Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Кафедра програмної інженерії

Лабораторна робота №1
з дисципліни: «Безпека програм та даних»
на тему:
«СТВОРЕННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АЛГОРИТМУ ШИФРУВАННЯ des»

Виконав:
ст. гр. ПЗПІ-22-2
Шелехань Олександр

Перевірив:
ст. викл. кафедри ПІ
Олійник О. О.

Харків 2025

МЕТА РОБОТИ

Ознайомитись з методами і засобами симетричної криптографії, отримати навички створювання програмних засобів з використанням криптографічних інтерфейсів.

ХІД РОБОТИ

Код програми
Файл gui.py:

```
import tkinter as tk
from tkinter import ttk, scrolledtext, messagebox
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
from des import DES

class DESApp:
    def __init__(self, root):
        self.root = root
        self.root.title("DES Шифрування")
        self.root.geometry("1000x800")

        self.last_entropies = []
        self.des_instance = None

        self._create_widgets()

    def _create_widgets(self):
        # Головний контейнер
        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E,
tk.N, tk.S))

        # Налаштування розмірів
        self.root.columnconfigure(0, weight=1)
        self.root.rowconfigure(0, weight=1)
        main_frame.columnconfigure(0, weight=1)

        # Заголовок
        title_label = ttk.Label(
            main_frame,
            text="DES Шифрування",
            font=('Arial', 16, 'bold')
        )
        title_label.grid(row=0, column=0, pady=(0, 10))

        # Notebook для вкладок
        self.notebook = ttk.Notebook(main_frame)
```

```
            self.notebook.grid(row=1, column=0, sticky=(tk.W, tk.E,
tk.N, tk.S))
            main_frame.rowconfigure(1, weight=1)

            # Вкладка шифрування/розшифрування
            self.crypto_frame = ttk.Frame(self.notebook,
padding="10")
            self.notebook.add(self.crypto_frame,
text="Шифрування/Розшифрування")

            # Вкладка ентропії
            self.entropy_frame = ttk.Frame(self.notebook,
padding="10")
            self.notebook.add(self.entropy_frame, text="Аналіз
ентропії")

            # Заповнення вкладок
            self._create_crypto_tab()
            self._create_entropy_tab()

        def _create_crypto_tab(self):
            """Створення вкладки шифрування/розшифрування"""

            # Поле для ключа
            key_frame = ttk.LabelFrame(self.crypto_frame, text="Ключ
(8 символів)", padding="10")
            key_frame.grid(row=0, column=0, sticky=(tk.W, tk.E),
pady=5)
            self.crypto_frame.columnconfigure(0, weight=1)

            self.key_entry = ttk.Entry(key_frame, width=40,
font=('Courier', 10))
            self.key_entry.grid(row=0, column=0, sticky=(tk.W, tk.E))
            self.key_entry.insert(0, "DESCRYPT")
            key_frame.columnconfigure(0, weight=1)

            # Кнопка перевірки ключа
            check_key_btn = ttk.Button(
                key_frame,
                text="Перевірити ключ",
                command=self._check_key
            )
            check_key_btn.grid(row=0, column=1, padx=(10, 0))

            # Поле для вхідного тексту
            input_frame = ttk.LabelFrame(self.crypto_frame,
text="Вхідний текст", padding="10")
            input_frame.grid(row=1, column=0, sticky=(tk.W, tk.E,
tk.N, tk.S), pady=5)
            self.crypto_frame.rowconfigure(1, weight=1)

            self.input_text = scrolledtext.ScrolledText(
```

```
                input_frame,
                height=8,
                font=('Courier', 10),
                wrap=tk.WORD
            )
            self.input_text.grid(row=0, column=0, sticky=(tk.W, tk.E,
tk.N, tk.S))
            self.input_text.insert(1.0, "Hello, DES!")
            input_frame.columnconfigure(0, weight=1)
            input_frame.rowconfigure(0, weight=1)

            # Кнопки
            buttons_frame = ttk.Frame(self.crypto_frame)
            buttons_frame.grid(row=2, column=0, pady=10)

            encrypt_btn = ttk.Button(
                buttons_frame,
                text="Зашифрувати",
                command=self._encrypt,
                width=20
            )
            encrypt_btn.grid(row=0, column=0, padx=5)

            decrypt_btn = ttk.Button(
                buttons_frame,
                text="Розшифрувати",
                command=self._decrypt,
                width=20
            )
            decrypt_btn.grid(row=0, column=1, padx=5)

            clear_btn = ttk.Button(
                buttons_frame,
                text="Очистити",
                command=self._clear_fields,
                width=20
            )
            clear_btn.grid(row=0, column=2, padx=5)

            # Поле для вихідного тексту
            output_frame = ttk.LabelFrame(self.crypto_frame,
text="Результат", padding="10")
            output_frame.grid(row=3, column=0, sticky=(tk.W, tk.E,
tk.N, tk.S), pady=5)
            self.crypto_frame.rowconfigure(3, weight=1)

            self.output_text = scrolledtext.ScrolledText(
                output_frame,
                height=8,
                font=('Courier', 10),
                wrap=tk.WORD,
                state=tk.DISABLED
```

```
            )
            self.output_text.grid(row=0, column=0, sticky=(tk.W,
tk.E, tk.N, tk.S))
            output_frame.columnconfigure(0, weight=1)
            output_frame.rowconfigure(0, weight=1)

        def _create_entropy_tab(self):
            """Створення вкладки аналізу ентропії"""

            # Фрейм для графіка
            self.figure = Figure(figsize=(8, 6), dpi=100)
            self.ax = self.figure.add_subplot(111)

            self.canvas = FigureCanvasTkAgg(self.figure,
self.entropy_frame)
            self.canvas.get_tk_widget().grid(row=0, column=0,
sticky=(tk.W, tk.E, tk.N, tk.S))

            self.entropy_frame.columnconfigure(0, weight=1)
            self.entropy_frame.rowconfigure(0, weight=1)

            # Початковий порожній графік
            self._plot_entropy([])

            # Таблиця з даними
            table_frame = ttk.LabelFrame(self.entropy_frame,
text="Детальна інформація", padding="10")
            table_frame.grid(row=1, column=0, sticky=(tk.W, tk.E),
pady=10)

            # Створення таблиці
            columns = ('Раунд', 'Ентропія', 'Одиниць', 'Нулів',
'Ймовірність 1')
            self.entropy_table = ttk.Treeview(table_frame,
columns=columns, show='headings', height=6)

            for col in columns:
                self.entropy_table.heading(col, text=col)
                self.entropy_table.column(col, width=100,
anchor=tk.CENTER)

