iCipher, un'applicazione in C++ per cifrare frasi e parole con delle rivisitazioni degli algoritmi più famosi

Lorenzo LEONI

Università degli studi di Bergamo, Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

1 maggio 2023

1 Introduzione

Esiste un modo per complicare e per rendere meno vulnerabili alcuni degli algoritmi di cifratura a sostituzione monoalfabetica più rinomati quali Cesare e Vigenère? L'obiettivo di iCipher è proprio questo, ovvero fornire all'utente un'applicazione in C++ attraverso la quale possa cifrare e decifrare frasi e parole utilizzando degli algoritmi più complessi, derivanti da quelli sopracitati.

2 Funzionalità

Attraverso un'interfaccia a riga di comando, l'utente ha la possibilità di accedere tramite iCipher a due servizi:

- **cifratura** e **decifrazione** di sequenze di caratteri tramite il cifrario di Cesare e la sua versione modificata, l'algoritmo di Vigenère e il cifrario di Leoni, una tecnica che itera n volte le precedenti;
- **memorizzazione** in un file CSV delle versioni criptate delle parole che sono state inserite nel programma. Per ognuna di esse vengono specificati anche i parametri dell'algoritmo di cifratura che è stato utilizzato.

L'applicazione opera completamente offline, pertanto non si corre il rischio di diffondere in rete i propri dati sensibili.

3 Architettura

Le classi necessarie per l'implementazione delle funzionalità di iCipher sono state raggruppate in 3 file header:

- **encryptors.h** (figura 1);
- **application.h** e **consts.h** (figura 2);

Tutte le classi (fatta eccezione per `Application`) sono state definite utilizzando il costrutto `struct` piuttosto che `class`; tale scelta risiede nel voler tenere meglio sotto controllo la visibilità dei campi, dei metodi e la tipologia di ereditarietà essendo `public` di default con il costrutto `struct`.

