# S-AES开发手册

#### main.py

main.py调用Window类，负责初始化界面。

#### s\_aes.py

s\_aes.py中，在SimpleAes类中实现了Simple-AES算法，内部的非实例函数分别对外提供二进制数组，二进制字符串，ascii，gbk，gb2312，gb18030编码的加解密及破解。

在GF24类中实现gf(2^4)域上的的加法和乘法，并重载了加号，乘号和格式化输出。

GF24类

primitive = 0b10011  
  
def \_\_init\_\_(self, value):  
 if isinstance(value, int) and 0 <= value <= 15:  
 self.value = value  
 elif isinstance(value, str):  
 if 0 <= int(value, 16) <= 15:  
 self.value = int(value, 16)  
  
def \_\_add\_\_(self, other):  
 return GF24(self.value ^ other.value)  
  
def \_\_mul\_\_(self, other):  
 result = 0  
 value\_1 = GF24(self.value)  
 value\_2 = GF24(other.value)  
  
 for i in range(4):  
 if value\_2.value & 1:  
 result ^= value\_1.value  
 if value\_1.value & 0b1000:  
 value\_1.value = (value\_1.value << 1) ^ GF24.primitive  
 else:  
 value\_1.value <<= 1  
 value\_2.value >>= 1  
 return GF24(result)  
  
def \_\_str\_\_(self):  
 return str(self.value)

SimpleAes类初始化

s\_box = [[9, 4, 10, 11],  
 [13, 1, 8, 5],  
 [6, 2, 0, 3],  
 [12, 14, 15, 7]]  
i\_s\_box = [[10, 5, 9, 11],  
 [1, 7, 8, 15],  
 [6, 0, 2, 3],  
 [12, 4, 13, 14]]  
mix\_box = [1, 4, 4, 1]  
i\_mix\_box = [9, 2, 2, 9]  
  
def \_\_init\_\_(self, value):  
 try:  
 if isinstance(value, int) and 0 <= value < 2 \*\* 16:  
 self.value = [value >> 12, value >> 8 & 15, value >> 4 & 15, value & 15]  
 elif isinstance(value, str):  
 value = int(value, 16)  
 if 0 <= value < 2 \*\* 16:  
 self.value = [value >> 12, value >> 8 & 15, value >> 4 & 15, value & 15]  
 elif isinstance(value, list) and len(value) == 4:  
 self.value = [None] \* 4  
 for i in range(len(value)):  
 if isinstance(value[i], int) and 0 <= value[i] <= 15:  
 self.value[i] = value[i]  
 elif isinstance(value[i], str):  
 value[i] = int(value[i], 16)  
 if 0 <= value[i] <= 15:  
 self.value[i] = value[i]  
 except ValueError:  
 self.value = None

算法实现

@staticmethod  
def s\_replace(s, box):  
 str\_s = bin(s)[2:]  
 str\_s = "0" \* (4 - len(str\_s)) + str\_s  
  
 h = int(str\_s[:2], 2)  
 v = int(str\_s[2:], 2)  
 return box[h][v]  
  
@staticmethod  
def half\_byte\_replace(list\_s, box):  
 return [SimpleAes.s\_replace(list\_s[i], box) for i in range(len(list\_s))]  
  
@staticmethod  
def row\_shift(list\_s):  
 list\_s[1], list\_s[3] = list\_s[3], list\_s[1]  
 return list\_s  
  
@staticmethod  
def column\_confuse(list\_s, box):  
 gf\_s = [GF24(list\_s[i]) for i in range(len(list\_s))]  
 gf\_b = [GF24(box[i]) for i in range(len(box))]  
  
 s = [0] \* 4  
 s[0] = (gf\_b[0] \* gf\_s[0] + gf\_b[2] \* gf\_s[1]).value  
 s[1] = (gf\_b[1] \* gf\_s[0] + gf\_b[3] \* gf\_s[1]).value  
 s[2] = (gf\_b[0] \* gf\_s[2] + gf\_b[2] \* gf\_s[3]).value  
 s[3] = (gf\_b[1] \* gf\_s[2] + gf\_b[3] \* gf\_s[3]).value  
 return s  
  
@staticmethod  
def key\_encrypt(list\_s, key):  
 return [s ^ k for s, k in zip(list\_s, key)]  
  