            scrollbar = ttk.Scrollbar(table_frame,
orient=tk.VERTICAL, command=self.entropy_table.yview)
            self.entropy_table.configure(yscroll=scrollbar.set)

            self.entropy_table.grid(row=0, column=0, sticky=(tk.W,
tk.E, tk.N, tk.S))
            scrollbar.grid(row=0, column=1, sticky=(tk.N, tk.S))

            table_frame.columnconfigure(0, weight=1)

        def _check_key(self):
```

```
            """Перевірка ключа на слабкість"""
            key_str = self.key_entry.get()

            if len(key_str) != 8:
                messagebox.showwarning(
                    "Попередження",
                    "Ключ повинен складатися рівно з 8 символів!"
                )
                return

            key_bytes = key_str.encode('utf-8')

            if DES.is_weak_key(key_bytes):
                messagebox.showerror(
                    "Слабкий ключ!",
                    "Цей ключ є слабким та небезпечним.\nВиберіть
інший ключ."
                )
            else:
                messagebox.showinfo(
                    "Перевірка пройдена",
                    "Ключ коректний."
                )

        def _encrypt(self):
            """Шифрування тексту"""
            try:
                # Отримання ключа
                key_str = self.key_entry.get()
                if len(key_str) != 8:
                    messagebox.showerror("Помилка", "Ключ повинен
бути 8 символів!")
                    return

                key_bytes = key_str.encode('utf-8')

                # Отримання тексту
                plaintext = self.input_text.get(1.0, tk.END).strip()
                if not plaintext:
                    messagebox.showerror("Помилка", "Введіть текст
для шифрування!")
                    return

                plaintext_bytes = plaintext.encode('utf-8')

                # Шифрування
                self.des_instance = DES(key_bytes)
                ciphertext =
self.des_instance.encrypt(plaintext_bytes)

                # Збереження ентропій
                self.last_entropies =
```

```
self.des_instance.get_last_round_entropies()

                # Відображення результату
                self.output_text.config(state=tk.NORMAL)
                self.output_text.delete(1.0, tk.END)
                self.output_text.insert(1.0, ciphertext.hex())
                self.output_text.config(state=tk.DISABLED)

                # Оновлення графіка ентропії
                self._plot_entropy(self.last_entropies)
                self._update_entropy_table(self.last_entropies)

            except ValueError as e:
                messagebox.showerror("Помилка", str(e))
            except Exception as e:
                messagebox.showerror("Помилка", f"Виникла помилка:
{str(e)}")

        def _decrypt(self):
            """Розшифрування тексту"""
            try:
                # Отримання ключа
                key_str = self.key_entry.get()
                if len(key_str) != 8:
                    messagebox.showerror("Помилка", "Ключ повинен
бути 8 символів!")
                    return

                key_bytes = key_str.encode('utf-8')

                # Отримання зашифрованого тексту
                ciphertext_hex = self.input_text.get(1.0,
tk.END).strip()
                if not ciphertext_hex:
                    messagebox.showerror("Помилка", "Введіть HEX-
текст для розшифрування!")
                    return

                # Видалення можливих пробілів
                ciphertext_hex = ciphertext_hex.replace(" ",
"").replace("\n", "")

                try:
                    ciphertext = bytes.fromhex(ciphertext_hex)
                except ValueError:
                    messagebox.showerror("Помилка", "Невірний HEX
формат!")
                    return

                # Розшифрування
                des = DES(key_bytes)
                plaintext = des.decrypt(ciphertext)
```

```
                # Відображення результату
                self.output_text.config(state=tk.NORMAL)
                self.output_text.delete(1.0, tk.END)
                self.output_text.insert(1.0, plaintext.decode('utf-
8'))
                self.output_text.config(state=tk.DISABLED)

            except ValueError as e:
                messagebox.showerror("Помилка", str(e))
            except Exception as e:
                messagebox.showerror("Помилка", f"Виникла помилка:
{str(e)}")

        def _clear_fields(self):
            """Очищення полів"""
            self.input_text.delete(1.0, tk.END)
            self.output_text.config(state=tk.NORMAL)
            self.output_text.delete(1.0, tk.END)
            self.output_text.config(state=tk.DISABLED)

        def _plot_entropy(self, entropies):
            """Побудова графіка ентропії"""
            self.ax.clear()

            if not entropies:
                self.ax.text(
                    0.5, 0.5,
                    'Виконайте шифрування для відображення графіка',
                    ha='center', va='center',
                    fontsize=12
                )
                self.ax.set_xlim(0, 1)
                self.ax.set_ylim(0, 1)
            else:
                rounds = [e['round'] for e in entropies]
                entropy_values = [e['entropy'] for e in entropies]

                self.ax.plot(rounds, entropy_values, 'b-o',
linewidth=2, markersize=6)
                self.ax.axhline(y=1.0, color='r', linestyle='--',
linewidth=1, label='Максимальна ентропія')
                self.ax.set_xlabel('Номер раунду', fontsize=11)
                self.ax.set_ylabel('Ентропія (біти)', fontsize=11)
                self.ax.set_title('Ентропія появи біту 1 на кожному
раунді', fontsize=12, fontweight='bold')
                self.ax.grid(True, alpha=0.3)
                self.ax.legend()
                self.ax.set_xlim(0, 17)
                self.ax.set_ylim(0, 1.1)

            self.canvas.draw()
```

```
    def _update_entropy_table(self, entropies):
        """Оновлення таблиці з даними ентропії"""
        # Очистка таблиці
        for item in self.entropy_table.get_children():
            self.entropy_table.delete(item)

        # Заповнення таблиці
        for e in entropies:
            prob_one = e['ones'] / (e['ones'] + e['zeros'])
            self.entropy_table.insert('', tk.END, values=(
                e['round'],
                f"{e['entropy']:.6f}",
                e['ones'],
                e['zeros'],
                f"{prob_one:.4f}"
            ))

def main():
    """Запуск програми"""
    root = tk.Tk()
    app = DESApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Файл des.py:

