#### main.py

main.py调用Window类，负责初始化界面。

#### s\_des.py

s\_des.py中，在SimpleDes类中实现了Simple-DES算法，分别对外提供二进制数组，二进制字符，ASCLL字符的加解密及破解。

SimpleDes初始化。

def \_\_init\_\_(self):  
 self.ip\_box = [2, 6, 3, 1, 4, 8, 5, 7]  
 self.p\_box\_4 = [2, 4, 3, 1]  
 self.p\_box\_8 = [6, 3, 7, 4, 8, 5, 10, 9]  
 self.p\_box\_10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]  
 self.e\_p\_box = [4, 1, 2, 3, 2, 3, 4, 1]  
 self.s\_box\_1 = [[[0, 1], [0, 0], [1, 1], [1, 0]],  
 [[1, 1], [1, 0], [0, 1], [0, 0]],  
 [[0, 0], [1, 0], [0, 1], [1, 1]],  
 [[1, 1], [0, 1], [0, 0], [1, 0]]]  
 self.s\_box\_2 = [[[0, 0], [0, 1], [1, 0], [1, 1]],  
 [[1, 0], [0, 0], [0, 1], [1, 1]],  
 [[1, 1], [1, 0], [0, 1], [0, 0]],  
 [[1, 0], [0, 1], [0, 0], [1, 1]]]  
  
 self.message = [None] \* 8  
 self.cipher = [None] \* 8  
 self.key = [None] \* 10

算法实现。

@staticmethod  
def permute(message, key):  
 m = [message[k - 1] for k in key]  
 return m  
  
@staticmethod  
def inverse(key):  
 k = [key.index(i+1)+1 for i in range(len(key))]  
 return k  
  
@staticmethod  
def left\_shift(message, key):  
 k = key % len(message)  
 m = message[k:] + message[:k]  
 return m  
  
@staticmethod  
def s\_4\_2(message, key):  
 h = message[0] \* 2 + message[3]  
 v = message[1] \* 2 + message[2]  
 return key[h][v]  
  
@staticmethod  
def xor(arr\_1, arr\_2):  
 result = [a ^ b for a, b in zip(arr\_1, arr\_2)]  
 return result  
  
def creat\_sub\_key(self):  
 k = self.permute(self.key, self.p\_box\_10)  
 left = k[:5]  
 right = k[5:]  
  
 left = self.left\_shift(left, 1)  
 right = self.left\_shift(right, 1)  
 k\_1 = self.permute(left+right, self.p\_box\_8)  
  
 left = self.left\_shift(left, 1)  
 right = self.left\_shift(right, 1)  
 k\_2 = self.permute(left + right, self.p\_box\_8)  
  
 return k\_1, k\_2  
  
def f(self, message, key):  
 m = self.permute(message, self.e\_p\_box)  
 s = self.xor(m, key)  
 left = s[:4]  
 right = s[4:]  
 left = self.s\_4\_2(left, self.s\_box\_1)  
 right = self.s\_4\_2(right, self.s\_box\_2)  
 return self.permute(left+right, self.p\_box\_4)  
  
def swap(self, m, k\_1, k\_2):  
 left\_1 = m[:4]  
 left\_2 = m[4:]  
  
 right\_1 = self.f(left\_2, k\_1)  
 right\_3 = self.xor(left\_1, right\_1)  
  
 right\_2 = self.f(right\_3, k\_2)  
 left\_3 = self.xor(left\_2, right\_2)  
  
 i\_ip\_box = self.inverse(self.ip\_box)  
  
 return left\_3+right\_3, i\_ip\_box

8位二进制数组加解密及破解。

def encrypt(self)

def decrypt(self)

def crack(self, message, cipher)

多位二机制字符串加解密及破解。

def encrypt\_a(self, message)

def decrypt\_a(self, cipher)

def crack\_a(self, message, cipher)

多位ASCLL字符串加解密及破解。

def encrypt\_b(self, message)

def decrypt\_b(self, cipher)

def crack\_b(self, message, cipher)

#### window.py

#### window.py中，Entry类调用SimpleDes类，并为界面提供3个Entry控件，方便界面操作；

class Entry:  
 def \_\_init\_\_(self):  
 self.entry\_1 = tk.Entry  
 self.entry\_2 = tk.Entry  
 self.entry\_3 = tk.Entry  
 self.sdes = s\_des.SimpleDes()  
  
 def on\_button\_click(self, leaf):  
 # 在按钮点击时执行的函数  
 entry1\_value = self.entry\_1.get()  
 entry2\_value = self.entry\_2.get()  
  
 if leaf == 1:  
 result = self.sdes.set\_key\_b(entry2\_value)  
 if result == "密钥设置成功":  
 if entry1\_value.startswith("0b"):  
 result = self.sdes.encrypt\_b(entry1\_value)  
 else:  
 if entry1\_value.isascii():  
 result = self.sdes.encrypt\_a(entry1\_value)  
 else:  
 result = "明文包含错误字符"  
  
 self.entry\_3.config(state="normal")  
 self.entry\_3.delete(0, tk.END)  
 self.entry\_3.insert(0, result)  
 self.entry\_3.config(state="readonly")  
  
