Optimized Quantum Implementation of SEED

Yujin Oh, Kyungbae Jang, Yujin Yang and Hwajeong Seo

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

- Introduction & Contribution
- Background
- Proposed Method
- Performance & Evaluation
- Conclusion

Introduction & Contribution

- **Grover's algorithm** reduce in the complexity of symmetric key cryptographic attacks to the square root.

  → This raises increasing challenges in considering symmetric key cryptography as secure.

- Establish secure **post-quantum** cryptographic systems.

  → There is a need for quantum **post-quantum security** evaluations of cryptographic algorithms.

- In this paper, we propose an optimized quantum circuits for **SEED**.

  → We assess the **post-quantum security** strength of SEED in accordance with NIST criteria.

Introduction & Contribution

1. Quantum Circuit Implementation of SEED

→ We demonstrate **the first implementation** of a quantum circuit for SEED, which is the one of Korean cipher.

2. Low-Depth Implementation of SEED

→ In our quantum circuit implementation of SEED, we focus on optimizing **Toffoli depth** and **full depth**.

3. Post-quantum Security Assessment of SEED

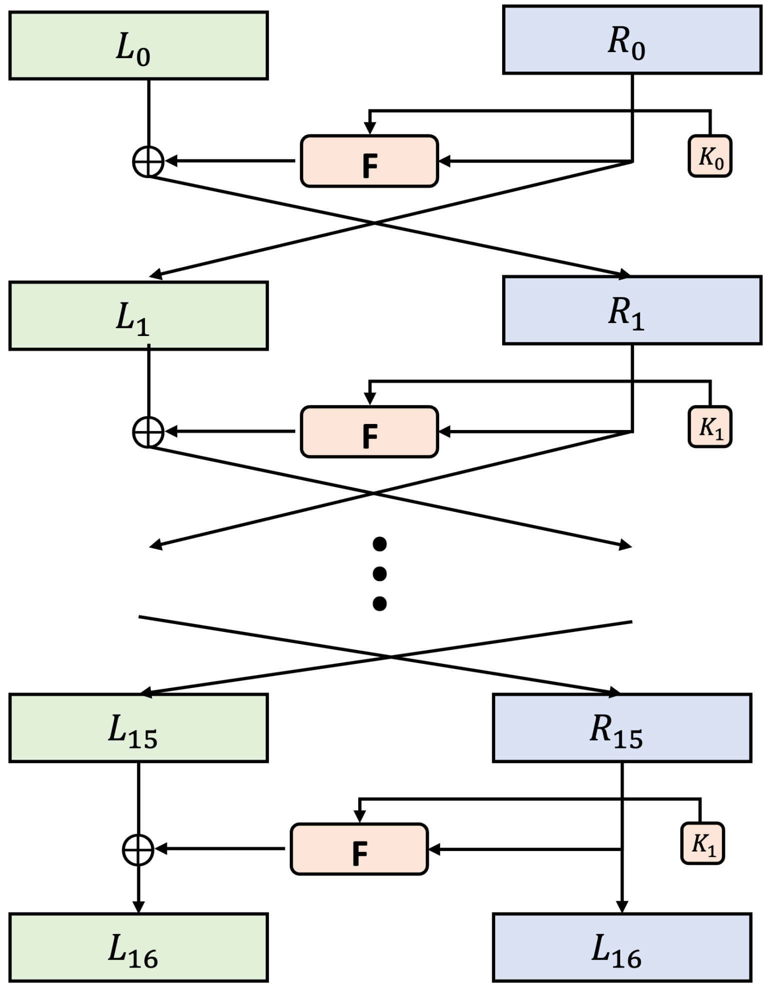
→ We estimate **the cost of Grover's key search** using an our implemented quantum circuit

→ We compare the estimated cost of Grover's key search for SEED with the **security levels** defined by NIST.