@staticmethod  
def g(w):  
 n\_0 = w >> 4  
 n\_1 = w & 15  
 n\_0 = SimpleAes.s\_replace(n\_0, SimpleAes.s\_box)  
 n\_1 = SimpleAes.s\_replace(n\_1, SimpleAes.s\_box)  
 n = n\_1 << 4 ^ n\_0  
 return n  
  
@staticmethod  
def key\_creat(key):  
 r\_1 = 128  
 r\_2 = 48  
  
 w\_0 = key[0] << 4 ^ key[1]  
 w\_1 = key[2] << 4 ^ key[3]  
 w\_2 = w\_0 ^ r\_1 ^ SimpleAes.g(w\_1)  
 w\_3 = w\_2 ^ w\_1  
 w\_4 = w\_2 ^ r\_2 ^ SimpleAes.g(w\_3)  
 w\_5 = w\_4 ^ w\_3  
  
 key\_1 = [w\_2 >> 4, w\_2 & 15, w\_3 >> 4, w\_3 & 15]  
 key\_2 = [w\_4 >> 4, w\_4 & 15, w\_5 >> 4, w\_5 & 15]  
  
 return key\_1, key\_2  
  
@staticmethod  
def encrypt(message, key):  
 key\_1, key\_2 = SimpleAes.key\_creat(key)  
 m = SimpleAes.key\_encrypt(message, key)  
 m = SimpleAes.half\_byte\_replace(m, SimpleAes.s\_box)  
 m = SimpleAes.row\_shift(m)  
 m = SimpleAes.column\_confuse(m, SimpleAes.mix\_box)  
 m = SimpleAes.key\_encrypt(m, key\_1)  
 m = SimpleAes.half\_byte\_replace(m, SimpleAes.s\_box)  
 m = SimpleAes.row\_shift(m)  
 m = SimpleAes.key\_encrypt(m, key\_2)  
 return m  
  
@staticmethod  
def decrypt(cipher, key):  
 key\_1, key\_2 = SimpleAes.key\_creat(key)  
 c = SimpleAes.key\_encrypt(cipher, key\_2)  
 c = SimpleAes.row\_shift(c)  
 c = SimpleAes.half\_byte\_replace(c, SimpleAes.i\_s\_box)  
 c = SimpleAes.key\_encrypt(c, key\_1)  
 c = SimpleAes.column\_confuse(c, SimpleAes.i\_mix\_box)  
 c = SimpleAes.row\_shift(c)  
 c = SimpleAes.half\_byte\_replace(c, SimpleAes.i\_s\_box)  
 c = SimpleAes.key\_encrypt(c, key)  
 return c

二进制字符串加解密

@staticmethod  
def encrypt\_b(message, key):  
 cipher\_l = []  
  
 if len(message) % 4 == 0:  
 for i in range(0, len(message), 4):  
 message\_l = [message[i], message[i + 1], message[i + 2], message[i + 3]]  
 m = SimpleAes(message\_l)  
 if m.value is None:  
 return "明文内容不正确"  
 k = SimpleAes(key)  
 c = SimpleAes.encrypt(m.value, k.value)  
 cipher\_l.extend(c)  
 else:  
 return "明文格式不正确"  
  
 return cipher\_l  
  
@staticmethod  
def decrypt\_b(cipher, key):  
 message\_l = []  
  
 if len(cipher) % 4 == 0:  
 for i in range(0, len(cipher), 4):  
 cipher\_l = [cipher[i], cipher[i + 1], cipher[i + 2], cipher[i + 3]]  
 c = SimpleAes(cipher\_l)  
 if c.value is None:  
 return "密文内容不正确"  
 k = SimpleAes(key)  
 c = SimpleAes.decrypt(c.value, k.value)  
 message\_l.extend(c)  
 else:  
 return "密文格式不正确"  
  
 return message\_l

ascii,gbk,gb2312,gb18030加解密

@staticmethod  
def encrypt\_c(message, key, codes):  
 try:  
 message\_s = ""  
 message\_l = []  
 message\_input = []  
 message\_output = []  
  