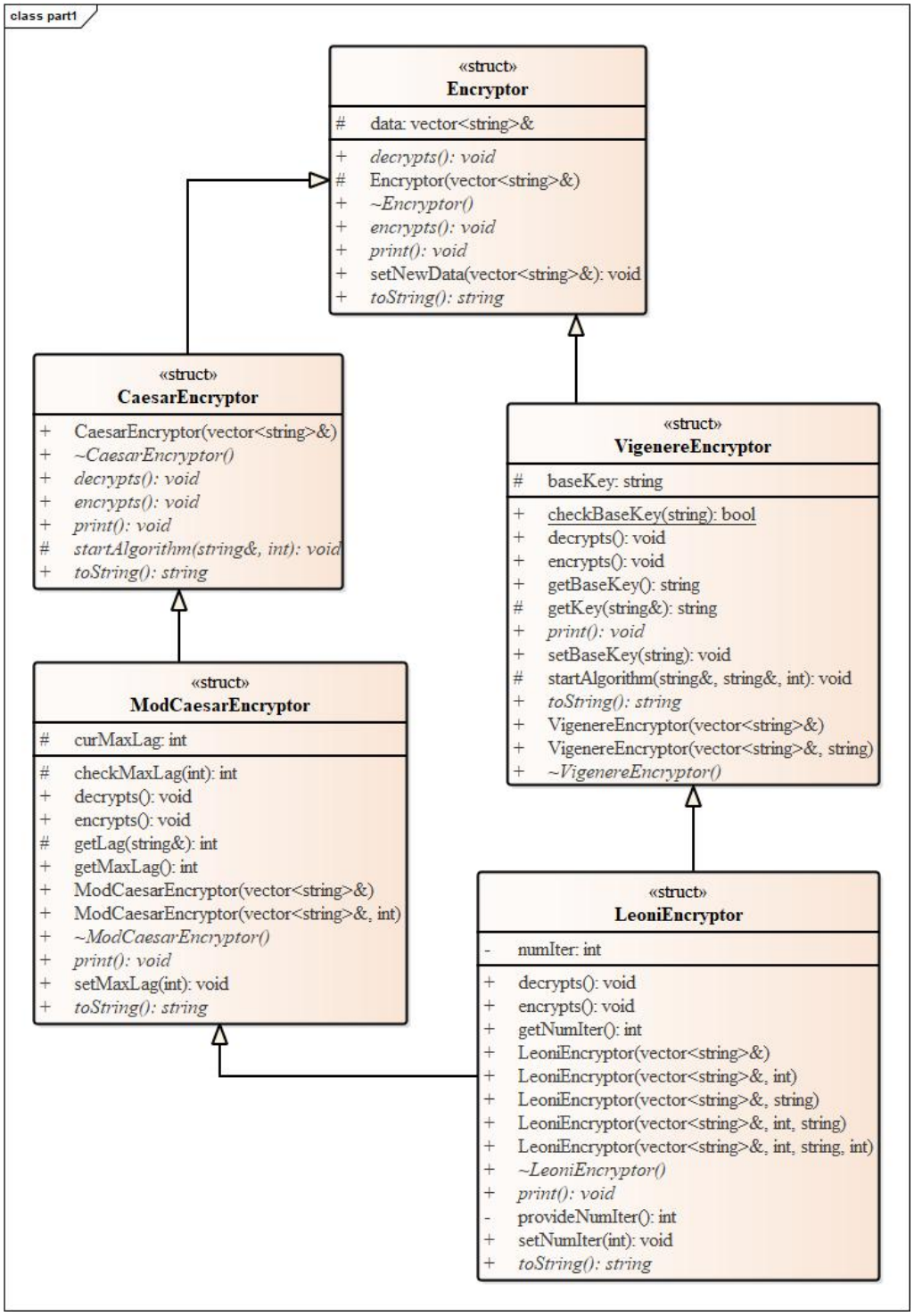

Figura 1: diagramma UML delle classi dichiarate in *encryptors.h*.

3.1 Header encryptors

La gerarchia tra le classi aventi il compito di implementare gli algoritmi di cifratura è stata organizzata in modo tale da realizzare un *struttura a diamante*. La `struct Encryptor` definisce i prototipi delle funzioni `encrypts()`, `decrypts()`, `toString()` e `print()`; esse sono dichiarate `virtual`, pertanto le classi derivate possono implementarle. Da notare che il costruttore di `Encryptor` riceve come argomento il riferimento alla struttura dati di stringhe `vector<string>` contenente le parole da cifrare o decifrare; ciò permette ai metodi `encrypts()` e `decrypts()` di agire direttamente sul vettore target, senza la necessità di creare una sua copia locale. Proseguendo nella descrizione della gerarchia, si osserva che sia `CaesarEncryptor` sia `VigenereEncryptor` ereditano pubblicamente e *virtualmente* dalla classe base. Tale scelta è necessaria affinché `LeoniEncryptor` erediti una sola volta da `Encryptor` (per via dell'ereditarietà multipla), ovvero si vuole evitare che vengano a crearsi

