

# Quantum Speedup on Exhaustive-search Attacks on Cryptosystems

Week 7 Report for Class 75

2017320009 Sangheon Lee

2017320023 Mingyu Cho

May 30, 2020

# Table of Contents

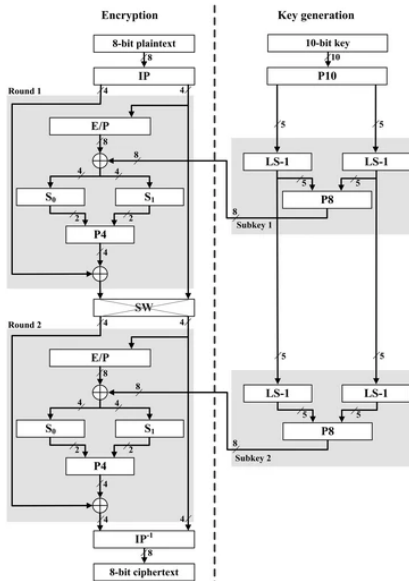
- 1 S-DES
- 2 Grover's Algorithm
- 3 Quantum S-DES Oracle
- 4 Quantum S-DES Implementation
- 5 Quantum S-DES Testing
- 6 Quantum S-DES Gate Analysis

# S-DES

# S-DES

- Simplified DES with 2 rounds
- Structure similar to DES but simplified with 10-bit key and 8-bit plaintext.
- Quantum oracle needs to be reversible, of which S-DES is (normally) not.

# Structure of S-DES

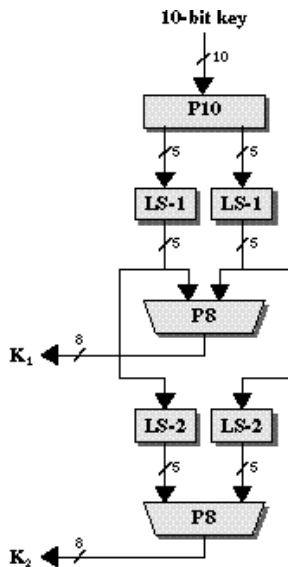


# Structure of S-DES

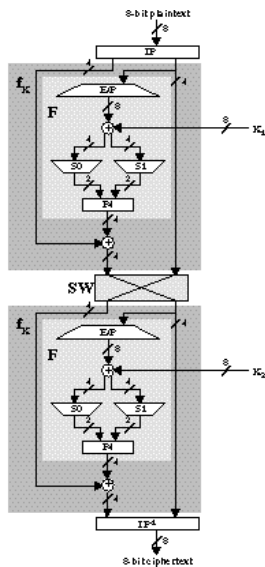
Consists of two main parts:

- Key Scheduling, where we generate subkeys from the given key
- Feistel function, where we encrypt the plaintext using the scheduled keys

# Structure of S-DES: Key Scheduling



# Structure of S-DES: Feistel function





# Grover's Algorithm

# Overview

- Grover's algorithm can find a specific state satisfying some condition among  $N = 2^n$  candidates in  $O(\sqrt{N})$  time, compared to classical runtime complexity  $O(N)$ .
- Grover's algorithm exploits qualities of quantum amplitudes to gain advantage of probability separation.
- It can brute-force 128-bit symmetric cryptographic key in roughly  $2^{64}$  iterations.

# Algorithm

Input:

- A quantum oracle  $\mathcal{O}$  which performs the operation  $\mathcal{O} |x\rangle = (-1)^{f(x)} |x\rangle$ , where  $f(x) = 0$  for all  $0 \leq x < 2^n$  except  $x_0$ , for which  $f(x_0) = 1$ .
  - Such quantum oracle is viable, and takes  $O(1)$  time.
- $n$  qubits initialized to the state  $|0\rangle$

Output:  $x_0$ , in runtime  $O(\sqrt{2^n})$  with error rate  $O(\frac{1}{2^n})$

# Algorithm

Procedure:

- ①  $|0\rangle^{\otimes n}$  (initial state)
- ②  $H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle = |\psi\rangle$  (Hadamard transform)
- ③  $[(2|\psi\rangle\langle\psi| - I)\mathcal{O}]^R |\psi\rangle \approx |x_0\rangle$  (Grover iteration for  $R \approx \frac{\pi}{4}\sqrt{2^n}$  times)
- ④  $x_0$  (measure)

Grover iteration in a nutshell: negate the amplitude of the desired state, followed by 'diffusion transform' which increases the amplitude of the desired state and lower the others.

