 6

JBB$^+$22. Kyungbae Jang, Anubhab Baksi, Jakub Breier, Hwajeong Seo, and Anupam Chattopadhyay. Quantum implementation and analysis of default. Cryptology ePrint Archive, Paper 2022/647, 2022. https://eprint.iacr.org/2022/647. 1

JBK$^+$22a. Kyungbae Jang, Anubhab Baksi, Hyunji Kim, Hwajeong Seo, and Anupam Chattopadhyay. Improved quantum analysis of SPECK and lowmc (full version). *IACR Cryptol. ePrint Arch.*, page 1427, 2022. 1

JBK$^+$22b. Kyungbae Jang, Anubhab Baksi, Hyunji Kim, Gyeongju Song, Hwajeong Seo, and Anupam Chattopadhyay. Quantum analysis of aes. Cryptology ePrint Archive, Paper 2022/683, 2022. https://eprint.iacr.org/2022/683. 3, 6

JNRV20. Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and lowmc. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 280–310. Springer, 2020. 5

JSK$^+$22. Kyungbae Jang, Gyeongju Song, Hyunjun Kim, Hyeokdong Kwon, Hyunji Kim, and Hwajeong Seo. Parallel quantum addition for korean block ciphers. *Quantum Inf. Process.*, 21(11):373, 2022. 1

Köl19. Lukas Kölsch. Xor-counts and lightweight multiplication with fixed elements in binary finite fields. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, pages 285–312, 2019. 2, 5

KPPY14. Khoongming Khoo, Thomas Peyrin, Axel York Poschmann, and Huihui Yap. FOAM: searching for hardware-optimal SPN structures and components with a fair comparison. In *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 433–450, 2014. 3

LJS$^+$22. Wai-Kong Lee, Kyungbae Jang, Gyeongju Song, Hyunji Kim, Seong Oun Hwang, and Hwajeong Seo. Efficient implementation of lightweight hash functions on gpu and quantum computers for iot applications. *IEEE Access*, 10:59661–59674, 2022. 1

LSL$^+$19. Shun Li, Siwei Sun, Chaoyun Li, Zihao Wei, and Lei Hu. Constructing low-latency involutory mds matrices with lightweight circuits. *IACR Transactions on Symmetric Cryptology*, pages 84–117, 2019. 2

LWF$^+$22. Qun Liu, Weijia Wang, Yanhong Fan, Lixuan Wu, Ling Sun, and Meiqin Wang. Towards low-latency implementation of linear layers. *IACR Transactions on Symmetric Cryptology*, 2022(1):158–182, Mar. 2022. 2

Max19. Alexander Maximov. Aes mixcolumn with 92 xor gates. Cryptology ePrint Archive, Report 2019/833, 2019. https://eprint.iacr.org/2019/833. 2

Ser17. Giuseppe Sergioli. Quantum circuit optimization for unitary operators over non-adjacent qudits. *Arxiv*, 11 2017. https://arxiv.org/pdf/1711.09765.pdf. 3

vH19. Iggy van Hoof. Space-efficient quantum multiplication of polynomials for binary finite fields with sub-quadratic toffoli gate count. *arXiv preprint arXiv:1910.02849*, 2019. 5, 6

WFH11. David S. Wang, Austin G. Fowler, and Lloyd C. L. Hollenberg. Surface code quantum computing with error rates over 1 *Phys. Rev. A*, 83:020302, Feb 2011. 3

XZL$^+$20. Zejun Xiang, Xiangyong Zeng, Da Lin, Zhenzhen Bao, and Shasha Zhang. Optimizing implementations of linear layers. *IACR Trans. Symmetric Cryptol.*, 2020(2):120–145, 2020. 1, 2, 6, 7, 8

YJBS23. Yujin Yang, Kyungbae Jang, Anubhab Baksi, and Hwajeong Seo. Optimized implementation and analysis of cham in quantum computing. *Applied Sciences*, 13(8), 2023. 6

ZH23. Chengkai Zhu and Zhenyu Huang. Optimizing the depth of quantum implementations of linear layers. Cryptology ePrint Archive, Paper 2023/165, 2023. https://eprint.iacr.org/2023/165. 3, 6, 7

ZWS$^+$20. Jian Zou, Zihao Wei, Siwei Sun, Ximeng Liu, and Wenling Wu. Quantum circuit implementations of AES with fewer qubits. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 697–726, Cham, 2020. Springer International Publishing. 5