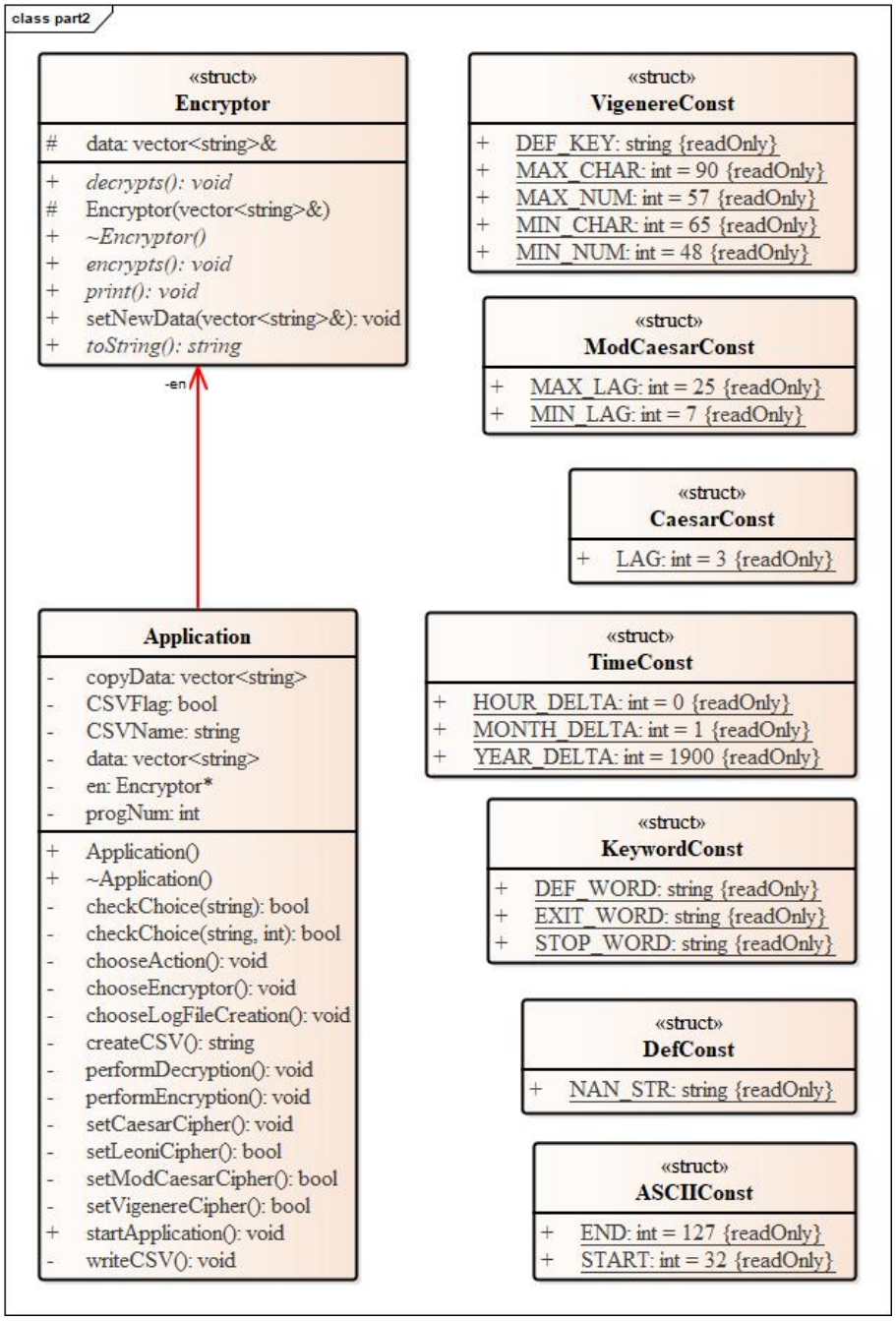

Figura 2: diagramma UML delle classi dichiarate in *application.h* e in *const.h*.

due istanze distinte di quest'ultima classe nell'istante in cui viene invocato uno dei costruttori di `LeoniEncryptor`. Infine, nel diagramma UML non è specificato, però è importante sottolineare una differenza: `CaesarEncryptor` e `VigenereEncryptor` *implementano* le interfacce dei metodi della classe `Encryptor`, mentre `ModCaesarEncryptor` effettua l'*overriding* degli stessi poiché ne eredita le implementazioni dalla sua classe base `CaesarEncryptor`.

3.2 Header application e consts

La classe `Application` è il cuore di iCipher, l'hub dal quale è possibile accedere a tutte le funzionalità dell'applicazione in C++. Il campo `en` è un puntatore a un oggetto di tipo `Encryptor`; grazie alla gerarchia sopracitata, è possibile assegnare alla variabile `en` il riferimento a un qualsiasi oggetto che eredita, direttamente o indirettamente, da `Encryptor`, istanza che cambia a seconda del cifrario che l'utente desidera utilizzare. Ciò è reso possibile dal *binding dinamico* (il quale viene abilitato dichiarando `virtual` i metodi di `Encryptor`): il compilatore risolve l'overriding in fase di esecuzione (*late binding*). Le classi definite in *consts.h*, invece, servono per fissare le costanti che vengono

utilizzate dall'applicazione e dagli algoritmi di cifratura, per esempio gli intervalli di caratteri della tabella ASCII che possono essere utilizzati per la cifratura e per la definizione dei chiave per il cifrario di Vigenère. La decisione di creare un file header apposta per le costanti risiede nella comodità di poterle modificare all'occorrenza da un unico file, senza così andare a rintracciarle singolarmente all'interno del codice.