Background

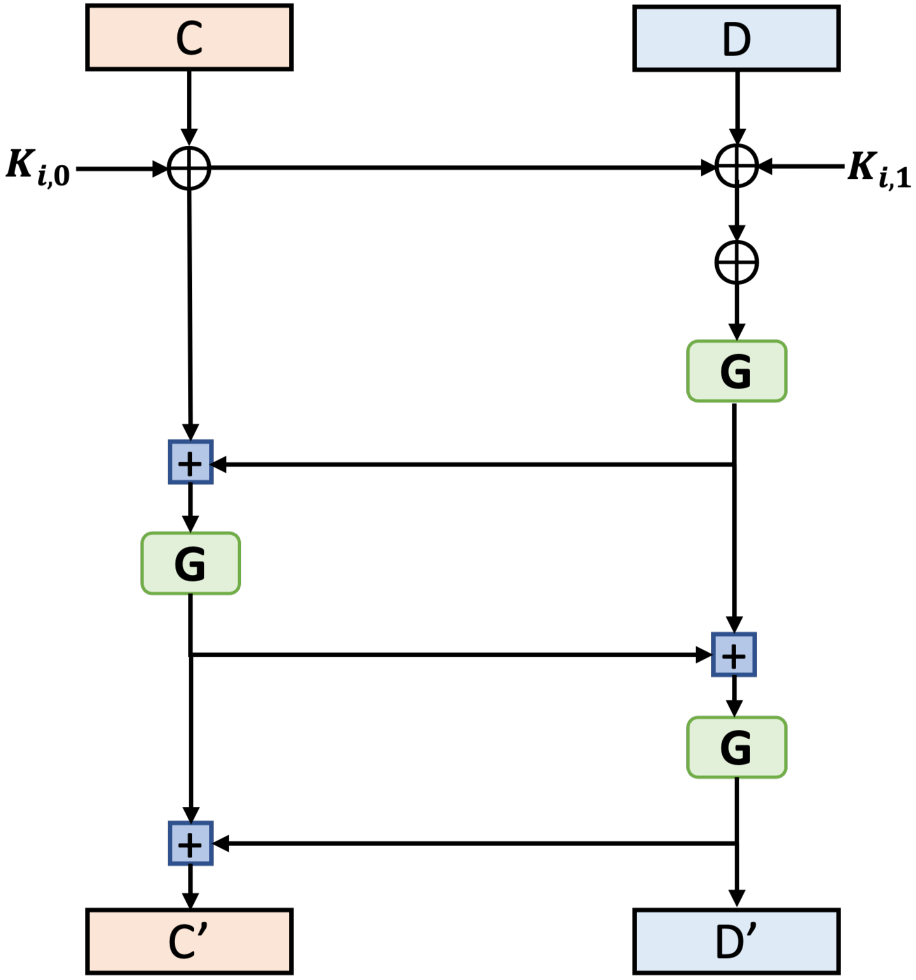
- **SEED**
  - SEED is a block cipher of **Feistel structure.**
  - Each round has a round function F .
  - A 128-bit block is **divided** into 64-bit blocks.
    - The right 64-bit block (R0) serves as the input to the F function with 64-bit round key.
    - The output of F function is XORed to the left 64-bit block (L0).