```
import math
from typing import List
from des_tables import *

class DES:
    def __init__(self, key: bytes):
        if len(key) != 8:
            raise ValueError("Ключ повинен бути 8 байт")

        self.key = key
        self.round_keys = []
        self.round_entropies = []

        if self.is_weak_key(key):
            raise ValueError("Слабкий ключ! Виберіть інший.")

        self.round_keys = self._generate_round_keys()

    @staticmethod
    def is_weak_key(key: bytes) -> bool:
        """Перевірка на слабкі ключі"""
        key_int = int.from_bytes(key, byteorder='big')
```

```
        if key_int in WEAK_KEYS:
            return True

        for pair in SEMI_WEAK_KEYS:
            if key_int == pair[0] or key_int == pair[1]:
                return True

        return False

    @staticmethod
    def _bytes_to_bits(data: bytes) -> List[int]:
        """Конвертація байтів у біти"""
        bits = []
        for byte in data:
            for i in range(7, -1, -1):
                bits.append((byte >> i) & 1)
        return bits

    @staticmethod
    def _bits_to_bytes(bits: List[int]) -> bytes:
        """Конвертація бітів у байти"""
        byte_array = []
        for i in range(0, len(bits), 8):
            byte = 0
            for j in range(8):
                if i + j < len(bits):
                    byte = (byte << 1) | bits[i + j]
            byte_array.append(byte)
        return bytes(byte_array)

    @staticmethod
    def _permute(bits: List[int], table: List[int]) -> List[int]:
        """Перестановка бітів згідно таблиці"""
        return [bits[i - 1] for i in table]

    @staticmethod
    def _left_shift(bits: List[int], n: int) -> List[int]:
        """Циклічний зсув вліво"""
        return bits[n:] + bits[:n]

    @staticmethod
    def _xor(bits1: List[int], bits2: List[int]) -> List[int]:
        """XOR двох списків бітів"""
        return [b1 ^ b2 for b1, b2 in zip(bits1, bits2)]

    @staticmethod
    def _calculate_entropy(bits: List[int]) -> float:
        """Обчислення ентропії"""
        if not bits:
            return 0.0
```

```
        ones = sum(bits)
        zeros = len(bits) - ones

        p1 = ones / len(bits) if len(bits) > 0 else 0
        p0 = zeros / len(bits) if len(bits) > 0 else 0

        entropy = 0.0
        if p0 > 0:
            entropy -= p0 * math.log2(p0)
        if p1 > 0:
            entropy -= p1 * math.log2(p1)

        return entropy

    def _generate_round_keys(self) -> List[List[int]]:
        """Генерація 16 раундових ключів"""
        key_bits = self._bytes_to_bits(self.key)

        # PC1
        key_56 = self._permute(key_bits, PC1)

        # Розділення на C і D
        C = key_56[:28]
        D = key_56[28:]

        round_keys = []

        for i in range(16):
            # Зсув
            C = self._left_shift(C, SHIFTS[i])
            D = self._left_shift(D, SHIFTS[i])

