第5章 信息表示与加解密

数据是信息的载体，同样的信息可以使用不同的数据表示格式加以编码表示。加密与解密的本质就是对信息的数据表示格式的一种编码。信息的加密与解密是保障信息安全的重要手段，其相关的算法与技术也是计算机科学领域的重要研究内容之一。

我们把原始的、以可读方式编码的数据称为“明文”。信息的加密的过程就是对原来为明文的数据，按某种变换方法进行重新编码，使其成为不可读的数据，从而达到隐藏信息的目的。被加密的数据称为“密文”。未授权的用户即使获得了密文，由于他们无法解读该数据，也无法从数据中获得有用的信息。信息的解密则是将信息从不可读的格式重新转换为可读的格式。

加密、解密过程如图 5-1 所示。

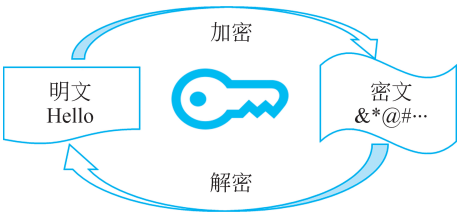

图 5-1　信息的加密、解密

由此可见，将明文字符串转换为密文字符串的变换方法称为加密算法。与之相对应的，将密文字符串转换为明文字符串的变换方法称为解密算法。加密和解密是相互依存、密不可分的。其中，加密、解密的关键依赖于“密钥”，通常密钥是由数字、字母或特殊符号组成的字符串。

5.1　加密原理

经典加密算法很多，按照国际上通行的惯例，可以根据双方收发的密钥是否相同将加密算法划分为两类：常规加密算法和公钥加密算法。

1. 常规加密算法

常规加密算法，也称私钥加密算法或对称加密算法，其特征是接收方和发送方使用相同

的密钥，即加密密钥和解密密钥是相同的或等价的。常规加密算法主要有 DES 算法、3DES 算法、TDEA 算法、IDEA 算法以及以代换密码和转轮密码为代表的古典密码等。

2. 公钥加密算法

公钥加密算法也称非对称加密算法，其特征是接收方和发送方使用的密钥互不相同，而且几乎不可能从加密密钥推导出解密密钥。公钥加密算法主要有 RSA 算法、背包算法、Elgamal、ECC(椭圆曲线加密算法)等。

本章以古典密码——凯撒密码(Caesar's code)为例，介绍一种信息的对称加密体制。

5.1.1 移位密码原理

凯撒加密是一种古老的对称加密体制，在古罗马的时候就已经很流行，是一种简单实用的移位对称加密方式。凯撒加密的对象是用古罗马文字表示的信息，古罗马的文字就是我们英语中熟知的 26 个拉丁字母。

凯撒加密的基本思想：把明文中的每个字母用它在字母表中位置向右(或向左)按照一个固定数目(n)进行偏移后得到的字母替换。解密就是加密的逆过程。

位数 n 为移位加密和解密的密钥，$n \geqslant 1$，一般默认加密时向右移位 n 位，密钥为 n，向左移位 n 位，则密钥为 $-n$。

例如，当密钥为 2 时，将明文每一个字母用其字母表中向右移位 2 位的字母来替换，即 A→C，B→D，C→E，以此类推，最后 Y→A，Z→B。

假设明文为“COMPUTER”，当密钥为 2 时，经凯撒加密运算后，得到的密文为“EQORWVGT”；当密钥为 3 时，得到的密文为“FRPSXWHU”。

显然，经过移位运算后得到的这两个字符串看上去毫无规则，也没有明确的含义，窃密者就算是获得了这两个字符串，如果不知道其中的奥秘也无法获得正确的信息。此外，我们可以发现，就算是对同一段明文采用相同的加密算法，当采用不同的密钥时，也会产生不同的密文。由此可见，字符的变换规则和密钥本身的安全性在保证加密算法的有效性上至关重要，一旦变换规则和密钥被敌方掌握，就毫无秘密可言。

