



Steganografia in un file di testo

Corso di Sicurezza dei sistemi informatici

Michelangelo Rinelli

Anno Accademico 2005/06

Steganografia

È l'arte di nascondere un messaggio all'interno di un altro messaggio

Stèganos = nascosto

Gràfein = scrivere





Steganografia in file di testo

Codifica del messaggio all'interno di un testo per mezzo di spazi o tabulazioni (*whitespaces*)

“Proof of Concept”

L'applicazione è il tentativo iniziale di sviluppare un'idea e lascia spazio a molte migliorie di implementazione.



Punti di forza

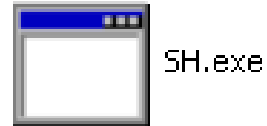
- Gli spazi bianchi non vengono visualizzati dalla maggior parte dei software di scrittura
- Spazi bianchi alla fine delle righe occorrono naturalmente in un file di testo.
- Le nuove tecnologie web-based sono basate su file di testo che vengono interpretati da software e non da umani (es. HTML,XML)



Punti deboli

- E' possibile nascondere pochi dati alla volta
- Notevole cambio di dimensione del file

SpaceHider



Sviluppata in ANSI C 99

- Lo standard ANSI permette il *porting* con facilità da una piattaforma all'altra
- L'uso di un linguaggio compilato garantisce prestazioni di tutto riguardo, oltre all'indipendenza da *framework* o *virtual machines*



Funzionamento

- L'applicazione nasconde il messaggio tramite spazi e tabulazioni alla fine della riga

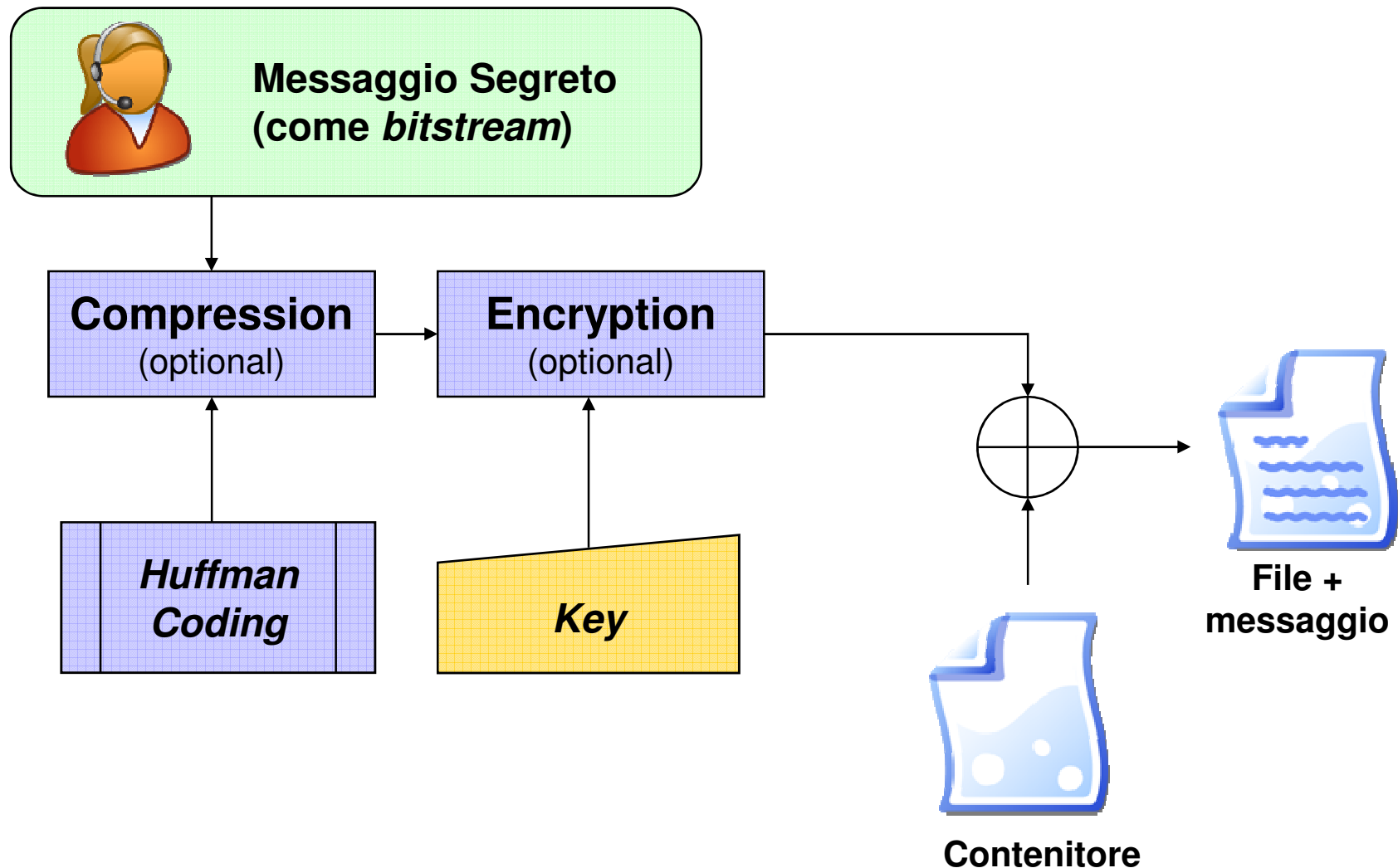
Il messaggio da steganografare viene considerato come un flusso di bit