            # PC2
            CD = C + D
            round_key = self._permute(CD, PC2)
            round_keys.append(round_key)

        return round_keys

    def _s_box_substitution(self, bits: List[int]) -> List[int]:
        """Підстановка через S-boxes"""
        result = []

        for i in range(8):
            block = bits[i * 6:(i + 1) * 6]

            row = (block[0] << 1) | block[5]
            col = (block[1] << 3) | (block[2] << 2) | (block[3]
<< 1) | block[4]

            val = S_BOXES[i][row][col]
```

```
                for j in range(3, -1, -1):
                    result.append((val >> j) & 1)

            return result

        def _f_function(self, right: List[int], round_key: List[int])
-> List[int]:
            """Функція f мережі Фейстеля"""
            # Розширення E
            expanded = self._permute(right, E)

            # XOR з ключем
            xored = self._xor(expanded, round_key)

            # S-boxes
            substituted = self._s_box_substitution(xored)

            # Перестановка P
            result = self._permute(substituted, P)

            return result

        def _des_block(self, block: bytes, decrypt: bool = False) ->
bytes:
            """Шифрування/розшифрування одного блоку"""
            bits = self._bytes_to_bits(block)

            # Початкова перестановка
            bits = self._permute(bits, IP)

            left = bits[:32]
            right = bits[32:]

            self.round_entropies = []

            keys = self.round_keys[::-1] if decrypt else
self.round_keys

            # 16 раундів
            for i in range(16):
                old_right = right[:]

                f_result = self._f_function(right, keys[i])
                right = self._xor(left, f_result)
                left = old_right

                # Розрахунок ентропії
                entropy = self._calculate_entropy(right)
                self.round_entropies.append({
                    'round': i + 1,
                    'entropy': entropy,
                    'ones': sum(right),
```

```
                    'zeros': len(right) - sum(right)
                })

            # Кінцева перестановка
            combined = right + left
            final_bits = self._permute(combined, IP_INV)

            return self._bits_to_bytes(final_bits)

        def encrypt(self, plaintext: bytes) -> bytes:
            """Шифрування"""
            # Padding
            padding_length = 8 - (len(plaintext) % 8)
            plaintext = plaintext + bytes([padding_length] *
padding_length)

            ciphertext = b''
            for i in range(0, len(plaintext), 8):
                block = plaintext[i:i + 8]
                ciphertext += self._des_block(block, decrypt=False)

            return ciphertext

        def decrypt(self, ciphertext: bytes) -> bytes:
            """Розшифрування"""
            if len(ciphertext) % 8 != 0:
                raise ValueError("Неправильна довжина")

            plaintext = b''
            for i in range(0, len(ciphertext), 8):
                block = ciphertext[i:i + 8]
                plaintext += self._des_block(block, decrypt=True)

            # Видалення padding
            padding_length = plaintext[-1]
            plaintext = plaintext[:-padding_length]

            return plaintext

        def get_last_round_entropies(self) -> List[dict]:
            """Отримання ентропій"""
            return self.round_entropies

    def main():
        """Тест"""
        key = b'DESCRYPT'
        plaintext = b'Hello, DES!'

        print("=" * 60)
        print("Тест DES")
        print("=" * 60)
        print(f"Ключ: {key}")
```

```
    print(f"Текст: {plaintext}")
    print()

    try:
        des = DES(key)

        ciphertext = des.encrypt(plaintext)
        print(f"Шифротекст (hex): {ciphertext.hex()}")
        print()

        print("Ентропія по раундах:")
        print("-" * 60)
        for info in des.get_last_round_entropies():
            print(f"Раунд {info['round']:2d}: "
                  f"Ентропія = {info['entropy']:.6f}, "
                  f"1 = {info['ones']:2d}, "
                  f"0 = {info['zeros']:2d}")
        print()

        decrypted = des.decrypt(ciphertext)
        print(f"Розшифроване: {decrypted}")
        print()

        if plaintext == decrypted:
            print("Тест пройдено!")
        else:
            print("Помилка!")

    except ValueError as e:
        print(f"Помилка: {e}")

if __name__ == "__main__":
    main()
```

Файл des_tables.py:

```
# Початкова перестановка
IP = [
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17, 9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
]

# Кінцева перестановка
IP_INV = [
    40, 8, 48, 16, 56, 24, 64, 32,
    39, 7, 47, 15, 55, 23, 63, 31,
```

```
    38, 6, 46, 14, 54, 22, 62, 30,
    37, 5, 45, 13, 53, 21, 61, 29,
    36, 4, 44, 12, 52, 20, 60, 28,
    35, 3, 43, 11, 51, 19, 59, 27,
    34, 2, 42, 10, 50, 18, 58, 26,
    33, 1, 41, 9, 49, 17, 57, 25
]

# Таблиця розширення E
E = [
    32, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32, 1
]

# Перестановка P
P = [
    16, 7, 20, 21, 29, 12, 28, 17,
    1, 15, 23, 26, 5, 18, 31, 10,
    2, 8, 24, 14, 32, 27, 3, 9,
    19, 13, 30, 6, 22, 11, 4, 25
]

# PC1
PC1 = [
    57, 49, 41, 33, 25, 17, 9,
    1, 58, 50, 42, 34, 26, 18,
    10, 2, 59, 51, 43, 35, 27,
    19, 11, 3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7, 62, 54, 46, 38, 30, 22,
    14, 6, 61, 53, 45, 37, 29,
    21, 13, 5, 28, 20, 12, 4
]