 for m in message:  
 message\_c = m.encode(codes)  
 message\_h = message\_c.hex()  
 if codes == "ascii":  
 message\_s = (2 - len(message\_h)) \* "0" + message\_h  
 elif codes == "gb2312" or codes == "gbk":  
 message\_s = (4 - len(message\_h)) \* "0" + message\_h  
 elif codes == "gb18030":  
 message\_s = (8 - len(message\_h)) \* "0" + message\_h  
 message\_l.append(message\_s)  
  
 if codes == "ascii":  
 if not len(message\_l) % 2 == 0:  
 message\_l.append("00")  
 message\_input = [message\_l[i] + message\_l[i + 1] for i in range(0, len(message\_l), 2)]  
 elif codes == "gb2312" or codes == "gbk":  
 message\_input = message\_l  
 elif codes == "gb18030":  
 message\_input = [message\_l[i // 2][:4] if i % 2 == 0 else message\_l[i // 2][4:]  
 for i in range(len(message\_l \* 2))]  
  
 for m\_i in message\_input:  
 m = SimpleAes(m\_i)  
 k = SimpleAes(key)  
 c = SimpleAes.encrypt(m.value, k.value)  
 message\_output.extend(c)  
  
 message\_l = [hex(message\_output[i])[2:] for i in range(len(message\_output))]  
 message\_s = "".join(message\_l)  
 return message\_s  
  
 except UnicodeEncodeError:  
 return "对应编码不支持该字符"

用户格式化输出

@staticmethod  
def decode\_h(message):  
 message\_s = ""  
 for i in range(0, len(message), 2):  
 message\_s += chr(int(message[i:i + 2], 16))  
 return message\_s  
  
@staticmethod  
def encode\_h(message):  
 message\_h = ""  
 for m in message:  
 message\_s = hex(ord(m))[2:]  
 message\_h += (2 - len(message\_s)) \* "0" + message\_s  
 return message\_h  
  
@staticmethod  
def list\_to\_string(list\_i):  
 list\_h = [hex(i)[2:] for i in list\_i]  
 return "".join(list\_h)

#### window.py

#### window.py中，Entry类调用SimpleAes类，编写了各个按钮的点击事件，并为界面提供5个Entry控件和1个Combobox控件，方便界面操作。

def \_\_init\_\_(self):  
 self.entry\_1 = tk.Entry  
 self.entry\_2 = tk.Entry  
 self.entry\_3 = tk.Entry  
 self.entry\_4 = tk.Entry  
 self.entry\_5 = tk.Entry  
 self.combo = ttk.Combobox  
  
 self.s\_aes = s\_aes.SimpleAes  
 self.key\_1 = 0  
 self.key\_2 = -1  
  
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 == 0:  
 codes = self.combo.get()  
  
 if not is\_hex\_string(entry2\_value):  
 result = "密钥格式不正确"  
 else:  
 if codes == "binary":  
 result = self.s\_aes.encrypt\_b(entry1\_value, entry2\_value)  
 if not (result == "明文内容不正确" or result == "明文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 else:  
 c = self.s\_aes.encrypt\_c(entry1\_value, entry2\_value, codes)  
 result = self.s\_aes.decode\_h(c)  
  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
 elif leaf == 1:  
 codes = self.combo.get()  
  
 entry3\_value = self.entry\_3.get()  
 if not (is\_hex\_string(entry2\_value) and is\_hex\_string(entry3\_value)):  
 result = "密钥格式不正确"  
 else:  
 if codes == "binary":  
 result = self.s\_aes.encrypt\_b(entry1\_value, entry2\_value)  
 if not (result == "明文内容不正确" or result == "明文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.encrypt\_b(result, entry3\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 else:  
 c = self.s\_aes.encrypt\_c(entry1\_value, entry2\_value, codes)  
 result = self.s\_aes.encrypt\_b(c, entry3\_value)  
 if not (result == "明文内容不正确" or result == "明文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decode\_h(result)  
  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
 elif leaf == 2:  
 codes = self.combo.get()  
  
