CSC110 Lecture 18: Introduction to Cryptography

Hisbaan Noorani

October 26, 2020

Contents

1 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]
```

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'	97	'a'	97	66	'B'
'v'	118	'r'	114	104	'h'
'i'	105	'i'	105	82	'R'
'd'	100	'o'	111	83	

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 7.1. - Remember that you can use `ord` and `chr` to convert back and forth between characters and numbers.
- `%` has higher precedence than `+/-`, so you'll probably 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.

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

    >>> encrypt_otp('david', 'HELLO')
```

```python
    ',&B53'
    """
    ciphertext = ''

    for i in range(len(plaintext)):
        ciphertext = ciphertext + chr((ord(plaintext[i]) + ord(k[i])) % 128)

    return ciphertext


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

    Precondtions:
        - all({ord(c) < 128 for c in ciphertext})
        - all({ord(c) < 128 for c in k})

    >>> decrypt_otp('david', ',&B53')
    'HELLO'
    """
    plaintext = ''

    for i in range(len(ciphertext)):
        plaintext = plaintext + chr((ord(ciphertext[i]) - ord(k[i])) % 128)

    return plaintext
```

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

```python
>>> plaintext = 'David'
>>> key = 'Mario'
>>> ciphertext = encrypt_otp(key, plaintext)
>>> decrypt_otp(key, ciphertext) == plaintext
True
```

2 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 and 2)?

```python
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 information.

    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
    """
    possible_powers_a = []
    possible_powers_b = []

    # NOTE: You could also use a while loop and exit after the first a
    #       and then use another one for the first b
    for i in range(1, p):
        if g_a = (g ** i) % p:
            possible_powers_a.append(i)
        if g_b = (g ** i) % p:
            possible_powers_b.append(i)

    a = possible_powers_a[0]
    b = possible_powers_b[0]

    return (g ** (a * b)) % p
```