# PC2
PC2 = [
    14, 17, 11, 24, 1, 5,
    3, 28, 15, 6, 21, 10,
    23, 19, 12, 4, 26, 8,
    16, 7, 27, 20, 13, 2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
]
```

```
# Зсуви
SHIFTS = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

# S-boxes
S_BOXES = [
    # S1
    [
        [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
        [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
        [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
        [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]
    ],
    # S2
    [
        [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
        [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
        [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
        [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]
    ],
    # S3
    [
        [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
        [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
        [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
        [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]
    ],
    # S4
    [
        [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
        [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
        [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
        [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]
    ],
    # S5
    [
        [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
        [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
        [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
        [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]
    ],
    # S6
    [
        [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
        [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
        [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
        [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]
    ],
    # S7
    [
        [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
        [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
        [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
```

```
        [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]
    ],
    # S8
    [
        [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
        [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
        [7, 11, 4, 1, 9, 12, 14, 2, 0, 5, 10, 3, 13, 8, 15, 6],
        [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]
    ]
]

# Слабкі ключі
WEAK_KEYS = [
    0x0101010101010101,
    0xFEFEFEFEFEFEFEFE,
    0xE0E0E0E0F1F1F1F1,
    0x1F1F1F1F0E0E0E0E,
]

# Напів-слабкі ключі
SEMI_WEAK_KEYS = [
    (0x011F011F010E010E, 0x1F011F010E010E01),
    (0x01E001E001F101F1, 0xE001E001F101F101),
    (0x01FE01FE01FE01FE, 0xFE01FE01FE01FE01),
    (0x1FE01FE00EF10EF1, 0xE01FE01FF10EF10E),
    (0x1FFE1FFE0EFE0EFE, 0xFE1FFE1FFE0EFE0E),