 entry3\_value = self.entry\_3.get()  
 if not (is\_hex\_string(entry2\_value) and is\_hex\_string(entry3\_value)):  
 result = "密钥格式不正确"  
 else:  
 if codes == "binary":  
 result = self.s\_aes.encrypt\_b(entry1\_value, entry2\_value)  
 if not (result == "明文内容不正确" or result == "明文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.encrypt\_b(result, entry3\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.encrypt\_b(result, entry2\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 else:  
 c = self.s\_aes.encrypt\_c(entry1\_value, entry2\_value, codes)  
 result = self.s\_aes.encrypt\_b(c, entry3\_value)  
 if not (result == "明文内容不正确" or result == "明文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.encrypt\_b(result, entry2\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decode\_h(result)  
  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
 elif leaf == 3:  
 codes = self.combo.get()  
  
 entry3\_value = self.entry\_3.get()  
 entry4\_value = self.entry\_4.get()  
 if not (is\_hex\_string(entry2\_value) and is\_hex\_string(entry3\_value) and is\_hex\_string(entry4\_value)):  
 result = "密钥格式不正确"  
 else:  
 if codes == "binary":  
 result = self.s\_aes.encrypt\_b(entry1\_value, entry2\_value)  
 if not (result == "明文内容不正确" or result == "明文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.encrypt\_b(result, entry3\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.encrypt\_b(result, entry4\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 else:  
 c = self.s\_aes.encrypt\_c(entry1\_value, entry2\_value, codes)  
 result = self.s\_aes.encrypt\_b(c, entry3\_value)  
 if not (result == "明文内容不正确" or result == "明文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.encrypt\_b(result, entry4\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decode\_h(result)  
  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
 elif leaf == 4:  
 codes = self.combo.get()  
  
 if not is\_hex\_string(entry2\_value):  
 result = "密钥格式不正确"  
 else:  
 if codes == "binary":  
 result = self.s\_aes.decrypt\_b(entry1\_value, entry2\_value)  
 if not (result == "密文内容不正确" or result == "密文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 else:  
 c = self.s\_aes.encode\_h(entry1\_value)  
 result = self.s\_aes.decrypt\_c(c, entry2\_value, codes)  
  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
 elif leaf == 5:  
 codes = self.combo.get()  
  
 entry3\_value = self.entry\_3.get()  
 if not (is\_hex\_string(entry2\_value) and is\_hex\_string(entry3\_value)):  
 result = "密钥格式不正确"  
 else:  
 if codes == "binary":  
 result = self.s\_aes.decrypt\_b(entry1\_value, entry3\_value)  
 if not (result == "密文内容不正确" or result == "密文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decrypt\_b(result, entry2\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 else:  
 c = self.s\_aes.encode\_h(entry1\_value)  
 result = self.s\_aes.decrypt\_b(c, entry3\_value)  
 if not (result == "密文内容不正确" or result == "密文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decrypt\_c(result, entry2\_value, codes)  
  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
 elif leaf == 6:  
 codes = self.combo.get()  
  
 entry3\_value = self.entry\_3.get()  
 if not (is\_hex\_string(entry2\_value) and is\_hex\_string(entry3\_value)):  
 result = "密钥格式不正确"  
 else:  
 if codes == "binary":  
 result = self.s\_aes.decrypt\_b(entry1\_value, entry2\_value)  
 if not (result == "密文内容不正确" or result == "密文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decrypt\_b(result, entry3\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decrypt\_b(result, entry2\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 else:  
 c = self.s\_aes.encode\_h(entry1\_value)  
 result = self.s\_aes.decrypt\_b(c, entry2\_value)  
 if not (result == "密文内容不正确" or result == "密文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decrypt\_b(result, entry3\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decrypt\_c(result, entry2\_value, codes)  
  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
 elif leaf == 7:  
 codes = self.combo.get()  
  
 entry3\_value = self.entry\_3.get()  
 entry4\_value = self.entry\_4.get()  
 if not (is\_hex\_string(entry2\_value) and is\_hex\_string(entry3\_value) and is\_hex\_string(entry4\_value)):  
 result = "密钥格式不正确"  
 else:  
 if codes == "binary":  
 result = self.s\_aes.decrypt\_b(entry1\_value, entry4\_value)  
 if not (result == "密文内容不正确" or result == "密文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decrypt\_b(result, entry3\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decrypt\_b(result, entry2\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 else:  
 c = self.s\_aes.encode\_h(entry1\_value)  
 result = self.s\_aes.decrypt\_b(c, entry4\_value)  
 if not (result == "密文内容不正确" or result == "密文格式不正确"):  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decrypt\_b(result, entry3\_value)  
 result = self.s\_aes.list\_to\_string(result)  
 result = self.s\_aes.decrypt\_c(result, entry2\_value, codes)  
  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
 elif leaf == 8:  
 result = self.middle\_attack()  
  