Cipher	block size	key size	rounds
SEED	128	128	16



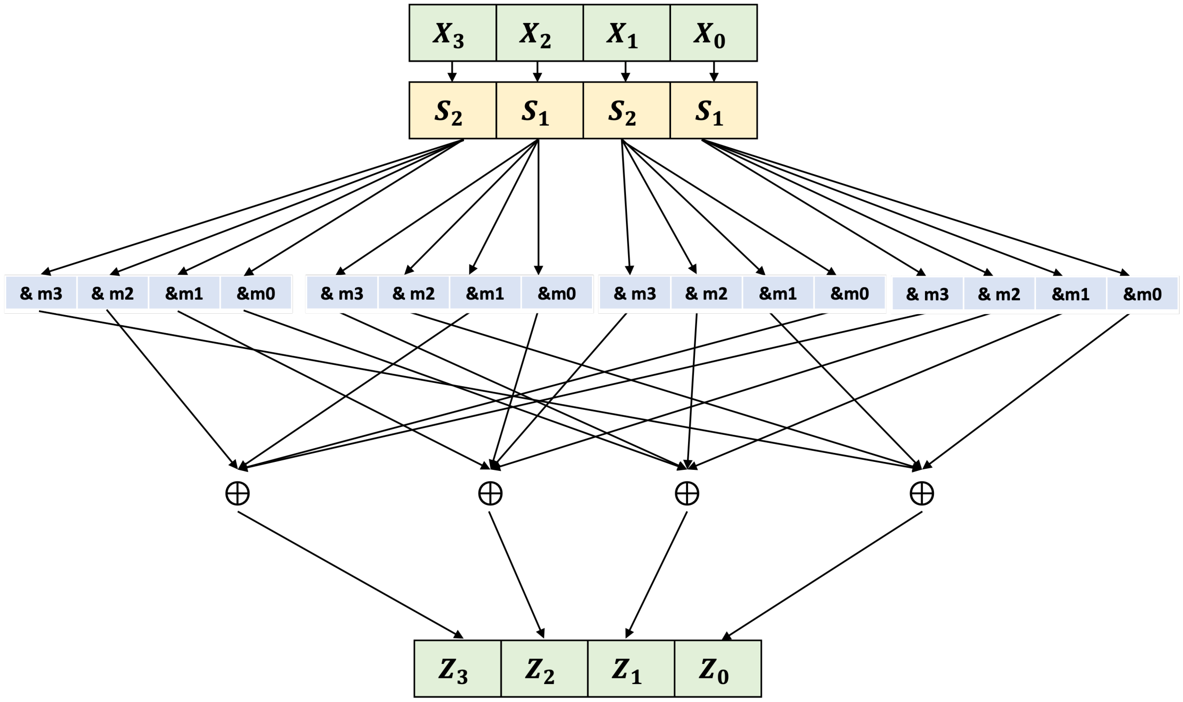
Background

- **F function**
  - The input of F function is 64-bit block and 64-bit round key $RK_i\ (K_{i,0}, K_{i,1})$
  - The 64-bit block is **divided** into two 32-bit blocks (C, D)
    - Each block is XORed with the round key.
  - The F function consists of **XOR operations** ($\oplus$), **modular additions** ($\boxplus$), and **G functions**.



Background

- **G function**
  - The 32-bit input block of the G function is **divided** into **8-bit** blocks (X_{0-3})
  - Each block becomes input for the **S-boxes**.
  - The output values of the S-boxes are **ANDed** (&) with the constants m_{0-3},
  - The results of these **AND** operations are XORed with each other to compute the final output (i.e., Z_{0-3}).



$$Y_3 = S_2(X_3), Y_2 = S_1(X_2), Y_1 = S_2(X_1), \ Y_0 = S_1(X_0)$$

$$Z_3 = (\mathrm{Y}_0 \& \mathrm{m}_3) \oplus (Y_1 \& m_0) \oplus (Y_2 \& m_1) \oplus (Y_3 \& m_2)$$

$$Z_2 = (\mathrm{Y}_0 \& \mathrm{m}_2) \oplus (Y_1 \& m_3) \oplus (Y_2 \& m_0) \oplus (Y_3 \& m_1)$$

$$Z_1 = (\mathrm{Y}_0 \& \mathrm{m}_1) \oplus (Y_1 \& m_2) \oplus (Y_2 \& m_3) \oplus (Y_3 \& m_0)$$

$$Z_0 = (\mathrm{Y}_0 \& \mathrm{m}_0) \oplus (Y_1 \& m_1) \oplus (Y_2 \& m_2) \oplus (Y_3 \& m_3)$$

Background

- **S-boxes**
  - It involves **exponentiation**, **matrix-vector multiplication**, and XORing a single constant.
  - Specifically, two distinct **S-boxes** (**S1 and S2**) are employed, which are calculated as follows:

$$S_1(x) = A^{(1)} \cdot x^{247} \oplus 169\ , \quad S_2(x) = A^{(2)} \cdot x^{251} \oplus 56$$

  - We use the primitive polynomial $\mathbb{F}_{2^8}/\ \boldsymbol{x^8 + x^6 + x^2 + x + 1}$