    (0xE0FEE0FEF1FEF1FE, 0xFEE0FEE0FEF1FEF1),
]
```

Результати:

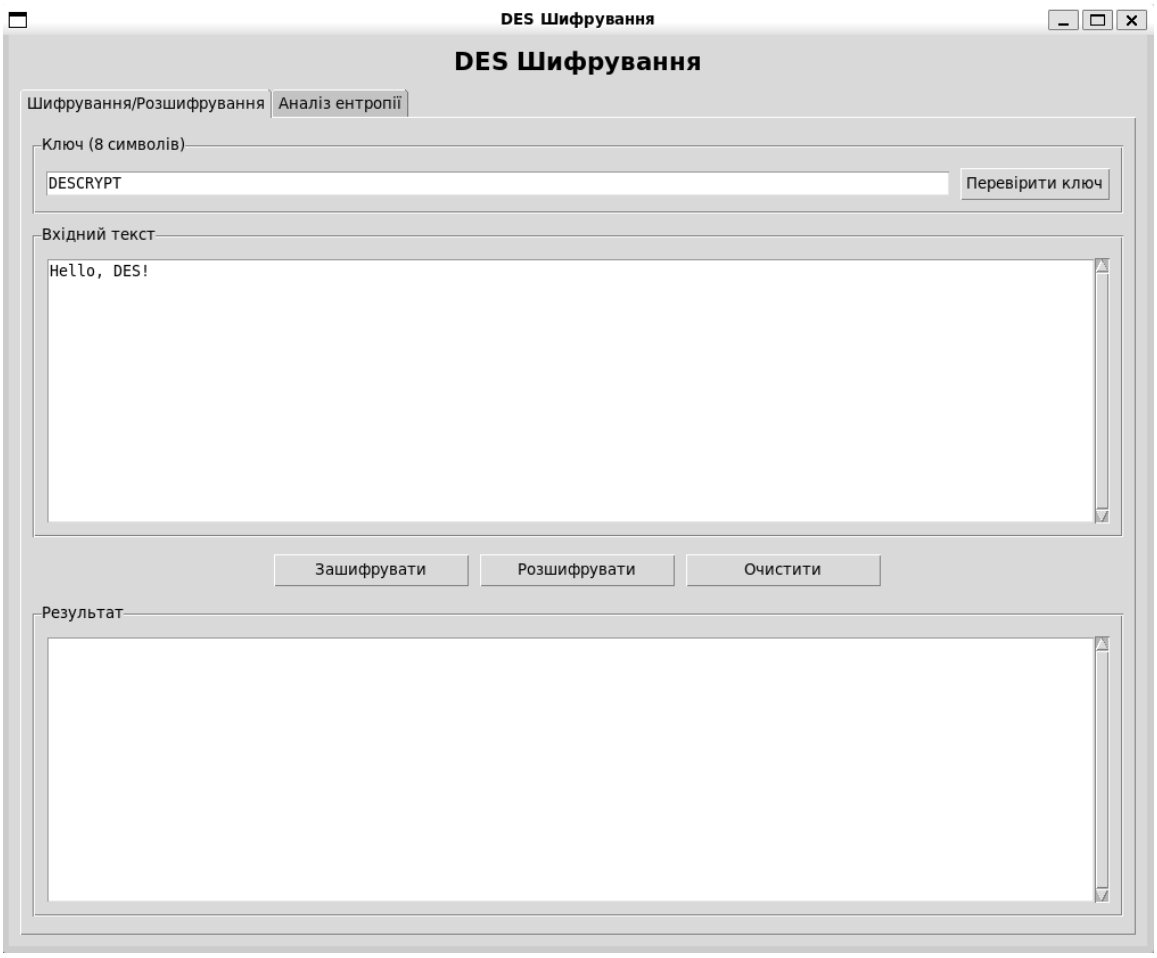


Рис. 1 – Інтерфейс програми

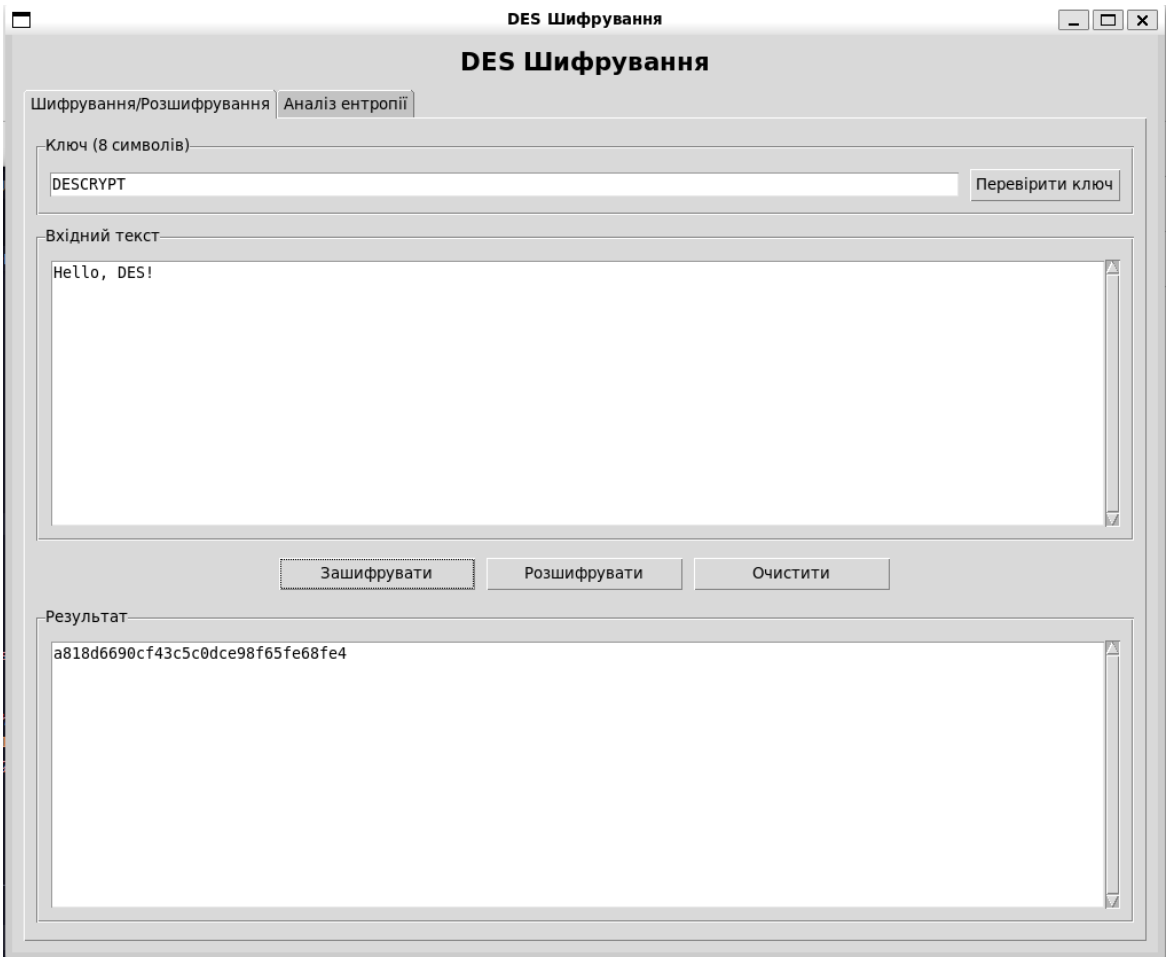


Рис. 2 – Шифрування тексту

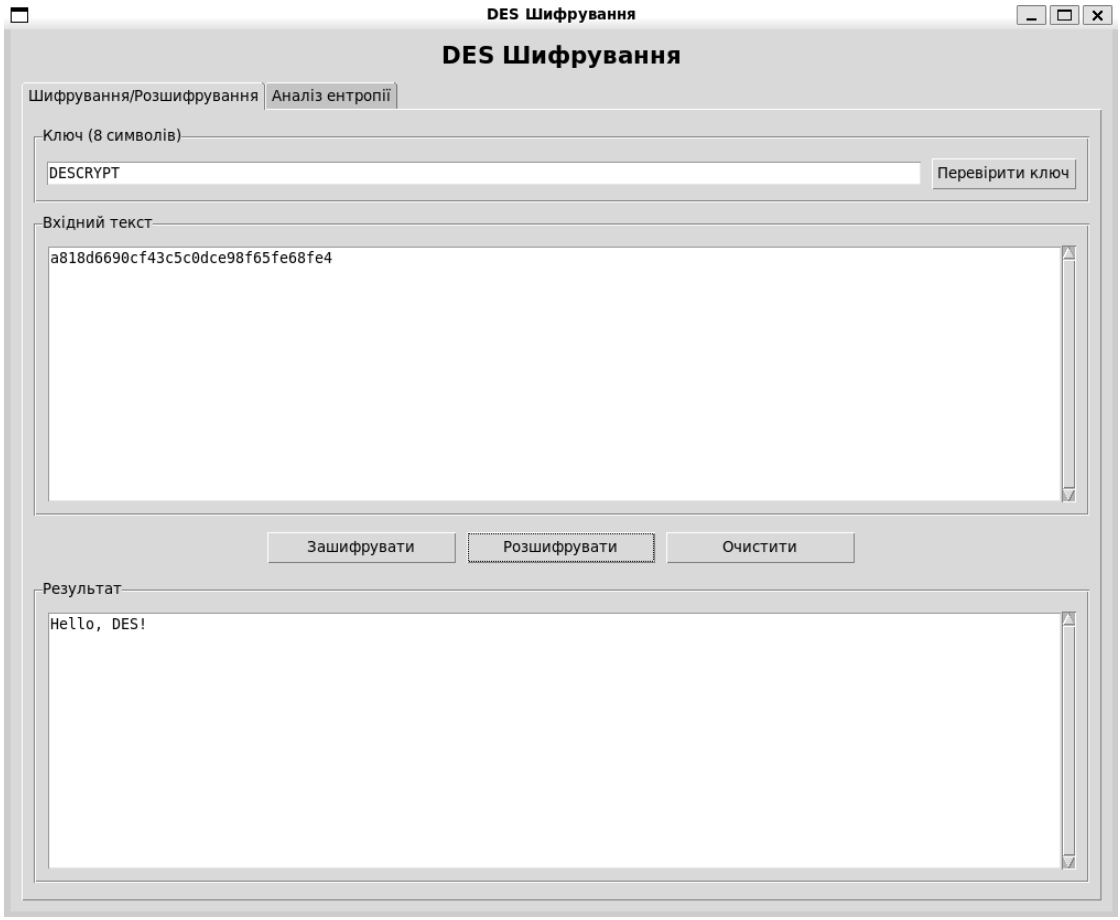
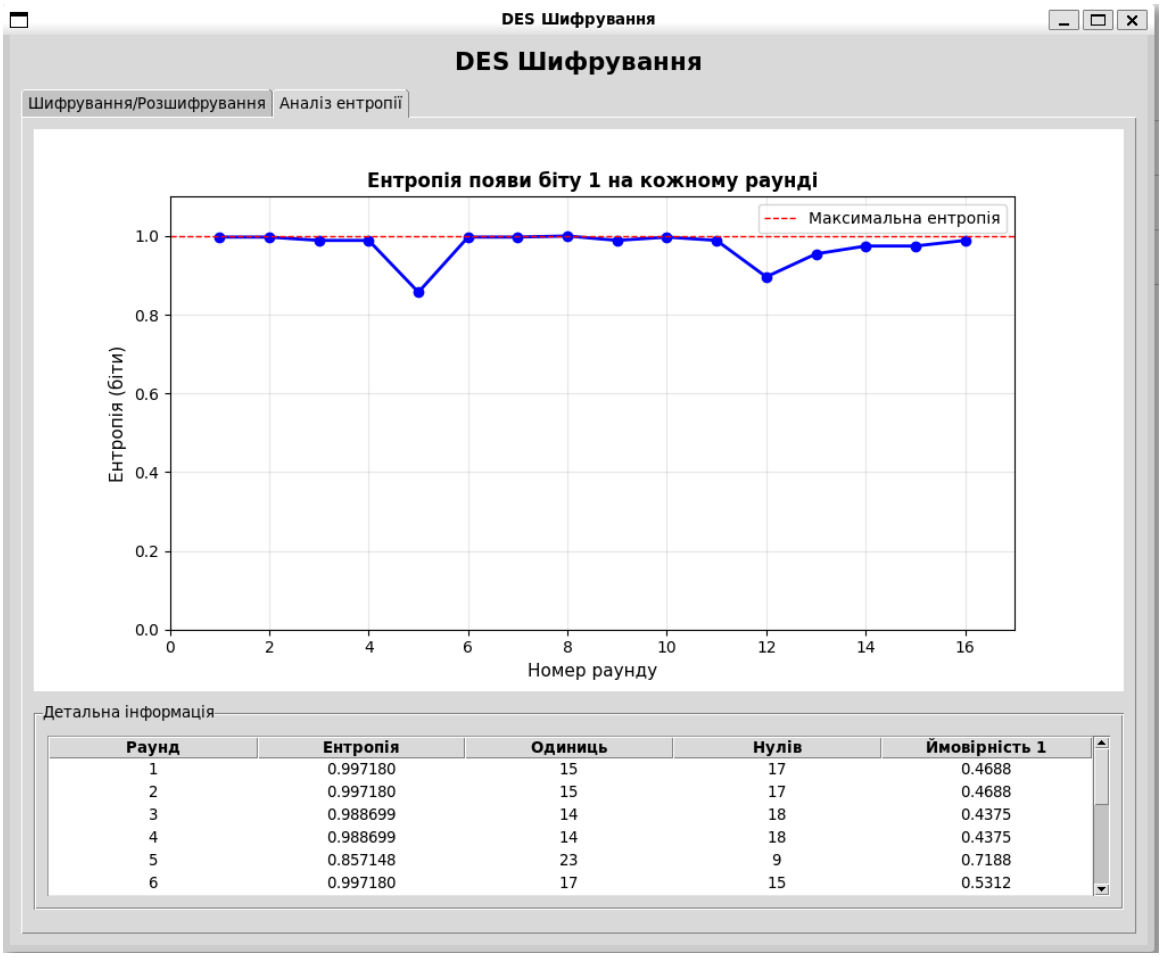


Рис. 3 – Розшифрування тексту



Раунд	Ентропія	Одиниць	Нулів	Ймовірність 1
1	0.997180	15	17	0.4688
2	0.997180	15	17	0.4688
3	0.988699	14	18	0.4375
4	0.988699	14	18	0.4375
5	0.857148	23	9	0.7188
6	0.997180	17	15	0.5312

Рис. 4 – Графік та таблиця ентропії

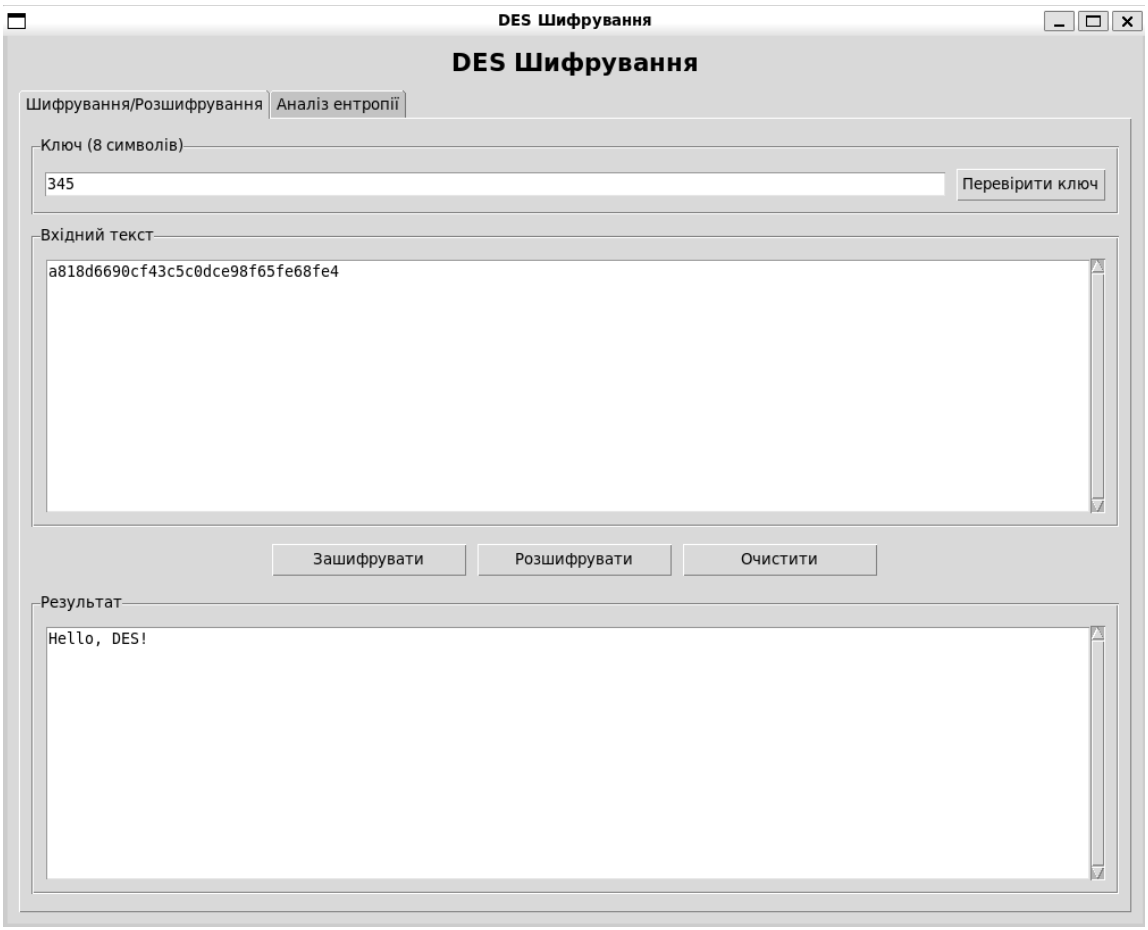


Рис. 5 – Введення ключа у поле

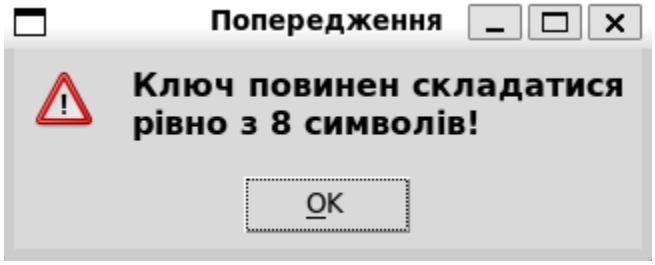


Рис. 6 – Перевірка ключа

Висновки: В ході виконання лабораторної роботи було створено програмну реалізацію алгоритму DES без використання готових крипто бібліотек. В результаті набуто практичних навичок програмування криптографічних алгоритмів та поглиблено розуміння принципів симетричного шифрування.