001011001011010011011001000

Tale flusso viene quindi sottoposto a vari processi, fino ad essere nascosto all'interno del contenitore

Funzionamento (codifica messaggio)





Compressione

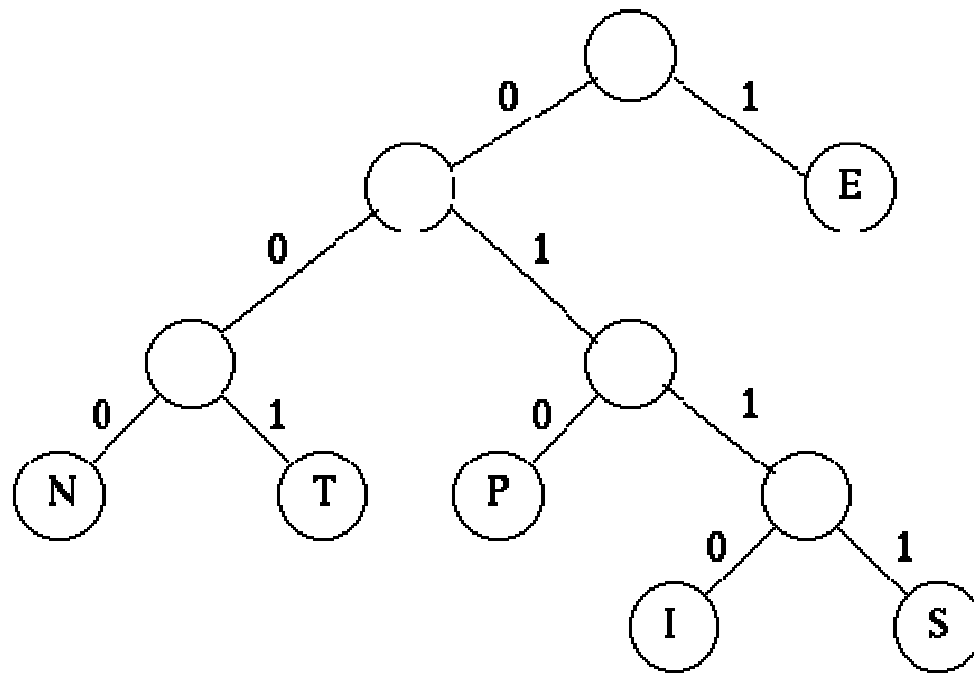
Vista la dimensione ridotta dei messaggi che è possibile codificare è necessario utilizzare un algoritmo di compressione.

Codice di Huffman

- Pesato secondo la frequenza dei caratteri nella lingua
- Non richiede *header* o *footer* (overhead nullo)

Huffman coding

In base alle frequenze di occorrenza delle lettere in un alfabeto si costruisce un albero da cui si ricavano le codifiche **a lunghezza variabile**



Esempio: 10111001 = "EST"