4 Cifrario di Leoni

L'algoritmo si basa sull'utilizzo combinato e iterato dei cifrari di Cesare modificato[1] e di Vigenère. Gli algoritmi 1 e 2 descrivono rispettivamente la cifratura e decifrazione. Si ricorda che la versione modificata dell'algoritmo di Cesare aggiunge un carattere in coda alla parola criptata in modo tale che ci sia la possibilità di risalire al *lag* (scelto casualmente) utilizzato, quindi n iterazioni comportano l'aggiunta di n caratteri alla parola originale.

nIter $\leftarrow$ *numero di iterazioni*;
data[ ] $\leftarrow$ *vettore di stringhe da criptare*;
key $\leftarrow$ *chiave crittografica per il cifrario di Vigenère*; `/* es. ULIVETO11 */`
$i \leftarrow 0$;
while $i < length(data)$ **do**
  $data[i] \leftarrow encrypts(data[i],$ *"Vigenère"*, $key)$;
  $data[i] \leftarrow encrypts(data[i],$ *"ModCaesar"*$)$;
  $i = i + 1$;
end

Algoritmo 1: algoritmo di cifratura del cifrario di Leoni

nIter $\leftarrow$ *numero di iterazioni*;
data[ ] $\leftarrow$ *vettore di stringhe da criptare*;
key $\leftarrow$ *chiave crittografica per il cifrario di Vigenère*;
$i \leftarrow 0$;
while $i < length(data)$ **do**
  $data[i] \leftarrow decrypts(data[i],$ *"ModCaesar"*$)$;
  $data[i] \leftarrow decrypts(data[i],$ *"Vigenère"*, $key)$;
  $i = i + 1$;
end

Algoritmo 2: algoritmo di decifrazione del cifrario di Leoni

5 Documento CSV

Se si desidera, iCipher dà la possibilità di salvare le versioni crittografate delle proprie parole o frasi all'interno di un file CSV. Per ognuna di esse vengono specificati anche i parametri dell'algoritmo che è stato utilizzato. Di seguito è riportato un esempio di documento CSV:

```
ProgNum,Password,EncryptorType,MaxLag,BaseKey,NumIter,(CryptedPw)
1,Lor3nzo-1998,Caesar cipher,3,//,//,(Oru6q}r04<<;)
2,Lor3nzo-1998,Modified Caesar cipher,25,//,//,(X{~?z&{9=EED>)
3,Lor3nzo-1998,Vigenere cipher,//,ULIVETO11,//,(A[[)Sn^^b.%!)
4,Lor3nzo-1998,Leoni cipher,25,ULIVETO,7,(gK9Uy.‘Hm‘[CBM";.JW)
```

[1] per una descrizione accurata dell'algoritmo si rimanda alla documentazione di PasswordCollector, sezione 5: `https://github.com/lamferzon/Password-collector-in-Java/blob/main/documentazione.pdf`

6 Repository GitHub

Il codice sorgente di iCipher è disponibile per il download al seguente repository:

`https://github.com/lamferzon/iCipher`

```
** iCipher **
Copyright 2023 Lorenzo Leoni (UniBG)

Do you want to create a log file? (Y/N) N

Which encryptor do you want to use?
1. Caesar cipher.
2. Modified Caesar cipher.
3. Vigen├¿re cipher.
4. Leoni cipher.
0. Exit.
Your choice: 4

Leoni cipher settings (QUIT to exit):
- Num. iterations (DEF to default (7)): 20
- Maximum lag (DEF to default (25)): DEF
- Base key (DEF to default (ULIVETO)): LAMFERZON17
```

(a)

```
What do you want to do?
1. Encrypting.
2. Decrypting.
0. Come back.
Your choice: 1

Insert passwords (QUIT to exit, STOP to stop):
0) Lorenzo-1998
1) 2020_Babb0
2) STOP

Crypted passwords:
0) Lorenzo-1998
-> =Dw>3#XZJNf)NG7)6U+=7o}'yf0pr.op
1) 2020_Babb0
-> a3e GF+wADOUqh078#x*OF!_3Grxa
```

(b)

Figura 3: un paio di screenshot dell'applicazione in esecuzione.