## Quantum S-DES Oracle

# Quantumizing S-DES

- Most parts are permutations or compressions; no problems here.
- Two parts poses a challenge:
  - 1 S-Boxes
  - 2 Expansion

# Dealing with Expansions

- Due to No-cloning theorem, directly copying a qubit is not possible.
- Use XOR(CNOT) operation to copy the information.

# Dealing with S-boxes

Consists of Lookup tables: Not ideal for quantum computing

$$S_0 = \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \quad S_1 = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix}$$

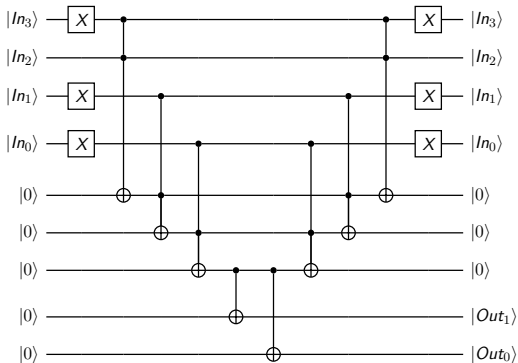


# Dealing with S-boxes: Quine-McCluskey Method

- Break the S-Boxes into fundamental and/or/not-gates
- Letting the input bits into S-Boxes  $ABCD$ :
  - S1 bit 1:  $AB'C' + A'B'C + BCD' + ABD + BC'D$
  - S1 bit 2:  $AD' + B'D' + A'BC + ABC'$
  - S2 bit 1:  $AB'C' + ACD + A'CD' + A'BD'$
  - S2 bit 2:  $B'C'D + AB'C + BCD + BC'D'$
- Directly calculating this would cause too many ancilla bits to use, i.e. for each and/or operation, one ancilla bit is required.

# Dealing with S-boxes: The Brute-force Method

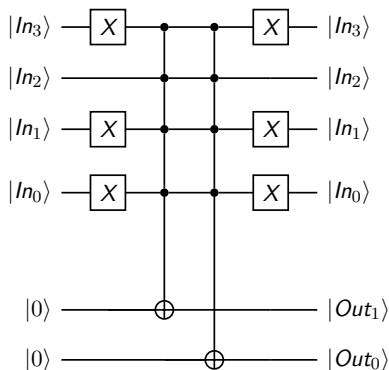
For each possibility in the S-box input, use a circuit similar to this:



...which requires 3 ancilla qubits.

# Dealing with S-boxes: The Brute-force Method

Thanks to the Microsoft Q# library, we don't need that much ancilla qubits.



No (explicit) ancilla qubits are required!

# Dealing with S-boxes: Combining the two methods?

- Still a hypothetical yet.
- The structure of Quine-McCluskey simplified S-Box is similar to Brute-force
- The number of terms may decrease significantly, and one ancilla bit may be removed compared to the first circuit.
- Needs further testing...

# Quantum S-DES Implementation

# Quantum S-DES Implementation

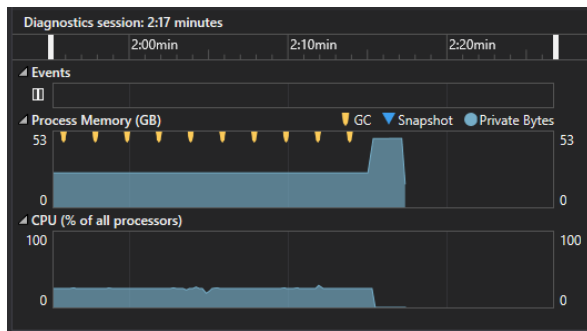
Initially, 34 explicit qubits were used in total.

- 10 qubits for input key
- 1 qubit for making an encryption oracle
- 8 qubits for (intermediate) plaintext
- 8 qubits for expanded plaintext
- 4 qubits for storing S-Box result
- 3 qubit for S-Box ancilla

S-DES implementation is written in Microsoft Q#. Will it run?

# Quantum S-DES Implementation

Yes, and no.



After allocating 48GB of RAM,  
`System.Runtime.InteropServices.SEHException` was raised,  
indicating an out-of-memory error.

# Quantum S-DES Implementation

Reducing the number of qubits was necessary...!

- 10 qubits for input key
- 1 qubit for making an encryption oracle
- 8 qubits for (intermediate) plaintext
- 8 qubits for expanded plaintext
- 4 qubits for storing S-Box result
- 3 qubits for S-Box ancilla



# Quantum S-DES Implementation

Reducing the number of qubits was necessary...!