 elif leaf == 2:  
 result = self.sdes.set\_key\_b(entry2\_value)  
 if result == "密钥设置成功":  
 if entry1\_value.startswith("0b"):  
 result = self.sdes.decrypt\_b(entry1\_value)  
 else:  
 result = self.sdes.decrypt\_a(entry1\_value)  
  
 self.entry\_3.config(state="normal")  
 self.entry\_3.delete(0, tk.END)  
 self.entry\_3.insert(0, result)  
 self.entry\_3.config(state="readonly")  
  
 elif leaf == 3:  
 if entry1\_value.startswith("0b") & entry2\_value.startswith("0b"):  
 result, times = self.sdes.crack\_b(entry1\_value, entry2\_value)  
 else:  
 if len(entry1\_value) == len(entry2\_value):  
 result, times = self.sdes.crack\_a(entry1\_value, entry2\_value)  
 else:  
 result = "明密文不对应"  
  
 self.entry\_3.config(state="normal")  
 self.entry\_3.delete(0, tk.END)  
 self.entry\_3.insert(0, result)  
 self.entry\_3.config(state="readonly")  
  
 elif leaf == 4:  
 if entry1\_value.startswith("0b") & entry2\_value.startswith("0b"):  
 keys, result = self.sdes.crack\_b(entry1\_value, entry2\_value)  
 else:  
 if len(entry1\_value) == len(entry2\_value):  
 keys, result = self.sdes.crack\_a(entry1\_value, entry2\_value)  
 else:  
 result = "明密文不对应"  
  
 self.entry\_3.config(state="normal")  
 self.entry\_3.delete(0, tk.END)  
 self.entry\_3.insert(0, result)  
 self.entry\_3.config(state="readonly")

#### Window类通过Notebook和Frame创建不同功能的分页，并调用Entry类，用按钮绑定事件。

class Window:  
 def main(self):  
 # 创建主窗口  
 root = tk.Tk()  
 root.title("ttk.Notebook 示例")  
 root.geometry("400x300+560+280")  
  
 # 创建分页标签控件  
 notebook = ttk.Notebook(root)  
  
 # 创建第一个选项卡  
 tab\_1 = ttk.Frame(notebook)  
 notebook.add(tab\_1, text="加密")  
  
 # 创建第二个选项卡  
 tab\_2 = ttk.Frame(notebook)  
 notebook.add(tab\_2, text="解密")  
  
 tab\_3 = ttk.Frame(notebook)  
 notebook.add(tab\_3, text="破解")  
  
 tab\_4 = ttk.Frame(notebook)  
 notebook.add(tab\_4, text="碰撞")  
  
 # 将 ttk.Notebook 放置在主窗口中  
 notebook.pack()  
  
 self.page(tab\_1, 1)  
 self.page(tab\_2, 2)  
 self.page(tab\_3, 3)  
 self.page(tab\_4, 4)  
  
 # 放大 ttk.Notebook 以适应窗口大小  
 notebook.pack(fill=tk.BOTH, expand=True)  
  
 # 启动主循环  
 root.mainloop()  
  
 @staticmethod  
 def page(root, leaf):  
 entry = Entry()  
  
 # 设置行和列的权重  
 root.grid\_rowconfigure(0, weight=1)  
 root.grid\_columnconfigure(0, weight=1)  
  
 # 创建 ttk.Frame 实例  
 frame = ttk.Frame(root)  
 frame.grid(row=0, column=0, padx=10, pady=10)  
  
 # 创建输入框  
 label\_1 = tk.Label(frame, text="账户:")  
 label\_1.grid(row=0, column=0, padx=20, pady=10, )  
 entry.entry\_1 = ttk.Entry(frame)  
 entry.entry\_1.grid(row=0, column=1, padx=20, pady=10)  
  
 label\_2 = tk.Label(frame, text="账户:")  
 label\_2.grid(row=1, column=0, padx=20, pady=10)  
 entry.entry\_2 = ttk.Entry(frame)  
 entry.entry\_2.grid(row=1, column=1, padx=20, pady=10)  
  
 label\_3 = tk.Label(frame, text="账户:")  
 label\_3.grid(row=2, column=0, padx=20, pady=10)  
 entry.entry\_3 = ttk.Entry(frame, state="readonly")  
 entry.entry\_3.grid(row=2, column=1, padx=20, pady=10)  
  
 # 创建按钮  
 button = ttk.Button(root)  
 button.grid(row=1, column=0, padx=10, pady=10)  
  
 if leaf == 1:  
 label\_1.config(text="明文")  
 label\_2.config(text="密钥")  
 label\_3.config(text="密文")  
 button.config(text="加密", command=lambda: entry.on\_button\_click(leaf))  
 elif leaf == 2:  
 label\_1.config(text="密文")  
 label\_2.config(text="密钥")  
 label\_3.config(text="明文")  
 button.config(text="解密", command=lambda: entry.on\_button\_click(leaf))  
 elif leaf == 3:  
 label\_1.config(text="明文")  
 label\_2.config(text="密文")  
 label\_3.config(text="密钥")  
 button.config(text="破解", command=lambda: entry.on\_button\_click(leaf))  
 elif leaf == 4:  
 label\_1.config(text="明文")  
 label\_2.config(text="密文")  
 label\_3.config(text="次数")  
 button.config(text="碰撞", command=lambda: entry.on\_button\_click(leaf))