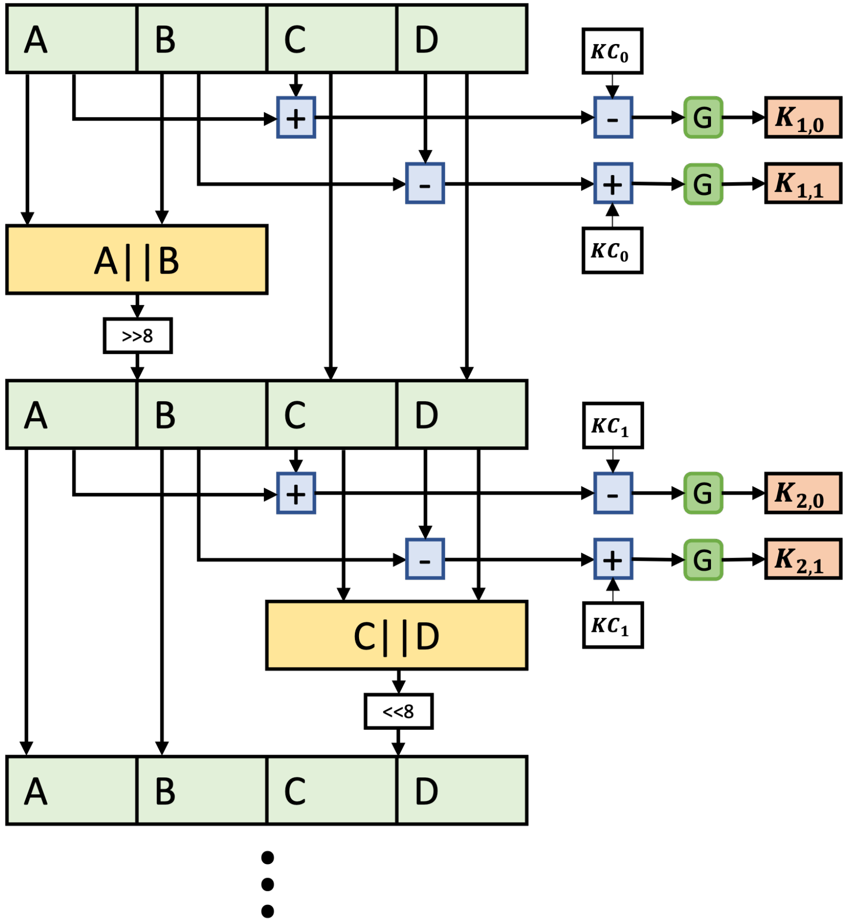
$$A^{(1)} = \begin{pmatrix} 1&0&0&0&1&0&1&0 \\ 1&1&1&1&1&1&1&0 \\ 1&0&0&0&0&1&0&1 \\ 0&1&0&0&0&0&1&0 \\ 0&1&0&0&0&1&0&1 \\ 0&0&1&0&0&0&0&1 \\ 1&0&0&0&1&0&0&0 \\ 0&0&0&1&0&1&0&0 \end{pmatrix}, \quad A^{(2)} = \begin{pmatrix} 0&1&0&0&0&1&0&1 \\ 1&0&0&0&0&1&0&1 \\ 1&1&1&1&1&1&1&0 \\ 0&0&1&0&0&0&0&1 \\ 1&0&0&0&1&0&1&0 \\ 1&0&0&0&1&0&0&0 \\ 0&1&0&0&0&0&1&0 \\ 0&0&0&1&0&1&0&0 \end{pmatrix}$$

Background

- **Key Schedule**
  - The 128-bit key is **divided** into **four** blocks.

    (A‖B‖C‖D, where ‖ denotes concatenation)
  - Key constant values(KC_i) are utilized.
  - Operations such as **shift** (≫, ≪), **modular addition**,

    modular subtraction (⊟), and **G function** are applied.



Background : Quantum gates

- **Quantum gates** commonly used for implementing quantum circuits of block ciphers

  → This is not an exhaustive list of all possible gates that can be used.

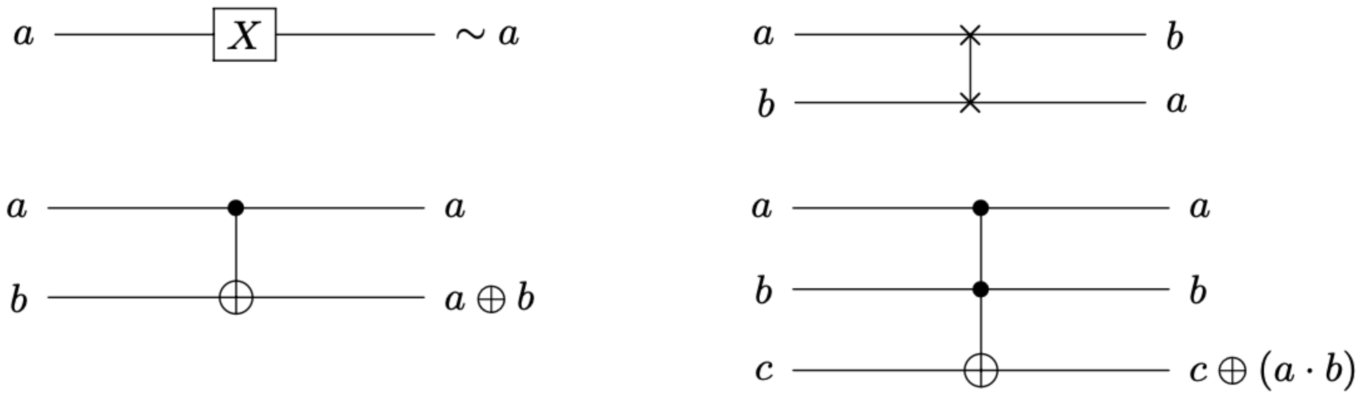
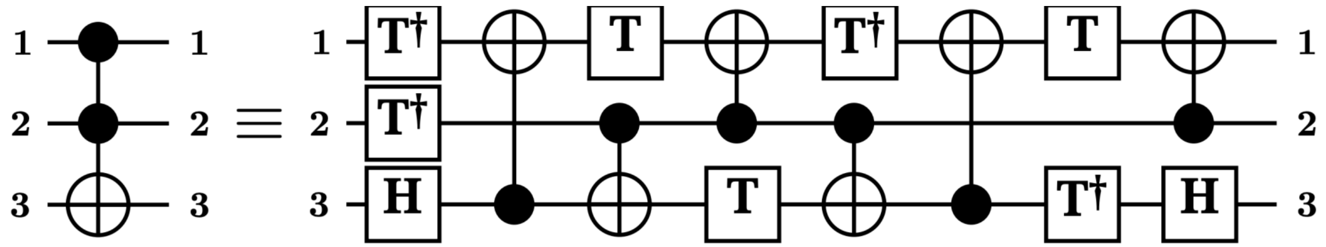


