Vigenère Cipher

Mark Slater (slides by Ben Morgan and Tom Latham)

UNIVERSITY OF BIRMINGHAM

THE UNIVERSITY OF WARWICK

The Vigenère Cipher

- A polyalphabetic substitution cipher
    - *The rule to substitute characters changes with each character in the input text*

- Originally described in 1553 by Giovan Battista Bellaso, but mistakenly attributed to Blaise de Vigenère (1586) by 19th Century cryptographers.

- Though occasionally broken before the 19th century, no published formal attack until Kasiski and Babbage in the mid 1800s.

Vigenère Cipher Encryption Substitution Rule

- Choose a **Keyword W** [1, N] characters long.

- Pair each character in **Keyword** with character in **Plaintext**, repeating/truncating **Keyword** if it is shorter/longer than **Plaintext**.

- Replace each character in **Plaintext** by **encrypting** it with a **CaesarCipher** of **Shift** equal to the position in the alphabet of the **Keyword** character that is paired with the **Plaintext** character

Encrypting With the Vigenère Cipher, W=KEY

Plaintext HELLOWORLD

Keyword KEYKEYKEYK

CaesarEncrypt('H',10)

CaesarEncrypt('O',4)

Ciphertext RIJVSUYVJN

Vigenère Cipher Decryption Substitution Rule

- Choose a **Keyword W** [1, N] characters long.

- Pair each character in **Keyword** with character in **CipherText**, repeating/truncating **Keyword** if it is shorter/longer than **CipherText**.

- Replace each character in **Plaintext** by **decrypting** it with a **CaesarCipher** of **Shift** equal to the position in the alphabet of the **Keyword** character that is paired with the **CipherText** character.

Decrypting With the Vigenère Cipher, W=KEY

Ciphertext RIJVSUYVJN

Keyword KEYKEYKEYK

CaesarDecrypt(‘R’,10)

CaesarDecrypt(‘S’,4)

Ciphertext HELLOWORLD

Exercise 1 – Add Vigenère Boiler Plate

- As with the Playfair cipher, we'll start with putting the boiler plate in that we'll fill in afterwards
- You will need to:
  1. Allow the user to give 'vigenere' as an argument to the '--cipher' command-line option
  2. Create a basic `VigenereCipher` class skeleton that contains a `std::string` member variable called `key_` and the function signatures given on the next slide
  3. When given on the command line, create a `VigenereCipher` object with the given key and call the `applyCipher` function
  4. Don't forget to add documentation and some initial tests!

Exercise 1 – Vigenère Function Signatures

```
void VigenereCipher::setKey( \
    const std::string& key )
{

}
```

```
VigenereCipher::VigenereCipher ( \
    const std::string& key )
{
    // Set the given key
    setKey(key);
}
```

```
std::string VigenereCipher::applyCipher( const std::string& inputText, \
    const CipherMode /*cipherMode*/ ) const
{
    return inputText;
}
```

Composition in C++

Mark Slater (slides by Tom Latham)

UNIVERSITY OF BIRMINGHAM

THE UNIVERSITY OF WARWICK

Composition of objects

- As we've mentioned before, re-using code is a good thing
  - Avoids duplicating code and hence reduces the burden of maintenance and the likelihood of bugs creeping in
- Object composition is an excellent way of re-using already tested code
- Composition means having a data member of a class that is itself an instance of another class
- That data member can then be used by the containing class to help perform some of its work for it
- We've actually already been doing this when we have containers and strings as data members of our cipher classes

```cpp
#include <vector>
#include "Employee.hpp"

class ProjectTeam {

 ...

 private:
  /// The leader of the team
  Employee teamLeader_;

  /// Other members of the team
  std::vector<Employee> team_;
}
```

Using composition in the Vigenère cipher implementation

- We've seen that the Vigenère cipher algorithm involves using a series of Caesar ciphers with different keys

- We don't want to re-implement the Caesar cipher algorithm within our Vigenère cipher class, we want to be able to reuse the code we've already written and tested

- We can do so by having data members of the `VigenereCipher` class that are themselves object instances of the `CaesarCipher` class

- We can then delegate some of the work of performing the encryption to those objects

Using composition in the Vigenère cipher implementation

- In particular, we want to have a number of `CaesarCipher` objects that are each associated with a character in the key word
- We can do this by using the `std::map`, which we used last week, to create a lookup table
- This table should be filled in the `setKey` member function
- Then in the `applyCipher` function it can be used to retrieve the `CaesarCipher` objects, which can then be used to encrypt the input text one letter at a time

```
#include <map>
#include <string>
#include "CaesarCipher.hpp"


class VigenereCipher {

 ...

 private:
  /// The cipher key
  std::string key_ = "";

  /// Lookup table
  std::map<char,CaesarCipher> charLookup_;
}
```

Exercise 2 - Complete Implementation of Vigenère Cipher

- We've now got a better idea how to implement the Vigenere Cipher, so make the following changes:

1. Add the lookup map member variable as in the previous slide

2. Put in the comment changes as shown in the next slide

3. Attempt to implement the functions as described in the comments

Exercise 2 – Vigenère Function Signatures

```
void VigenereCipher::setKey( \
    const std::string& key )
{
    // Store the key
    key_ = key;

    // Make sure the key is uppercase

    // Remove non-alphabet characters

    // Check if the key is empty and
replace with default if so

    // loop over the key
        // Find the letter position in the
alphabet

        // Create a CaesarCipher using
this position as a key

        // Insert a std::pair of the
letter and CaesarCipher into the lookup
}
```

```
std::string VigenereCipher::applyCipher( \
    const std::string& inputText, \
    const CipherMode /*cipherMode*/ )
const
{
    // For each letter in input:

        // Find the corresponding letter
in the key,
        // repeating/truncating as
required

        // Find the Caesar cipher from the
lookup

        // Run the (de)encryption

        // Add the result to the output

    return inputText;
}
```