解密的过程则是加密的逆运算，当我们已知加密算法为凯撒加密，而密钥为 3 时，则面对密文“FRPSXWHU”做移位运算，将每个对应的字母用字母表位置向前的第 3 个字母替换，就会得到解密后的明文“COMPUTER”。

凯撒加密其实并不仅仅适用于对罗马字母的加密，在信息数字化的今天，该技术可以扩展到对所有信息的加密上。我们知道，在计算机中所有的数据在存储和运算时都要使用二进制数表示。对字符的编码有 ASCII、中国国家标准汉字编码和 Unicode 等编码标准，这些编码标准统一规定了常用的字母或符号用哪些二进制数来表示，是人们共同使用的标准编码规则。

下面以基于 ASCII 码的信息加密为例，介绍凯撒加密算法的 Python 实现方法。

5.1.2 ASCII 码

ASCII 英文全称是 American Standard Code for Information Interchange，即美国标准信息交换代码，是由美国国家标准学会（American National Standard Institute，ANSI）制定的一套字符编码方案。它已被国际标准化组织（International Organization for Standardization，ISO）定为国际标准，称为 ISO 646 标准，适用于所有拉丁文字字母。

一个 ASCII 码一般由 8 位二进制组成，实际只使用低 7 位，最高位恒设为 0，如图 5-2 所示。

L	H							
	0000	0001	0010	0011	0100	0101	0110	0111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENG	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	↑	n	~
1111	SI	US	/	?	O	←	o	DEL

注：H表示高4位，L表示低4位。

图 5-2 ASCII 编码表

说明：

（1）列标题中显示的是字符的 8 位二进制编码中的低 4 位（L），行标题给出编码的高 4 位（最高位恒为 0），一个字符的编码由所在行列的低 4 位编码和高 4 位编码组合形成。例如，数字字符'0'的 8 位二进制编码为 00110000，对应十进制整数为 48。由此可见，ASCII 编码表中数字的 ASCII 码与它表示的数值是完全不同的两个概念。大写字母'A'的 8 位二进制编码为 01000001，对应十进制整数为 65。小写字母'a'的 8 位二进制编码为 01100001，对应十进制整数为 97。

（2）ASCII 标准使用 8 位二进制为每个字符编码，理论上能够表示 $2^8=256$ 个字符，但实际上，由于最高位恒为 0，因此其表示的字符个数为 $2^7=128$ 个，剩余的一半编码空置留待他用。编号为十进制数的 0～31 及 127（共 33 个）是控制字符或通信专用字符，如控制符 LF（换行）、CR（回车）、DEL（删除）等；32～126（共 95 个）是字符，其中 48～57 为 0～9 这 10 个阿拉伯数字，65～90 为 26 个大写英文字母，97～122 为 26 个小写英文字母。

（3）字符依据 0～9＜A～Z＜a～z 的规则编码，即数字字符'0'的编码比数字字符'9'的编码小，并按 0～9 的顺序递增，字母'A'的编码比字母'Z'的编码小，并按 A～Z 的顺序递增，同一个英文字母，其大写形式的编码比小写形式的编码小 32。

现在我们已经掌握了字符的 ASCII 编码方式，在这种编码模式下，将一个字符移位替

换为另一个字符，只需要对该字符的 ASCII 码进行相应的加减运算即可。

5.1.3 字符与编码的转换函数

在 Python 中，我们不能对字符本身进行直接的算术运算。例如，将字符'a'右移 2 位替换为字符'c'，如果直接写成“'a'+2”，会产生语法错误，而是应该先获取字符'a'的 ASCII 编码，即 97，然后执行 97+2，得到 99，最后将 99 转换为对应的 ASCII 字符，即字符'c'。显然，在这个过程中涉及字符与编码之间的两个转换运算，即将字符转换为编码的运算和将编码转换为字符的运算。

Python 语言的内置函数 chr()和 ord()就提供了上述两个转换的计算，下面分别介绍这两个函数及其使用方法。

1. chr()函数

格式：

```
chr(<数值表达式>)
```

说明：该函数的参数是一个整数型数据(int)，其数值的取值范围为 0～255，用于将一个整数转换为对应 ASCII 码的字符。返回值类型为字符串类型(string)。