Fig. 5: Quantum gates: X (left top), Swap (right top), CNOT (left bottom) and Toffoli (right bottom) gates.



Toffoli gate decomposition (T- depth 4, total depth 8)

Background : Grover's key search

- **Key search using Grover's Algorithm**

1. Prepare a k-qubit key in a **superposition state** using **Hadamard gates**.

$$H^{\otimes k}|0\rangle^{\otimes k} = |\psi\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) = \frac{1}{2^{k/2}}\sum_{x=0}^{2^k-1}|x\rangle$$

2. This circuit encrypts a known plaintext(p) in a **superposition state** using a pre-prepared key, producing ciphertexts for every possible key value.

If the ciphertext matches the expected ciphertext, the sign of the desired key state to be recovered is **negated**.

$$f(x) = \begin{cases} 1 \text{ if } Enc_{key}(p) = c \\ 0 \text{ if } Enc_{key}(p) \neq c \end{cases}$$

$$U_f(|\psi\rangle\,|-\rangle) = \frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\,|x\rangle\,|-\rangle$$

3. The Diffusion Operator serves to **amplify the amplitude** of the target key state indicated by the oracle, identifying it by flipping the sign of said amplitude to negative.

Background : NIST criteria

- **The NIST criteria for quantum attack complexity.**
  - **Level - 1** $(2^{170} \rightarrow 2^{157})$

    → attacks on the security definition must require resources comparable to AES-128 key search.
  - **Level - 3** $(2^{233} \rightarrow 2^{221})$

    → attacks on the security definition must require resources comparable to AES-192 key search.
  - **Level - 5** $(2^{298} \rightarrow 2^{285})$

    → attacks on the security definition must require resources comparable to AES-256 key search.

Proposed Method

- **Implementation of S-box**
  - In quantum computers, the utilization of **look-up table** for implementing S-boxes **is not appropriate.**

    → We employ quantum gates to implement the S-boxes based on **Boolean expression**

  - Equations for S-boxes (primitive polynomials $p(x)$: $\mathbb{F}_{2^8}/\ x^8 + x^6 + x^2 + x + 1)$

$$S_1(x) = A^{(1)} \cdot x^{247} \oplus 169 \qquad x^{-1} \equiv x^{254}\ mod\ p(x)$$

$$S_2(x) = A^{(2)} \cdot x^{251} \oplus 56 \qquad (x^{-1})^8 \equiv x^{257}\ mod\ p(x)$$

$$(x^{-1})^4 \equiv x^{251}\ mod\ p(x)$$

  - We use **Itoh-Tsujii algorithm** to compute x^{-1}

$$x^{-1} = x^{254} = \left((x \cdot x^2) \cdot (x \cdot x^2)^4 \cdot (x \cdot x^2)^{16} \cdot x^{64}\right)^2$$

    → **Squaring and multiplication**

Proposed Method

- **Squaring in a S-box**
  - In squaring, modular reduction can be employed **PLU factorization** because it is a linear operation.

    → Without allocating additional ancilla qubits (i.e., **in-place**), **using only CNOT gates**.

