CSC110 Lecture 18: Introduction to Cryptography

Print this handout

Exercise 1: The One-Time Pad Cryptosystem

1. Suppose we want to encrypt the plaintext `'david'` using the one-time pad cryptosystem and the secret key `'mario'`. Fill in the table below to come up with the encrypted ciphertext. You may find the following useful:

```
>>> [ord(char) for char in 'david']
[100, 97, 118, 105, 100]
>>> [ord(char) for char in 'mario']
[109, 97, 114, 105, 111]
```

If you are trying to do this first without using the Python console, you can use the ASCII code chart found at the bottom of this worksheet.

message char	ord of message char	key char	ord of key char	ord of ciphertext char	ciphertext char
`'d'`	100	`'m'`	109	81	`'Q'`
`'a'`		`'a'`			
`'v'`		`'r'`			
`'i'`		`'i'`			
`'d'`		`'o'`			

2. Next, implement the one-time pad cryptosystem by completing the following two functions `encrypt_otp` and `decrypt_otp`. Some tips/hints:

   - The implementation is quite similar to the Caesar cipher from Section 8.1.
   - Remember that you can use `ord` and `chr` to convert back and forth between characters and numbers.
   - `%` has higher preecedence than `+`/`-`, so you'll need to do `(a + b) % n` instead of `a + b % n`.

```
def encrypt_otp(k: str, plaintext: str) -> str:
    """Return the encrypted message of plaintext using the key k with the
    one-time pad cryptosystem.

    Preconditions:
        - len(k) >= len(plaintext)
        - all({ord(c) < 128 for c in plaintext})
        - all({ord(c) < 128 for c in k})

    >>> encrypt_otp('david', 'HELLO')
    ',&B53'
    """

def decrypt_otp(k: str, ciphertext: str) -> str:
    """Return the decrypted message of ciphertext using the key k with the
    one-time pad cryptosystem.

    Preconditions:
        - len(k) >= len(ciphertext)
        - all({ord(c) < 128 for c in ciphertext})
        - all({ord(c) < 128 for c in k})

    >>> decrypt_otp('david', ',&B53')
    'HELLO'
    """
```

3. Check if you can get the original plaintext message back when using your `encrypt_otp` and `decrypt_otp` functions:

```
>>> plaintext = 'david'
>>> key = 'mario'
>>> ciphertext = encrypt_otp(key, plaintext)
>>> decrypt_otp(key, ciphertext)
'david'
```

Exercise 2: The Diffie-Hellman key exchange algorithm

We discussed in lecture how the Diffie-Hellman key exchange is computationally secure. But it's important to remember that computational security is not the same as theoretical security. Let's implement a *brute-force* algorithm for an eavesdropper to take the p, g, $g^a \% p$, and $g^b \% p$ values that Alice and Bob communicate from the algorithm, and uses this to determine the shared secret key.

Your algorithm should try to recover one of the exponents a or b simply by try all possible values: $\{1, 2, \ldots, p - 1\}$. This is computationally inefficient in practice when p is chosen to be extremely large. But how quickly can we do it with small prime numbers (e.g., 23)?

```
def break_diffie_hellman(p: int, g: int, g_a: int, g_b: int) -> int:
    """Return the shared Diffie-Hellman secret key obtained from the eavesdropped inform

    Remember that the secret key is (g ** (a * b)) % p, where a and b are the
    secret exponents chosen by Alice and Bob.
    You'll need to find at least one of a and b to compute the secret key.

    Preconditions:
        - p, g, g_a, and g_b are the values exhanged between Alice and Bob
          in the Diffie-Hellman algorithm

    >>> p = 23
    >>> g = 2
    >>> g_a = 9  # g ** 5 % p
    >>> g_b = 8  # g ** 14 % p
    >>> break_diffie_hellman(p, g, g_a, g_b)  # g ** (5 * 14) % p
    16
    """
```

ASCII chart

```
 0 NUL    16 DLE    32      48 0    64 @    80 P    96 `   112 p
 1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
 2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
 3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
 4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
 5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
 6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
 7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
 8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
 9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
15 SI     31 US     47 /    63 ?    79 O    95 _   111 o   127 DEL
```