- 10 qubits for input key
- 1 qubit for making an encryption oracle
- 8 qubits for (intermediate) plaintext
- ~~8 qubits for expanded plaintext~~
- 4 qubits for storing S-Box result
- 3 qubits for S-Box ancilla

We can directly use plaintext qubits to create result qubits from S-Box. After then we can undo all the operations (permutation, *etc.*) which altered plaintext.

# Quantum S-DES Implementation

Reducing the number of qubits was necessary...!

- 10 qubits for input key
- 1 qubit for making an encryption oracle
- 8 qubits for (intermediate) plaintext
- ~~8 qubits for expanded plaintext~~
- 4 2 qubits for storing S-Box result
- 3 qubits for S-Box ancilla

We can directly use plaintext qubits to create result qubits from S-Box. After then we can undo all the operations (permutation, *etc.*) which altered plaintext.

Additionally, each S-Box operation is independent. Thus we can reuse S-box result qubits after re-initialization.

# Quantum S-DES Implementation

Reducing the number of qubits was necessary...!

- 10 qubits for input key
- 1 qubit for making an encryption oracle
- 8 qubits for (intermediate) plaintext
- ~~8 qubits for expanded plaintext~~
- 4 2 qubits for storing S-Box result
- ~~3 qubits for S-Box ancilla~~

We can directly use plaintext qubits to create result qubits from S-Box. After then we can undo all the operations (permutation, *etc.*) which altered plaintext.

Additionally, each S-Box operation is independent. Thus we can reuse S-box result qubits after re-initialization.

Finally, redundant S-Box ancilla qubits were removed.

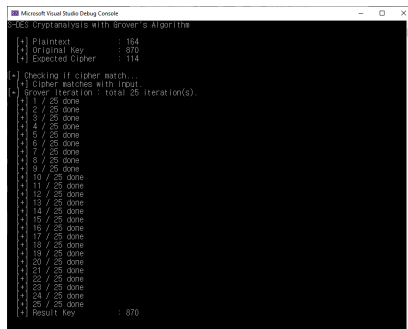
# Quantum S-DES Implementation

However, applying Grover's was not simple as it looked

- The oracle should be adjoint, but original version includes measurement (in result ciphertext and S-Box)
- How to create an oracle qubit rather than measuring the result ciphertext?
  - perform CNOT gate 8 times (with 7 more qubits) : infeasible
  - Microsoft Q#'s Controlled functor : doable
- Similar to S-Box Controlling
- S-Box also required measurement initially, but change to forementioned circuit

Result : reversible S-DES oracle.

# Quantum S-DES Implementation



```
Microsoft Visual Studio Debug Console
S-DES Cryptanalysis with Grover's Algorithm

[+] Plaintext      : 164
[+] Original Key   : 870
[+] Expected Cipher : 114

[+] Checking if cipher match...
[+] Cipher matches with input.
[+] Grover Iteration : total 25 iteration(s).
[+] 1 / 25 done
[+] 2 / 25 done
[+] 3 / 25 done
[+] 4 / 25 done
[+] 5 / 25 done
[+] 6 / 25 done
[+] 7 / 25 done
[+] 8 / 25 done
[+] 9 / 25 done
[+] 10 / 25 done
[+] 11 / 25 done
[+] 12 / 25 done
[+] 13 / 25 done
[+] 14 / 25 done
[+] 15 / 25 done
[+] 16 / 25 done
[+] 17 / 25 done
[+] 18 / 25 done
[+] 19 / 25 done
[+] 20 / 25 done
[+] 21 / 25 done
[+] 22 / 25 done
[+] 23 / 25 done
[+] 24 / 25 done
[+] 25 / 25 done
[+] Result Key      : 870
```

Expected key matched with original key (with runtime 19 min 31 sec, performing each S-DES encryption within 45s on average).