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

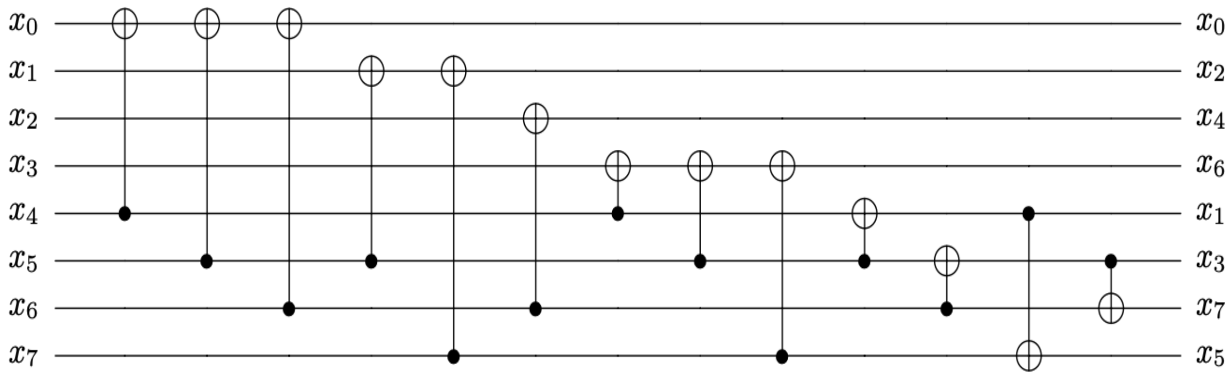


Fig. 6: Squaring in $\mathbb{F}_{2^8}/(x^8 + x^6 + x^5 + x + 1)$

Proposed Method

- **Multiplication in a S-box**
  - We apply **WISA’22 [Jang et al.] multiplication**

    →optimized with a **Toffoli depth of one** for any field size.

  - WISA’22[Jang et al.] multiplication
    - Using the **Karatsuba algorithm recursively** and allocating additional ancilla qubits.

      → All the AND operations become independent and the operations of **all Toffoli gates in parallel.**

      → The allocated ancilla qubits can be **reused** through **reverse operations**.

[Jang et al.] K. Jang, W. Kim, S. Lim, Y. Kang, Y. Yang, and J. Hwa, “Optimized implementation of quantum binary field multiplication with toffoli depth one,” in International Conference on Information Security Applications.

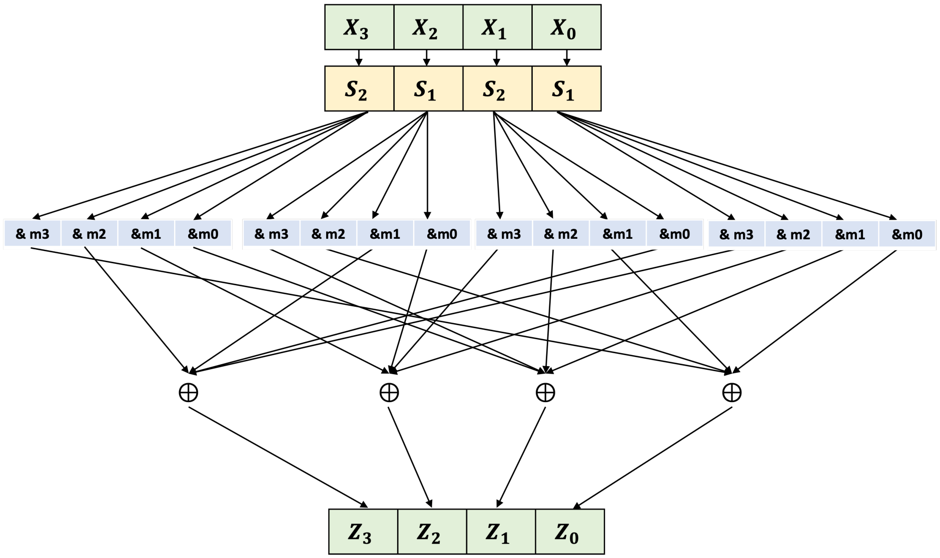
Proposed Method

- **Implementation of G function**
  - Each S-box requires **38 ancilla qubits** (specifically for **multiplication**).

    → Ancilla qubits can be initialized using **reverse operations** in WISA'22 multiplication, enabling their reuse.
  - S-boxes are executed sequentially → **increasing the depth.**

    → We optimize the depth by **parallelizing** **four S-boxes.**

    → This is achieved by allocating a total of **152 (38 × 4)** ancilla qubits at first.



