# A Framework for Depth-Efficient Quantum Implementations of Linear Layers

Kyungbae Jang[1], Anubhab Baksi[2], and Hwajeong Seo[1]

[1] Hansung University, Seoul, South Korea
[2] Lund University, Sweden
starj1023@gmail.com, anubhab.baksi@eit.lth.se, hwajeong84@gmail.com

**Abstract.** Quantum depth plays a critical role in determining the performance of quantum implementations, yet quantum programming tools often fail to produce depth-optimal compilations of linear layers. In this work, we present a systematic and automated framework that reorders quantum gate sequences of linear layers to obtain depth-efficient quantum implementations. Our method consistently produces circuits with lower depth compared to prior implementations.
We apply the framework to a range of cryptographic operations, including the AES MixColumn, internal layers of the AES S-box, binary field squaring, and modular reduction in binary field multiplication. In all these cases, our method achieves meaningful reductions in quantum depth—for example, lowering the depth of the AES MixColumn and S-box circuits.
This work explores optimal quantum circuit designs for quantum programming tools, improves the accuracy of quantum resource estimation for cryptanalysis, and supports more realistic evaluations of post-quantum security.

**Keywords:** Quantum depth · Linear layer · Framework · Quantum programming tool

## 1 Introduction

Quantum computing poses a substantial threat to modern cryptography, especially to public key algorithms. Shor's algorithm [15], for instance, can reduce the complexity of attacking such algorithms to polynomial time, and this has led to extensive research into the vulnerabilities of public key systems against quantum adversaries [13,5,2,8]. Symmetric key ciphers are generally regarded as more resilient to quantum attacks, since Grover's algorithm [4] provides only a quadratic speedup by reducing the search complexity of a $k$-bit key to roughly $\sqrt{2^k}$ operations. Nevertheless, to maintain comparable levels of post-quantum security, symmetric ciphers must account for quantum key search attacks with complexity reduced by a square root.

Estimating the quantum resources required for such attacks is a central component of security evaluation. Accurate resource estimates depend not only on

asymptotic complexity but also on practical implementation details of the underlying quantum circuits. However, quantum programming tools (such as ProjectQ [16][3], Qiskit[4], and Q#[5]) often fail to produce depth-optimal implementations of linear layers at the compiler level, which are fundamental building blocks in quantum circuit simulations. Since quantum depth directly influences the runtime of an algorithm under noise and decoherence, estimating optimal/correct quantum depth of these layers is crucial for realistic cryptanalytic assessments.

Let us briefly review some irregular results on quantum depth in previous quantum implementations. In the AES quantum circuit implementation presented at Eurocrypt'20 by Jaques et al. [9], the same MixColumn from [3] was implemented. However, [9] reported a much higher quantum depth compared to [3]. In [6], Jang et al. analyzed this issue and additionally proposed depth-optimized implementations of the AES S-box and MixColumn by reordering quantum gate sequences to make them suitable for quantum programming tools. In [8], the authors presented a compiler-friendly implementation of binary field squaring for quantum programming tools by adjusting the order of sequential gate operations.

In this work, we discuss this issue and introduce a framework that automatically reorders quantum gate sequences to discover depth-efficient implementations of linear layers. The key idea is to explore multiple valid execution orders of a circuit and to select those that minimize quantum depth. Unlike manual reordering, which has been used in previous works [6,8], our framework systematizes and automates the process, enabling more consistent improvements across diverse settings. We further demonstrate the practicality of our approach by applying it to real-world examples, where our tool identifies implementations with reduced and more accurate quantum depths compared to previously reported results.

### Contribution

The contributions of this paper are summarized as follows:

1. **Reordering Framework for Quantum Gate Sequences.** We present an automated framework that systematically reorders CNOT and X gate sequences in quantum linear layers to obtain depth-efficient implementations. By constructing dependency graphs, applying randomized topological sorts, and selecting optimal schedules, our method systematically replaces manual reordering approaches.
2. **Depth-Efficient Quantum Implementations for Linear Layers.** Using our framework, we identify depth-efficient implementations of quantum linear layers for quantum programming tools. By exploring reordered gate schedules, our method achieves reduced circuit depth compared to previous implementations.