例如，在 Shell 中输入 chr(97)，结果显示：'a'。

```
>>>chr(97)
'a'
```

2. ord()函数

格式：

```
ord('字符串')
```

说明：该函数的参数是一个字符型数据(string)，用于将字符转换成它对应的 ASCII 码的十进制整数值。返回值类型为整数型数据类型(int)。

例如，在 Shell 中输入 ord('a')，结果显示为 97。

```
>>> ord('a')
97
```

注意： ord()函数的参数必须为字符型数据，若采用 ord()函数将 ASCII 码中的阿拉伯数字字符转换为对应的整数值，数字符必须带单引号或者双引号，例如数字符'2'的 ASCII 码是 00110010，对应的十进制整数是 50，那么函数 ord('2')的计算结果为 50。

代码如下：

```
>>> ord('2')
50
```

5.2 字符串加解密

在进行字符串加解密学习之前，首先介绍单个字符的加解密过程。

5.2.1 单个字符加解密

例 5-1 从键盘输入一个字母，实现移位为 2 的加密，并输出加密结果。

创建文件“encoding.py”如下：

```
pla=input("请输入要加密的字母,以回车结束:")
n=2
cip=chr(ord(pla)+n)
print(cip)
```

说明：

(1) 使用 input()函数，获取要加密的字母，并保存在一个字符串变量 pla 中，其中 pla 是 plaintext(明文)的缩写。

(2) 表达式“cip=chr(ord(pla)+n)”中涉及两个函数的嵌套调用。

① 调用函数 ord(pla)，其作用是将变量 pla 中的字符转换为相应的 ASCII 码的十进制整数形式。

② 对该整数执行加 2 运算，得到移位运算后字符的整数值。

③ 将第二步中的运算结果作为函数参数放到 chr()函数中，其作用是得到该整数对应的字符。

④ 将得到的字符存放在变量 cip 中，其中 cip 是 ciphertext(密文)的缩写。

(3) 使用 print()函数，将加密结果输出。

运行程序代码，输入任意字母，运行结果如图 5-3 所示。

从运行结果可以看出，当输入的字母为 a～x 或 A～X 的任意字母时，加密结果仍为 26 个字母，但如果输入的字母是 y(Y)或者 z(Z)时，输出结果就不再属于 26 个英文字母的范围。这是因为对 ASCII 码直接加上移位密钥 n 的运算方式有可能会导致计算结果超出 26 个英文字母的表示范围。

如何使加密结果限定在大写字母或小写字母的范围内呢？可以利用求余运算来完成。

修改“encoding.py”，代码如下。

```
pla=input("请输入要加密的字母,以回车结束: ")
n=2
cip=chr((ord(pla)-ord('a')+n)%26+ord('a'))
print(cip)
```

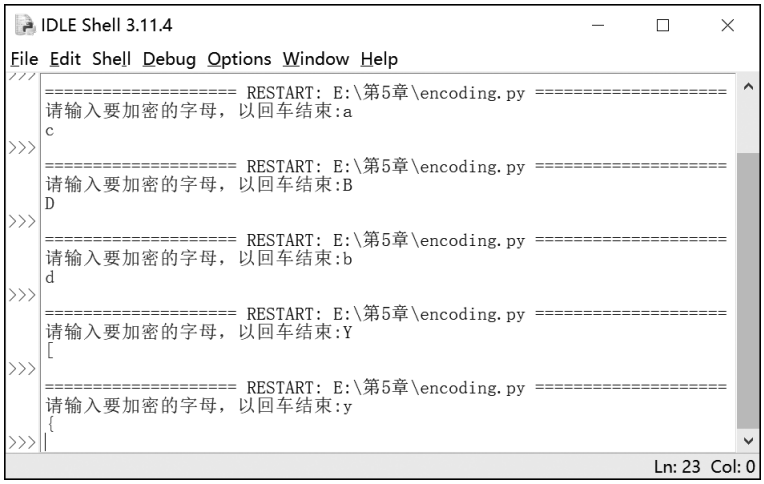

图 5-3　单个字符加密

程序的运行结果如图 5-4 所示。

图 5-4　单个字符循环加密文件运行结果

上述的加密过程中，密钥是由题目给定，但加密程序应允许用户设置密钥 n，用一个整型变量 n 表示和存放用户从键盘输入的密钥。

```
n=int(input("请输入密钥: "))
pla=input("请输入要加密的字母,以回车结束: ")
cip=chr((ord(pla)-ord('a')+n)%26+ord('a'))
print(cip)
```