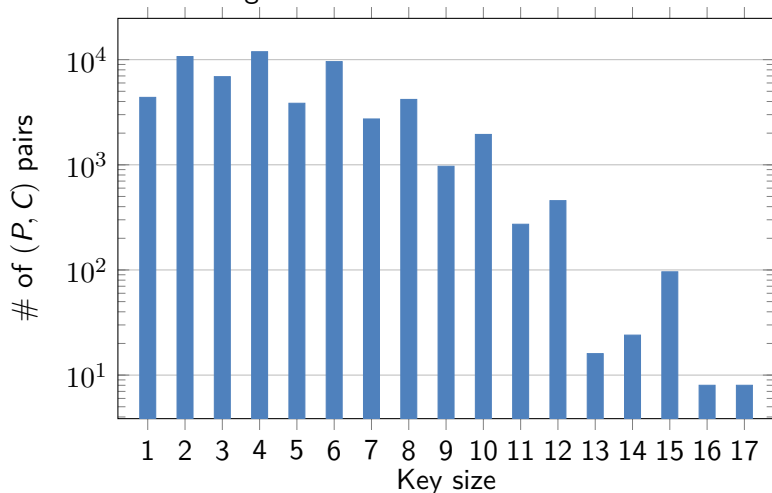
## Quantum S-DES Testing

# Prerequisites

- Let  $SDES(P, K) = C$  be a function s.t. encrypting plaintext  $P$  with S-DES and key  $K$  yields  $C$ .
- Let  $f: \{0, 1\}^8 \times \{0, 1\}^8 \rightarrow \mathcal{P}(\{0, 1\}^{10})$  be a function such that  $f(P, C) = \{K \in \{0, 1\}^{10} \mid SDES(P, K) = C\}$ .
- Since key size is 4x of cipher size, we guessed that for arbitrary  $P$  and  $C$ ,  $|f(P, C)|$  is likely to be 4.

# Prerequisites : Key Size

...and we were wrong!





# Quantum S-DES Testing : Testing with $\#Key = 1$

If  $(P, C) = (10100100, 00110011)$ . Then  $f(P, C) = \{1101100110\}$ .

- Program yielded correct key.
- By dumping key qubits, we can observe the probability of each basis.
  - At first, the probability of answer basis is 0.008766 while the others are 0.000969.
  - The gap widens next turn: 0.024224 and 0.000954.
  - Ultimately it becomes 0.999461 and 0.000001 (at 25th iteration).

## Quantum S-DES Testing : Testing with $\#Key = 4$

If  $(P, C) = (11000111, 00010100)$ . Then  $|f(P, C)| = 4$ .

- Program yielded wrong key.
- By dumping key qubits, we can observe the probability of each basis.
  - At first, the probability of answer basis is 0.008698 while the others are 0.000946.
  - The gap widens next turn: 0.023659 and 0.000888.
  - It peaks at 12th iteration: 0.249987 and 0.000000.
  - However it goes back next turn: 0.246547 and 0.000014
  - Ultimately the probability of answer basis becomes lower than the others: 0.000575 and 0.000978.
- Although the algorithm and the implementation is correct, what happened here?

# Quantum S-DES Testing: What Went Wrong?

- When running Grover's Algorithm, there is a number of desired steps, say  $n$ .
- If we overshoot that number of step, the probability of basis decreases!
- On the  $2n$ -th step, the probability becomes all the same for each of the cases again, forming a cycle.

## Quantum S-DES Gate Analysis

# Overview

- Most part can be expressed in permutations, but there are parts where we need to use quantum gates.
- In S-Box, we inevitably need to use Pauli-X and Toffoli gates.
- In EP, we need to copy the bits, and thereby requires the use of Toffoli gates.
- In round application, we need to XOR the subkey with some data or S-box result, requiring us to use CNOT gates.

# Number of Gates: S-Box

- Both S-box requires certain number of Pauli-X and Toffoli(CCNOT) gates.
- The sum of gates required for each of the 16 inputs possible:

	Pauli-X	Toffoli
S-box 1	50	18
S-box 2	44	15

- On average,  $\sim 3$  Pauli-X gates and  $\sim 1$  Toffoli gate is used.

# Number of Gates: S-Box

- What about Quine-McCluskey Method?
- Looking at S-box 1:
  - S1 bit 1:  $AB'C' + A'B'C + BCD' + ABD + BC'D$
  - S1 bit 2:  $AD' + B'D' + A'BC + ABC'$
- S1 requires 7 OR operations, all of which must be done with Toffoli gates.
- Compared to the brute-force method, this takes even more gates!

## Number of Gates: S-Box Input and Output

- Assume 3 Pauli-X gates and 1 Toffoli gate per single S-box calculation.
  - For input, 4 CNOT gates are required.
  - To copy the output, 2 more CNOT gates are required.
  - To make this reversible, S-box application and input processes must be reversed, requiring 2 S-Box applications and 8 CNOT gates.
- ⇒ Total of 10 CNOT gates, 6 Pauli-X gates, and 2 Toffoli gates.



# Number of Gates: Apply Rounds

- Two S-Boxes are applied per round.
    - The remaining operations are all permutations.
  - There are two rounds in total.
- ⇒ Total of 40 CNOT gates, 24 Pauli-X gates, and 8 Toffoli gates per single oracle call!