[3] https://projectq.ch/
[4] https://www.ibm.com/quantum/qiskit
[5] https://learn.microsoft.com/ko-kr/azure/quantum/qsharp-overview

3. **Applications to Real-World Examples.**
   We demonstrate the effectiveness of our framework on various linear layers, including AES MixColumn and binary field squaring. In addition, we apply it to the internal layers of the AES S-box and to binary field multiplication, where our tool likewise achieves reductions in overall circuit depth.

## 2 Foundations

### 2.1 Quantum Gates

Several quantum gates are commonly used to implement arithmetic in quantum circuits, including the X (NOT), CNOT, and Toffoli (CCNOT) gates, as shown in Figure 1. The X gate corresponds to the classical NOT operation, flipping the state of a qubit (i.e., $\mathrm{X}(a) = \sim a$). The CNOT gate acts on two qubits, modifying the target qubit depending on the control qubit. If the control is 1, the target flips; if the control is 0, the target remains the same (i.e., $\mathrm{CNOT}(a, b) = (a, b \oplus a)$). Since this is equivalent to XORing the control qubit with the target, the CNOT gate functions as a quantum counterpart to the classical XOR operation. The Toffoli gate involves three qubits—two controls and one target. The target flips only when both control qubits are 1 (i.e., $\mathrm{Toffoli}(a, b, c) = (a, b, c \oplus (a \cdot b))$). This corresponds to XORing the AND of the control qubits with the target, making the Toffoli gate a quantum analogue of the classical AND.

Using these gates, encryption algorithms can be implemented in quantum circuits by substituting classical NOT, XOR, and AND with their quantum equivalents.

Optimizing quantum circuits often focuses on reducing the number of Toffoli gates, since they are particularly costly to implement. Each Toffoli gate is built from $T$ gates (which affect the $T$-depth) along with Clifford gates. Various decomposition methods have been proposed, and the full depth refers to the circuit depth after decomposition. In this work, we estimate the resources using a decomposition with 7 $T$ gates and 8 Clifford gates, giving a $T$-depth of 4 and a full depth of 8 for a single Toffoli gate, one of the methods proposed in [1].

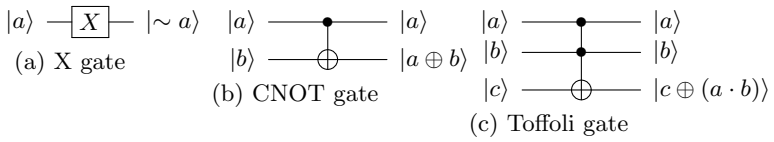


Fig. 1: Quantum gates

### 2.2 Quantum Implementation of Linear Layer

Optimizing linear layers [12,14,17,10,11] has been one of the important topics in cryptographic implementations. Such operations, including the MixColumns

transformation in AES, multiplications with MDS matrices, or diffusion layers in hash functions, frequently dominate the overall performance and resource usage in both hardware and software.

In the quantum context, the implementation of linear layers plays an equally crucial role, as they often dominate the circuit depth in Grover's [4] or Shor's algorithms [15]. Linear transformations are typically realized using CNOT gates (corresponding to the classical XOR operation). However, early implementations revealed significant inconsistencies in depth estimation across different realizations of the same operation. For example, the AES MixColumn implementation presented by Jaques et al. [9] resulted in much higher quantum depth than the earlier design by Grassl et al. [3], despite targeting the identical mapping. Such discrepancies highlighted the limitations of quantum programming tools, which frequently generate serialized schedules that fail to capture depth-optimal orderings of CNOT operations.

### 2.3 Related Work

In [6], Jang et al. addressed this problem by manually reordering quantum gate sequences to improve compatibility with compiler heuristics. They demonstrated depth-optimized implementations of the AES S-box and MixColumn through systematic reordering, achieving substantial reductions in circuit depth. Similar compiler-friendly techniques were applied to binary field squaring [8], showing that careful adjustment of gate execution order can eliminate unnecessary dependencies. Together, these works revealed that the depth of quantum linear layers is not determined solely by the algebraic transformation itself but is strongly influenced by scheduling strategies at the compiler level.

