Cryptography: Algorithms and Keys Explained

In Python, the `bytes` type is an immutable sequence of integers in the range 0 <= x < 256. They are primarily used for handling binary data, interacting with network protocols, reading/writing binary files, or dealing with low-level cryptography.

Here are the basic operations and methods you need to know when dealing with `bytes` in Python:

1. Creating Bytes

You can create `bytes` objects in a few different ways:

Python

```
# 1. Using a bytes literal (prefixing a string with 'b')
b1 = b'hello'

# 2. Using the bytes() constructor with an iterable of integers (0-255)
b2 = bytes([104, 101, 108, 108, 111])  # b'hello'

# 3. Creating an empty bytes object of a specific length (filled with null bytes)
b3 = bytes(5)  # b'\x00\x00\x00\x00\x00'
```

2. Converting Between Strings and Bytes (Encode / Decode)

Strings in Python are sequences of Unicode characters, while bytes are sequences of raw 8-bit values. You must encode strings into bytes, and decode bytes back into strings.

Python

```
text = "Python "

# String to Bytes (Encoding)
b_data = text.encode('utf-8')
print(b_data)  # Output: b'Python \xf0\x9f\x90\x8d'

# Bytes to String (Decoding)
decoded_text = b_data.decode('utf-8')
print(decoded_text)  # Output: 'Python '
```

3. Accessing and Slicing Elements

Important Note: When you access a single element via indexing, Python returns an **integer**, not a byte character. However, slicing returns a new `bytes` object.

Python

```
b = b'Python'

# Indexing returns an integer (ASCII value)
```

```
print(b[0])      # Output: 80 (ASCII for 'P')

# Slicing returns a bytes object
print(b[0:2])    # Output: b'Py'
```

4. Concatenation and Repetition

Just like strings, you can add and multiply bytes.

Python

```
# Concatenation
b_concat = b'Hello' + b' ' + b'World'
print(b_concat)  # Output: b'Hello World'

# Repetition
b_repeat = b'AB' * 3
print(b_repeat)  # Output: b'ABABAB'
```

5. Hexadecimal Conversions

When working with binary data (like hashes or cryptography), you often need to convert between `bytes` and hexadecimal strings.

Python

```
# Bytes to Hex String
b_mac = b'\x00\x1A\x2B\x3C\x4D\x5E'
print(b_mac.hex())            # Output: '001a2b3c4d5e'
print(b_mac.hex(':'))         # Output: '00:1a:2b:3c:4d:5e' (Python 3.8+)

# Hex String to Bytes
hex_str = "deadbeef"
b_from_hex = bytes.fromhex(hex_str)
print(b_from_hex)             # Output: b'\xde\xad\xbe\xef'
```

6. Searching and Replacing

Most string methods are also available for `bytes` objects, but their arguments **must also be bytes**.

Python

```
b_text = b'apple banana apple'

# Searching
print(b_text.find(b'banana'))  # Output: 6
print(b'apple' in b_text)      # Output: True

# Replacing
print(b_text.replace(b'apple', b'orange')) # Output: b'orange banana orange'

# Splitting
print(b_text.split(b' '))      # Output: [b'apple', b'banana', b'apple']
```

7. Mutability (Using `bytearray`)

Because `bytes` are immutable, you cannot change an element once created (e.g., `b[0] = 65` will raise a `TypeError`). If you need to manipulate bytes in place to save memory, use a `bytearray`:

Python

```
mut_b = bytearray(b'hello')
mut_b[0] = 74                # Change 'h' (104) to 'J' (74)
print(mut_b)                 # Output: bytearray(b'Jello')

# Convert back to immutable bytes when done
final_b = bytes(mut_b)
```