 if len(result) == 2:  
 self.entry\_4.config(state="normal")  
 self.entry\_4.delete(0, tk.END)  
 self.entry\_4.insert(0, result[0])  
 self.entry\_4.config(state="readonly")  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result[1])  
 self.entry\_5.config(state="readonly")  
 else:  
 self.entry\_4.config(state="normal")  
 self.entry\_4.delete(0, tk.END)  
 self.entry\_4.insert(0, result)  
 self.entry\_4.config(state="readonly")  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
 elif leaf == 9:  
 entry3\_value = self.entry\_3.get()  
  
 if not is\_hex\_string(entry2\_value):  
 result = "密钥格式不正确"  
 elif not is\_hex\_string(entry3\_value):  
 result = "初始向量格式不正确"  
 else:  
 result = ""  
 if len(entry1\_value) % 4 == 0:  
 init\_v = entry3\_value  
 for i in range(0, len(entry1\_value), 4):  
 m = entry1\_value[i:i+4]  
 m = hex(int(m, 16) ^ int(init\_v, 16))[2:]  
 m = (4 - len(m)) \* "0" + m  
 c = self.s\_aes.encrypt\_b(m, entry2\_value)  
 if not (c == "明文内容不正确" or c == "明文格式不正确"):  
 init\_v = self.s\_aes.list\_to\_string(c)  
 result += init\_v  
 else:  
 break  
  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
 elif leaf == 10:  
 entry3\_value = self.entry\_3.get()  
  
 if not is\_hex\_string(entry2\_value):  
 result = "密钥格式不正确"  
 elif not is\_hex\_string(entry3\_value):  
 result = "初始向量格式不正确"  
 else:  
 result = ""  
 if len(entry1\_value) % 4 == 0:  
 for i in range(len(entry1\_value)-4, -1, -4):  
 c = entry1\_value[i:i + 4]  
 if i > 3:  
 init\_v = entry1\_value[i - 4:i]  
 else:  
 init\_v = entry3\_value  
 m = self.s\_aes.decrypt\_b(c, entry2\_value)  
 if not (m == "密文内容不正确" or m == "密文格式不正确"):  
 m = self.s\_aes.list\_to\_string(m)  
 m = hex(int(m, 16) ^ int(init\_v, 16))[2:]  
 m = (4 - len(m)) \* "0" + m  
 result = m + result  
 else:  
 break  
  
 self.entry\_5.config(state="normal")  
 self.entry\_5.delete(0, tk.END)  
 self.entry\_5.insert(0, result)  
 self.entry\_5.config(state="readonly")  
  
def middle\_attack(self):  
 entry1\_value = self.entry\_1.get()  
 entry2\_value = self.entry\_2.get()  
  
 for i in range(self.key\_1, 2 \*\* 16):  
 key\_1 = hex(i)[2:]  
 key\_1 = (4 - len(key\_1)) \* "0" + key\_1  
 for j in range(2 \*\* 16):  
 key\_2 = hex(j)[2:]  
 key\_2 = (4 - len(key\_2)) \* "0" + key\_2  
  
 if i == self.key\_1 and j <= self.key\_2:  
 continue  
  
 result\_1 = self.s\_aes.encrypt\_b(entry1\_value, key\_1)  
 if not (result\_1 == "明文内容不正确" or result\_1 == "明文格式不正确"):  
 result\_1 = self.s\_aes.list\_to\_string(result\_1)  
 else:  
 return result\_1  
  
 result\_2 = self.s\_aes.decrypt\_b(entry2\_value, key\_2)  
 if not (result\_2 == "密文内容不正确" or result\_2 == "密文格式不正确"):  
 result\_2 = self.s\_aes.list\_to\_string(result\_2)  
 else:  
 return result\_2  
  
 if result\_1 == result\_2:  
 self.key\_1 = i  
 self.key\_2 = j  
 return key\_1, key\_2  
  
 return "无后续密钥对"