Proposed Method

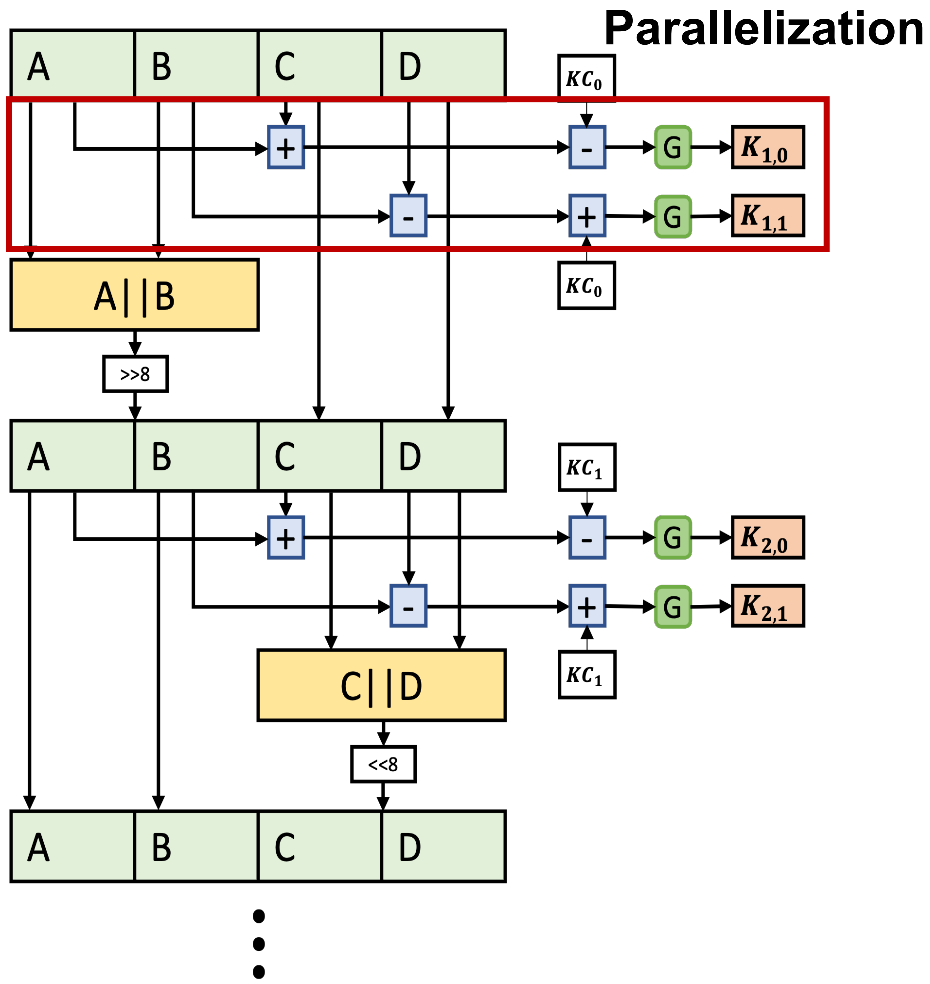
- **Implementation of Key Schedule**
  - Two 32-qubit subkeys ($K_{i,0}$, $K_{i,1}$) are generated.
  - The implementation is **parallelized** by operating two processes **simultaneously.**

    → We allocate two sets of 152 ancilla qubits(**152 x 2**)

    → **reduce the circuit depth**
  - We utilize the **CDKM adder** for addition in quantum.

    →Only **one** ancilla qubit is needed
  - We utilize the **logical Swap** that change the index of qubits.

    → Instead of Swap gates



Proposed Method

- **Implementation of Key Schedule**
  - For quantum addition, we allocate **two pairs of qubits (32 × 2)** to store the **KeyConstant values** (using on $K_{i,0}, K_{i,1}$ respectively).
  - Different KeyConstant values used in each round.