## 3 Our Framework

In this section, we introduce our framework for finding depth-efficient quantum implementations of linear layer operations. The main idea is to explore multiple valid execution orders of a given circuit. By generating diverse candidates and evaluating their quantum depth, we can identify implementations that minimize depth while preserving functional correctness. Figure 2 illustrates the overall workflow of our approach, which consists of three key steps: construction of the dependency graph (Algorithm 1, Section 3.1), randomized topological sorting (Algorithm 2, Section 3.2), and sampling and selection of the most depth-efficient implementations (Algorithm 3, Section 3.3).

### 3.1 Build Dependency Graph

Algorithm 1 builds a dependency graph from a sequence of CNOT and X gates. Each gate is treated as an operation that either reads or writes a qubit: a CNOT reads its control qubit and writes to its target, while an X gate only writes to its target. In this model, we define two ordering constraints.

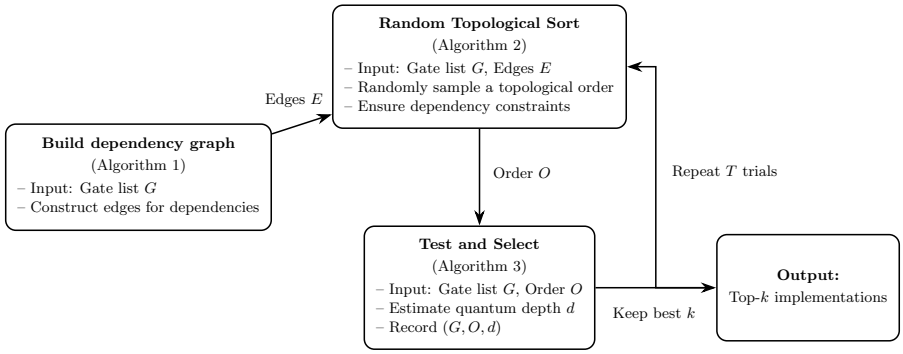


Fig. 2: Overview of our framework.

First, a Read-After-Write (RAW) edge is added whenever a CNOT reads a control qubit that was updated by a previous gate. This guarantees that the read sees the correct value. Second, a Write-After-Read (WAR) edge is added when a gate writes to a qubit that was used earlier as a control. This prevents the write from overwriting the value before all dependent reads have finished. Note that Write-After-Write (WAW) edges are not included, since reordering consecutive writes that are not read in between does not change the output of circuit and only reduces diversity of reordering.

The result of Algorithm 1 is a directed acyclic graph in which the nodes represent gates and the edges capture the RAW and WAR dependencies. This graph then be used to generate valid topological orderings of the gates, allowing different schedules that preserve correctness while exposing opportunities to reduce quantum depth.

### 3.2 Random Topological Sort

A *topological sort* is a linear ordering of the nodes in a directed acyclic graph (DAG) such that every directed edge $(u \to v)$ ensures that $u$ appears before $v$ in the ordering. In this context, the nodes correspond to gates, and the edges encode RAW and WAR dependencies between them.

Algorithm 2 generates a valid execution order of gates that respects all dependency constraints from Algorithm 1. Unlike a deterministic topological sort, our method introduces randomness in the selection step. At each iteration, among all gates whose dependencies have been satisfied (i.e., zero in-degree), one is chosen uniformly at random and appended to the current schedule. This process repeats until either all gates are placed or no valid gate remains. If the process succeeds, the output is a randomized topological ordering of the entire gate list; otherwise, it returns `None`.

This randomized procedure offers the following advantage for quantum programming tools: it explores multiple valid schedules instead of producing a single

**Algorithm 1:** Build dependency graph

**Input:** Gate list $G = \{g_1, \ldots, g_n\}$ with $g = \texttt{CNOT}(ctrl, tgt)$ or $\texttt{X}(tgt)$
**Output:** Directed edge set $E \subseteq \{(i \to j) \mid 1 \le i < j \le n\}$