#### 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。

@staticmethod  
def main():  
 # 创建主窗口  
 root = tk.Tk()  
 root.title("ttk.Notebook 示例")  
 root.geometry("500x400+510+230")  
  
 # 创建分页标签控件  
 notebook = ttk.Notebook(root)  
  
 # 创建第一个选项卡  
 tab\_1 = ttk.Notebook(notebook)  
 notebook.add(tab\_1, text="加密")  
  
 tab\_1\_1 = ttk.Frame(notebook)  
 tab\_1.add(tab\_1\_1, text="普通加密")  
 tab\_1\_2 = ttk.Frame(notebook)  
 tab\_1.add(tab\_1\_2, text="双重加密")  
 tab\_1\_3 = ttk.Frame(notebook)  
 tab\_1.add(tab\_1\_3, text="三重加密(32bits)")  
 tab\_1\_4 = ttk.Frame(notebook)  
 tab\_1.add(tab\_1\_4, text="三重加密(48bits)")  
  
 # 创建第二个选项卡  
 tab\_2 = ttk.Notebook(notebook)  
 notebook.add(tab\_2, text="解密")  
  
 tab\_2\_1 = ttk.Frame(notebook)  
 tab\_2.add(tab\_2\_1, text="普通解密")  
 tab\_2\_2 = ttk.Frame(notebook)  
 tab\_2.add(tab\_2\_2, text="双重解密")  
 tab\_2\_3 = ttk.Frame(notebook)  
 tab\_2.add(tab\_2\_3, text="三重解密(32bits)")  
 tab\_2\_4 = ttk.Frame(notebook)  
 tab\_2.add(tab\_2\_4, text="三重解密(48bits)")  
  
 tab\_3 = ttk.Frame(notebook)  
 notebook.add(tab\_3, text="中间相遇攻击")  
  
 tab\_4 = ttk.Notebook(notebook)  
 notebook.add(tab\_4, text="密码分组链(CBC)模式")  
 tab\_4\_1 = ttk.Frame(notebook)  
 tab\_4.add(tab\_4\_1, text="加密")  
 tab\_4\_2 = ttk.Frame(notebook)  
 tab\_4.add(tab\_4\_2, text="解密")  
  
 # 将 ttk.Notebook 放置在主窗口中  
 notebook.pack()  
  
 Window.page(tab\_1\_1, 0)  
 Window.page(tab\_1\_2, 1)  
 Window.page(tab\_1\_3, 2)  
 Window.page(tab\_1\_4, 3)  
 Window.page(tab\_2\_1, 4)  
 Window.page(tab\_2\_2, 5)  
 Window.page(tab\_2\_3, 6)  
 Window.page(tab\_2\_4, 7)  
 Window.page(tab\_3, 8)  
 Window.page(tab\_4\_1, 9)  
 Window.page(tab\_4\_2, 10)  
  
 # 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)  
 entry.entry\_1 = ttk.Entry(frame)  
 label\_2 = tk.Label(frame)  
 entry.entry\_2 = ttk.Entry(frame)  
 label\_3 = tk.Label(frame)  
 entry.entry\_3 = ttk.Entry(frame)  
 label\_4 = tk.Label(frame)  
 entry.entry\_4 = ttk.Entry(frame)  
 label\_5 = tk.Label(frame)  
 entry.entry\_5 = ttk.Entry(frame, state="readonly")  
  
 label\_6 = tk.Label(frame, text="编码:")  
 entry.combo = ttk.Combobox(frame)  
 entry.combo.configure(width=10)  
 entry.combo["value"] = ("binary", "ascii", "gbk", "gb2312", "gb18030")  
 entry.combo.set("binary")  
 entry.combo["state"] = "readonly"  
  