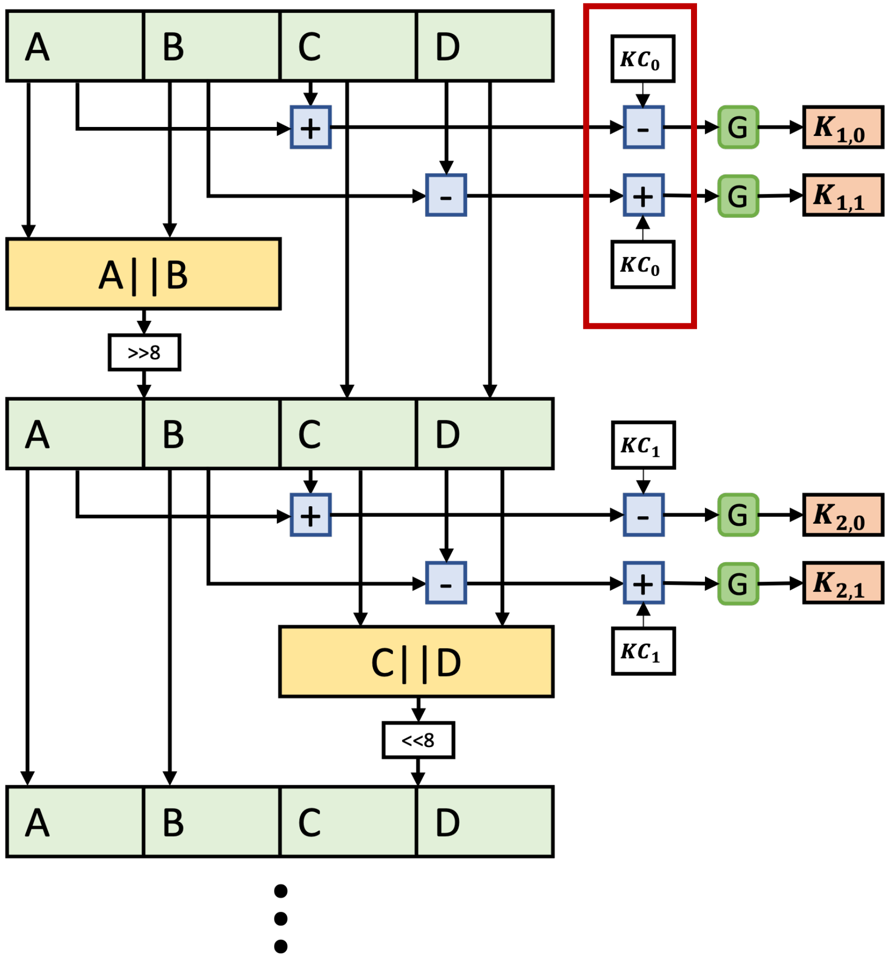
    → allocate and store new qubits every time.

    → Instead, we utilize **reverse operations**

    → initialize and reuse the qubits.

    →The **reverse operation** for the KeyConstant of quantum state involves only **X gates**.

    → a trivial overhead on the circuit depth.



Performance & Evaluation

- **Estimation of the quantum resources required for SEED**
  - We focus on reducing the **Toffoli depth** and **full depth**. (Trade-off with qubits)

    → For the **trade-off**, we report the **TD-M** and **FD-M** cost.

    (TD-M : Toffoli Depth x qubit, FD-M : Full Depth x qubit)

  - **NCT(NOT, CNOT, Toffoli) level**

Table 1: Required quantum resources for SEED quantum circuit implementation

Cipher	#X	#CNOT	#Toffoli	Toffoli depth	#Qubit	Depth	TD-M cost
SEED	8116	409,520	41,392	321	41,496	11,837	13,320,216

  - **Clifford + T level**

Table 2: Required decomposed quantum resources for SEED quantum circuit implementation

Cipher	#Clifford	$\#T$	T-depth	#Qubit	Full depth	FD-M cost
SEED	748,740	289,680	1,284	41,496	34,566	1,434,350,736

Performance & Evaluation

- **Grover's key search**
  - Grover's key search cost: the quantum resources x 2 x $\left\lfloor \frac{\pi}{4}\sqrt{2^k} \right\rfloor$
  - The Grover's key search attack cost for SEED is $\mathbf{1.291 \cdot 2^{164}}$

  → **SEED can be evaluated as achieving post-quantum security Level 1**.

Table 3: Cost of the Grover's key search for SEED

Cipher	Total gates	Total depth	Cost (complexity)	#Qubit	*TD-M* cost	*FD-M* cost
SEED	$1.559 \cdot 2^{84}$	$1.657 \cdot 2^{79}$	$1.291 \cdot 2^{164}$	41,497	$1.246 \cdot 2^{88}$	$1.049 \cdot 2^{95}$

Conclusion

- This paper presents the **first implementation** of a quantum circuit for **SEED**.

- We focus on optimizing **Toffoli and full depths**.

  → Utilizing **parallelization** and optimized **multiplication**, **squaring** and an **adder**.

- We analyze the cost of Grover's key search attack.

  → We confirm that SEED achieves post-quantum security **Level 1**.

Q & A