$E \leftarrow \emptyset$
**for each** qubit label $q$: $Writes[q] \leftarrow [\,]$, $Reads[q] \leftarrow [\,]$
**for** $i \leftarrow 1$ to $n$ **do** ▷ collect per-qubit reads/writes
    **if** $g_i$ is `CNOT` **then**
        $Reads[ctrl(g_i)]$.append$(i)$
        $Writes[tgt(g_i)]$.append$(i)$
    **else if** $g_i$ is `X` **then**
        $Writes[tgt(g_i)]$.append$(i)$
    **end if**
**end for**
**for** $j \leftarrow 1$ to $n$ **do** ▷ RAW: all prior writes → current read
    **if** $g_j$ is `CNOT` **then**
        $c \leftarrow ctrl(g_j)$
        $P \leftarrow \{\, i \in Writes[c] \mid i < j \,\}$
        **if** $P \neq \emptyset$ **then**
            $E \leftarrow E \cup \{\, (i \to j) \mid i \in P \,\}$
        **end if**
    **end if**
**end for**
**for** $j \leftarrow 1$ to $n$ **do** ▷ WAR: all prior reads → current write
    **if** $g_j$ is `CNOT` **or** `X` **then**
        $q \leftarrow tgt(g_j)$
        $R \leftarrow \{\, i \in Reads[q] \mid i < j \,\}$
        **if** $R \neq \emptyset$ **then**
            $E \leftarrow E \cup \{\, (i \to j) \mid i \in R \,\}$
        **end if**
    **end if**
**end for**
**return** $E$

order, thereby enlarging the search space of possible gate schedules. By generating a diverse set of candidates, we can then simulate these schedules to identify those that achieve lower circuit depth.

### 3.3 Sampling and Selection

Algorithm 3 describes the outer loop of our framework. Given a gate list $G$, the procedure first constructs the dependency graph $E$ using Algorithm 1 to ensure that all RAW and WAR constraints are preserved. The objective is to explore the space of valid gate schedules and identify those that minimize circuit depth.

In each iteration up to the sampling budget $T$, Algorithm 2 generates a candidate order $O$ that satisfies all dependencies. If $O$ is invalid (`None`) or has already been encountered ($O \in Seen$), the sample is discarded to avoid redundancy. Otherwise, $O$ is added to the set of visited schedules. The candidate order is

**Algorithm 2:** Randomized topological sort

**Input:** Gate list $G = \{g_1, \ldots, g_n\}$, dependency edges $E$
**Output:** A randomized topological order $Order$ or `None`

Compute in-degrees $\deg[j]$ for all $1 \le j \le n$
$Pool \leftarrow \{\, j \mid \deg[j] = 0 \,\}$, $Order \leftarrow [\,]$
**while** $Pool \neq \emptyset$ **do**
    Randomly shuffle $Pool$
    $u \leftarrow$ pop one from $Pool$
    append $u$ to $Order$
    **for** each $(u \to w) \in E$ **do**
        $\deg[w] \leftarrow \deg[w] - 1$
        **if** $\deg[w] = 0$ **then**
            add $w$ to $Pool$
        **end if**
    **end for**
**end while**
**if** $|Order| = n$ **then**
    **return** $Order$
**else**
    **return** `None`
**end if**

**Algorithm 3:** Test and select

**Input:** Gate list $G$, sample budget $T$, top-$k$
**Output:** Top-$k$ schedules with minimum depth

$E \leftarrow$ **Build dependency graph**$(G)$
$Seen \leftarrow \emptyset$ ▷ set of visited orders
$Cand \leftarrow \emptyset$ ▷ set of candidate schedules
**for** $t \leftarrow 1$ to $T$ **do**
    $O \leftarrow$ **Randomized topological sort**$(E)$
    **if** $O =$ `None` **or** $O \in Seen$ **then**
        **continue**
    **end if**
    $Seen \leftarrow Seen \cup \{O\}$
    $d \leftarrow$ **Estimate depth**$(G, O)$ ▷ e.g., `depth_of_dag`
    $Cand \leftarrow Cand \cup \{(O, d)\}$
**end for**
Sort $Cand$ by depth in ascending order
**return** first $k$ elements of $Cand$

then simulated using a resource counter in quantum programming tools (e.g., ProjectQ, Q# or Qiskit) to measure its circuit depth $d$, and the pair $(O, d)$ is stored.

After $T$ iterations, the algorithm sorts all collected candidates by depth in ascending order and returns the top-$k$ schedules. This process enables the generation of a diverse pool of valid quantum implementations by reporting the most depth-efficient candidates for further evaluation.