说明：

(1) 函数 int()用于将 input()从键盘获取字符串类型数据转换为整型。

(2) 若需要将大写字母的加密结果同样限定在大写字母中，只需将第三行代码中'a'改为'A'。

运行上述代码，结果如图 5-5 所示。

图 5-5　单个字符指定密钥循环解密

对于移位加密来说，解密的过程就是按照相反的方向移动 n 位，请读者自行思考并实现。

5.2.2 字符串加解密

例 5-2 从键盘输入字符串"computer"，实现密钥为 n 的加密，并输出加密结果。

分析：字符串是由一个个字符组成的，前面学习了如何加密单个字符，那么字符串的加密需要解决以下几个问题。

1. 获取键盘输入的字符串（英文字母）

使用 input()函数，获取要加密的文字内容，并保存在一个字符串变量中。

代码如下：

```
pla=input("请输入要加密的文字,以回车结束:")
```

2. 拆解字符串

利用列表结构(list)将变量 pla 中的字符串分割为单个字符，并按序保存在列表中。

代码如下：

```
>>> pla="abc"
>>> pla
'abc'
>>> list=list(pla)
>>> list
['a', 'b', 'c']
```

由此，list 中的每个成员就是 pla 字符串中的单个字符。

3. 统一大小写字母

对于大写字母和小写字母的加密计算在表达式上有细微的不同，为了计算方便，可以将需要处理的字符串进行初步处理，将它们统一转换为小写字母再进行移位加密，代码如下：

```
letter_list=list(pla.lower())
```

说明：pla.lower()的作用是将变量 pla 中的字符串全部转换为小写字母。执行该语句，pla 中的字符串转换为小写字母，并存储在明文字符列表 letter_list 中。

4. 定义对单个字符进行加密运算的函数

由于需要对字符列表中的每个字符分别执行加密计算，也就是该加密计算需要反复多次使用，因此一种较为便利的方法就是将该功能封装成函数，从而方便在需要的时候多次调用。定义加密函数的代码如下。

```
def move_letter(letter,n):
    if letter=='':
        return('')
    else:
        return chr((ord(letter)-ord('a')+n)%26+ord('a'))
```

说明：

(1) 自定义函数 move_letter(letter,n)有两个参数，其中 letter 参数表示要进行加密的单个字母，n 表示移位的个数，即密钥。

(2) 函数的返回值是经过加密后的字符。

(3) 第 2 行和第 3 行代码是对字符串中可能存在的空格字符的情况进行处理，这里假设针对空格字符不做移位处理，原样输出空格。

5. 定义一个列表，用于存储经过加密后的字符

代码如下。

```
e_list=[]
```

说明：

(1) 由于尚未开始加密运算，所以一开始该列表为空。

(2) 可以利用 append()函数将加密后的字符加入 e_list 中，代码如下。

```
e_list.append()
```

6. 循环遍历并加密明文字符列表中的每个字符

使用循环结构，遍历 letter_list 中的每个字母，并将字母作为参数放入 move_letter()函数中进行加密运算，并将加密后的字符逐个添加到 e_list 列表中保存起来，代码如下。

```
for x in letter_list:
    c=move_letter(x,n)
    e_list.append(c)
```

说明：其中第 2 行和第 3 行的代码都属于循环体要执行的步骤，因此要注意缩进。

7. 将密文字符列表中的字符组合成字符串

代码如下。

```
e_letter=''.join(e_list)
```

说明：

(1) ''.join 前的是一对单引号，不是双引号，并且这一对单引号中不包含任何符号。join()函数的作用是将字符列表中的字符按序组合成字符串，且字符之间无分隔符。