 # 创建按钮  
 button = ttk.Button(root)  
 button.grid(row=1, column=0, padx=10, pady=10)  
 button.config(command=lambda: entry.on\_button\_click(leaf))  
  
 label\_1.grid(row=0, column=0, padx=20, pady=10)  
 entry.entry\_1.grid(row=0, column=1, padx=20, pady=10)  
 label\_2.grid(row=1, column=0, padx=20, pady=10)  
 entry.entry\_2.grid(row=1, column=1, padx=20, pady=10)  
  
 if leaf == 0 or leaf == 4:  
 if leaf == 0:  
 label\_1.config(text="明文")  
 label\_5.config(text="密文")  
 button.config(text="加密")  
 elif leaf == 4:  
 label\_1.config(text="密文")  
 label\_5.config(text="明文")  
 button.config(text="解密")  
  
 label\_2.config(text="密钥")  
 label\_5.grid(row=2, column=0, padx=20, pady=10)  
 entry.entry\_5.grid(row=2, column=1, padx=20, pady=10)  
 label\_6.grid(row=3, column=0, padx=20, pady=10)  
 entry.combo.grid(row=3, column=1, padx=20, pady=10)  
 elif leaf == 1 or leaf == 2 or leaf == 5 or leaf == 6:  
 if leaf == 1 or leaf == 2:  
 label\_1.config(text="明文")  
 label\_5.config(text="密文")  
 button.config(text="加密")  
 elif leaf == 5 or leaf == 6:  
 label\_1.config(text="密文")  
 label\_5.config(text="明文")  
 button.config(text="解密")  
  
 label\_2.config(text="密钥1")  
 label\_3.config(text="密钥2")  
 label\_3.grid(row=2, column=0, padx=20, pady=10)  
 entry.entry\_3.grid(row=2, column=1, padx=20, pady=10)  
 label\_5.grid(row=3, column=0, padx=20, pady=10)  
 entry.entry\_5.grid(row=3, column=1, padx=20, pady=10)  
 label\_6.grid(row=4, column=0, padx=20, pady=10)  
 entry.combo.grid(row=4, column=1, padx=20, pady=10)  
 elif leaf == 3 or leaf == 7:  
 if leaf == 3:  
 label\_1.config(text="明文")  
 label\_5.config(text="密文")  
 button.config(text="加密")  
 elif leaf == 7:  
 label\_1.config(text="密文")  
 label\_5.config(text="明文")  
 button.config(text="解密")  
  
 label\_2.config(text="密钥1")  
 label\_3.config(text="密钥2")  
 label\_3.grid(row=2, column=0, padx=20, pady=10)  
 entry.entry\_3.grid(row=2, column=1, padx=20, pady=10)  
 label\_4.config(text="密钥3")  
 label\_4.grid(row=3, column=0, padx=20, pady=10)  
 entry.entry\_4.grid(row=3, column=1, padx=20, pady=10)  
 label\_5.grid(row=4, column=0, padx=20, pady=10)  
 entry.entry\_5.grid(row=4, column=1, padx=20, pady=10)  
 label\_6.grid(row=5, column=0, padx=20, pady=10)  
 entry.combo.grid(row=5, column=1, padx=20, pady=10)  
 elif leaf == 8:  
 label\_1.config(text="明文")  
 label\_2.config(text="密文")  
 label\_4.config(text="密钥1")  
 label\_4.grid(row=2, column=0, padx=20, pady=10)  
 entry.entry\_4.config(state="readonly")  
 entry.entry\_4.grid(row=2, column=1, padx=20, pady=10)  
 label\_5.config(text="密钥2")  
 label\_5.grid(row=3, column=0, padx=20, pady=10)  
 entry.entry\_5.grid(row=3, column=1, padx=20, pady=10)  
 button.config(text="破解")  
  
 label\_6.destroy()  
 entry.combo.destroy()  
 elif leaf == 9 or leaf == 10:  
 if leaf == 9:  
 label\_1.config(text="明文")  
 label\_5.config(text="密文")  
 button.config(text="加密")  
 elif leaf == 10:  
 label\_1.config(text="密文")  
 label\_5.config(text="明文")  
 button.config(text="解密")  
  
 label\_2.config(text="密钥")  
 label\_3.config(text="初始向量")  
 label\_3.grid(row=2, column=0, padx=20, pady=10)  
 entry.entry\_3.grid(row=2, column=1, padx=20, pady=10)  
 label\_5.grid(row=3, column=0, padx=20, pady=10)  
 entry.entry\_5.grid(row=3, column=1, padx=20, pady=10)  
  
 label\_6.destroy()  
 entry.combo.destroy()