## 4 Application to Real-World Examples

This section evaluates the performance of our reordering framework by applying it to quantum implementations that involve linear layers. We adopt various quantum implementations as benchmarks, reorder their gate sequences using our tool, and then assess the effectiveness of our approach. A summary of the results is provided in Table 1. In our table, only the AES MixColumn in [9] is an in-place implementation, while the others are out-of-place implementations.

### 4.1 AES MixColumn

In the AES quantum circuit implementation presented at Eurocrypt'20 by Jaques et al. [9], the authors implemented the same MixColumn as in the PQCrypto'16 AES quantum circuit implementation by Grassl et al. [3]. However, the following is noted in [9]: *"Note that [3] describes the same technique, while achieving a significantly smaller design than the one we obtain, but we were not able to reproduce these results."* This limitation motivates the development of our reordering framework for quantum gate sequences, which systematically finds depth-efficient implementations compatible with quantum programming tools.

The MixColumn operation in AES represents a linear transformation over the binary field. Since its quantum implementation consists primarily of CNOT gates, it provides a direct setting for applying our reordering framework. By applying our framework to the MixColumn implementation of [9], we obtain a depth-efficient gate sequence with a quantum depth of 81, reduced from the 111 reported in [9] (see Table 1).

### 4.2 AES S-box

Although the S-box is inherently non-linear, its internal decomposition reveals several small linear layers. This indicates that even in circuits dominated by non-linear transformations, depth reduction can be achieved by exploiting internal linear layers.

In [6], Jang et al. presented depth-optimized AES quantum S-box implementations by manually reordering gate sequences to achieve low quantum depth in quantum programming tools. In contrast, our framework automates this process to systematically obtain improved results. Among the three S-boxes in [6], this work further optimizes the Toffoli depth-4, low-qubit-count design, reducing the quantum depth from 67 to 64, as shown in Table 1.

### 4.3 Squaring in Binary Field

Squaring in binary fields is a linear operation that maps each input bit to a shifted position in the output. By reordering CNOT gates, our framework reduce the overall depth of the squaring circuit. In [8], the compiler-friendly implementation of squaring was proposed to reduce the quantum depth. In this work, we

Table 1: Comparison of quantum implementation depths.

| Linear layer | Size (matrix) | Source | #Qubit | Full depth |
|---|---|---|---|---|
| AES MixColumn | 32 × 32 | Jaques et al. [9] | 32* | 111 |
| | | This paper | | **81** |
| | | Liu et al. [12] | 135 | 13 |
| | | This paper | | **9** |
| | | Shi et al. [14] | 131 | 14 |
| | | This paper | | **9** |
| | | Sun et al. [17] | 122 | 18 |
| | | This paper | | **11** |
| | | Li et al. [10] | 137 | 11 |
| | | This paper | | **8** |
| | | Lin et al. [11] | 123 | 16 |
| | | This paper | | **11** |
| AES S-box (internal linear layers) | - | Jang et al. [6]❁ | 76 | 67 |
| | | This paper | | **64** |
| Squaring ($\mathbb{F}_{2^{16}}$) | 16 × 16 | $J^+$ [8] (Naïve) | 32 | 22 |
| | | $J^+$ [8] (Compiler-friendly) | | 14 |
| | | This paper | | **11** |
| Squaring ($\mathbb{F}_{2^{127}}$) | 127 × 127 | $J^+$ [8] (Naïve) | 254 | 175 |
| | | $J^+$ [8] (Compiler-friendly) | | 125 |
| | | This paper | | **125** |
| Multiplication ($\mathbb{F}_{2^{16}}$) | - | $J^+$ [7] (adopted in [8]) | 243 | 43 |
| | | This paper | | **41** |
| Multiplication ($\mathbb{F}_{2^{163}}$) | - | $J^+$ [7] (adopted in [8]) | 13161 | 66 |
| | | This paper | | **65** |

❁: Toffoli depth 4 and low qubit count version.
*: In-place implementation.

further reduce the quantum depth of their compiler-friendly implementation of squaring over $\mathbb{F}_2^{16}$ [8] from 14 to 11 (see Table 1). For squaring over $\mathbb{F}_2^{127}$ [8], our method achieves the same quantum depth as in [8].