(2) 合并后的结果保存在 e_letter 字符串变量中。

至此,例 5-2 的问题都已解决,能够实现字符串加密并输出。

完整代码如下。

```
def move_letter(letter,n):
    if letter=='':
        return('')
    else:
        return chr((ord(letter)-ord('a')+n)%26+ord('a'))
n=int(input('请输入密钥:'))
pla=input('请输入要加密的字符串,以回车结束:')
letter_list=list(pla.lower())
e_list=[]
for x in letter_list:
    c=move_letter(x,n)
    e_list.append(c)
e_letter=''.join(e_list)
print("加密后字符串:",e_letter)
```

运行结果如图 5-6 所示。

图 5-6 字符串加密

对于移位加密来说,解密的过程就是按照相反的方向移动 n 位,所以要调用 move_letter()函数,只要修改参数即可。

代码如下。

```
for x in e_list:
    c=move_letter(x,-n)
    d_list.append(c)
d_letter=''.join(d_list)
print("解密后字符串:", d_letter)
```

运行结果如图 5-7 所示。

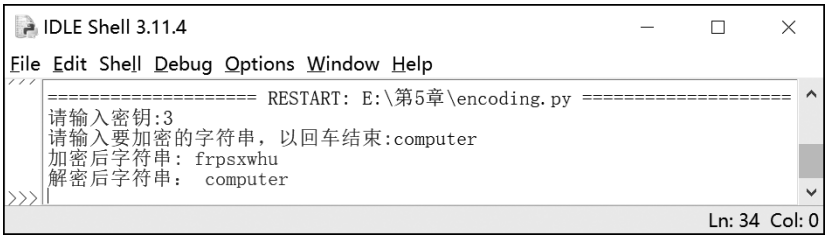

图 5-7 字符串解密

5.3 文件加解密

至此，我们已经完成了对单个字符串的加密与解密。我们知道，单个字符串能够表达的信息是很有限的，一段完整的信息通常以文件的形式进行保存和传输。如何对一个文件进行加密和解密呢？我们将在本节中完成这个任务。

5.3.1 从文件中读取数据

文件通常是存储在磁盘等外部存储介质中，如果要处理文件中的数据，首先需要将文件数据读取到内存中，然后才能进行计算。Python 提供了两种文件读取方式：一次性读取文件全部内容；每次读取文件中的一行数据。

首先需要创建一个包含多行字符的文本文件。如图 5-8 所示，创建了一个记事本文件，名为“file.txt”。

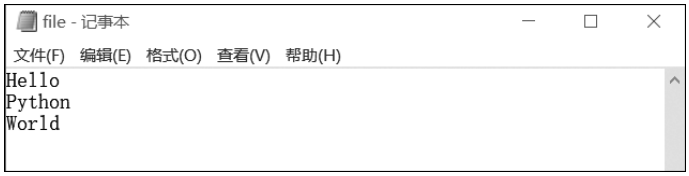

图 5-8 创文本“file.txt”文件

下面分别使用这两种方式读取该文件。

1. 读取整个文件

下面的程序文件“file_reader.py”可以实现打开并读取 file.txt 文件，再将其内容显示在屏幕上，代码如下。

```
import os
f=open("file.txt")
contents=f.read()
print(contents)
f.close()
```

说明：

(1) os 模块：使用 import 指令，导入 os 模块。该模块中包含了对文件进行的操作(如打开文件、关闭文件、读文件和写文件等)的一些函数，方便人们使用。

(2) open()函数：不管以任何方式使用文件(读或写)，都要先打开文件。open()函数需要一个参数，就是要打开的文件的名称。其中，文件的名称需要区分两种情况：如果文件与处理文件的 Python 程序放在同一个目录中，则只需要填写文件的名称，如“file.txt”；如果要打开的文件不在程序文件所属目录下，那么文件的名称就应该包含文件在计算机中的完整路径名称，如“D:\qycache\file.txt”。