### 4.4 Multiplication in Binary Field

Multiplication in binary fields is performed as a normal multiplication followed by modular reduction, where the reduction relies on a linear mapping that can be implemented using CNOT gates. We apply our reordering framework to the modular reduction circuits in [7] and obtain a reduction in quantum depth, as shown in Table 1, as shown in Table 1. Similar to the S-box (Section 4.2), this result is obtained by applying our tool to the small linear layer component (i.e., not the entire linear layer).

## 5 Conclusion

In this work, we present a systematic framework for constructing depth-efficient quantum implementations of linear layers. Unlike earlier approaches that relied on manual reordering of gate sequences, our method automates the process. With this approach, we consistently obtain quantum circuits of reduced depth across a variety of linear operations.

Our experimental evaluation demonstrates the practical impact of the framework. Applied to well-studied primitives such as AES MixColumn, AES S-box internal layers, binary field squaring, and modular reduction in binary field multiplication, the framework achieves meaningful depth reductions compared to previous implementations.

Overall, our work emphasizes that the depth of a quantum implementation is determined not only by the algebraic structure of the transformation but also by the scheduling of gates enforced by quantum programming tools. By providing a generalizable and automated method, this work advances accurate and efficient resource estimation for quantum cryptanalysis and supports more realistic assessments of post-quantum security.

## References

1. 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 3
2. Banegas, G., Bernstein, D.J., Van Hoof, I., Lange, T.: Concrete quantum cryptanalysis of binary elliptic curves. Cryptology ePrint Archive (2020) 1
3. 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) 2, 4, 8
4. 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, 4
5. Häner, T., Jaques, S., Naehrig, M., Roetteler, M., Soeken, M.: Improved quantum circuits for elliptic curve discrete logarithms. In: Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11. pp. 425–444. Springer (2020) 1
6. 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, 4, 8, 9
7. Jang, K., Kim, W., Lim, S., Kang, Y., Yang, Y., Seo, H.: Optimized implementation of quantum binary field multiplication with toffoli depth one. In: Information Security Applications: 23rd International Conference, WISA 2022, Jeju Island, South Korea, August 24–26, 2022, Revised Selected Papers. pp. 251–264. Springer (2023) 9

8. Jang, K., Srivastava, V., Baksi, A., Sarkar, S., Seo, H.: New quantum cryptanalysis of binary elliptic curves. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025**(2), 781–804 (2025) 1, 2, 4, 8, 9
9. 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, 4, 8, 9
10. Li, S., Sun, S., Li, C., Wei, Z., Hu, L.: Constructing low-latency involutory mds matrices with lightweight circuits. IACR Transactions on Symmetric Cryptology pp. 84–117 (2019) 3, 9
11. Lin, D., Xiang, Z., Zeng, X., Zhang, S.: A framework to optimize implementations of matrices. In: Paterson, K.G. (ed.) Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12704, pp. 609–632. Springer (2021). https://doi.org/10.1007/978-3-030-75539-3_25, https://doi.org/10.1007/978-3-030-75539-3_25 3, 9
12. Liu, Q., Wang, W., Fan, Y., Wu, L., Sun, L., Wang, M.: Towards low-latency implementation of linear layers. IACR Transactions on Symmetric Cryptology pp. 158–182 (2022) 3, 9
13. Roetteler, M., Naehrig, M., Svore, K.M., Lauter, K.: Quantum resource estimates for computing elliptic curve discrete logarithms. In: Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23. pp. 241–270. Springer (2017) 1
14. Shi, H., Feng, X., Xu, S.: A framework with improved heuristics to optimize low-latency implementations of linear layers. IACR Transactions on Symmetric Cryptology **2023**(4), 489–510 (2023) 3, 9
15. 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, 4
16. Steiger, D.S., Häner, T., Troyer, M.: Projectq: an open source software framework for quantum computing. Quantum **2**, 49 (2018) 2
17. Sun, Y., Li, T.: Improving the circuit implementation of the aes linear layer by improving boyar-peralta's algorithm and using sub-graph reconstruction. Cryptology ePrint Archive (2025) 3, 9