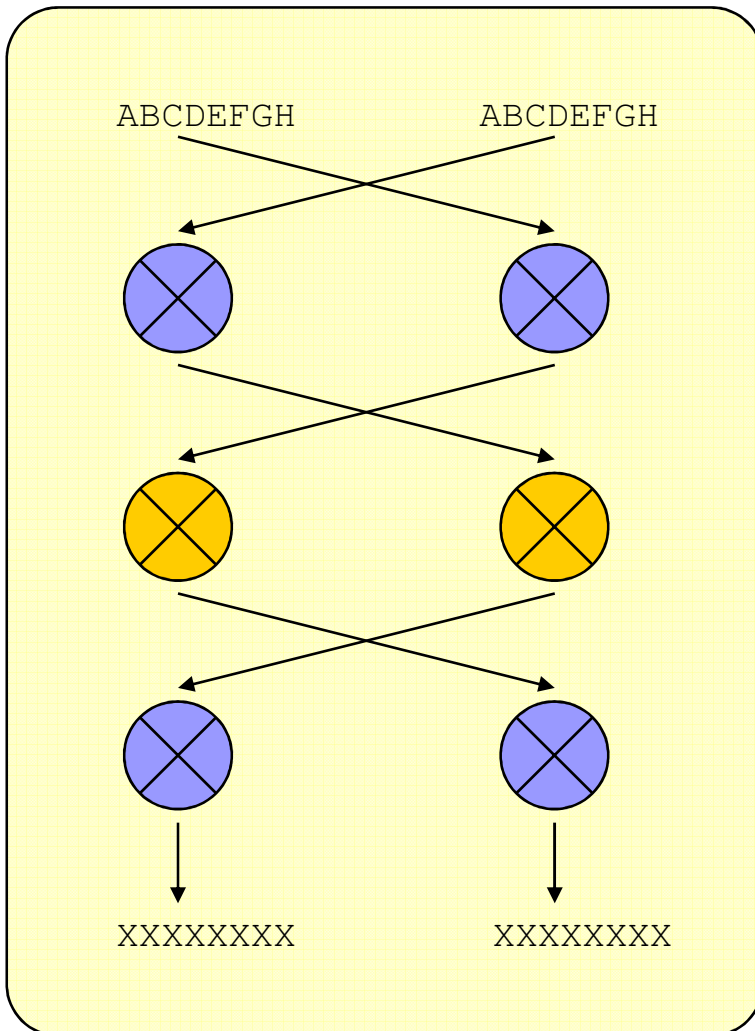


Quando comprimere?

Nei messaggi in cui sono molto frequenti caratteri generalmente non frequenti la compressione con Huffman potrebbe essere **controproducente**.

Il programma calcola al volo il tasso di compressione che si ottiene, e se questo risulta svantaggioso consiglia di non utilizzare la compressione.

Cifratura



E' stato sviluppato un semplice algoritmo di tipo *cipher block* a chiave *simmetrica*: **buruCrypto**

- *Key length: 64bit*
- *Cipher size: 16bit*

Dalla chiave vengono generate 4 sottochiavi, alternate secondo un preciso criterio (*key scheduling*)



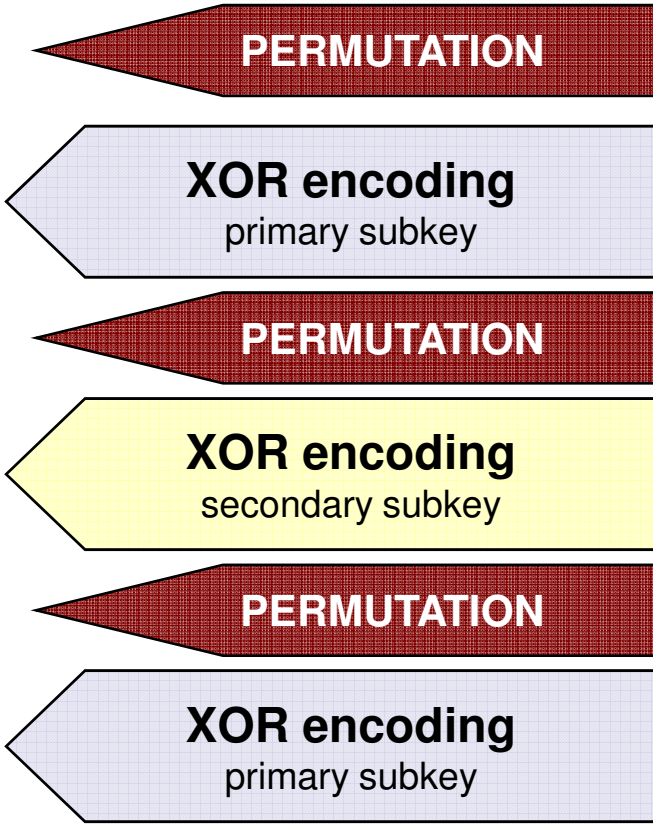
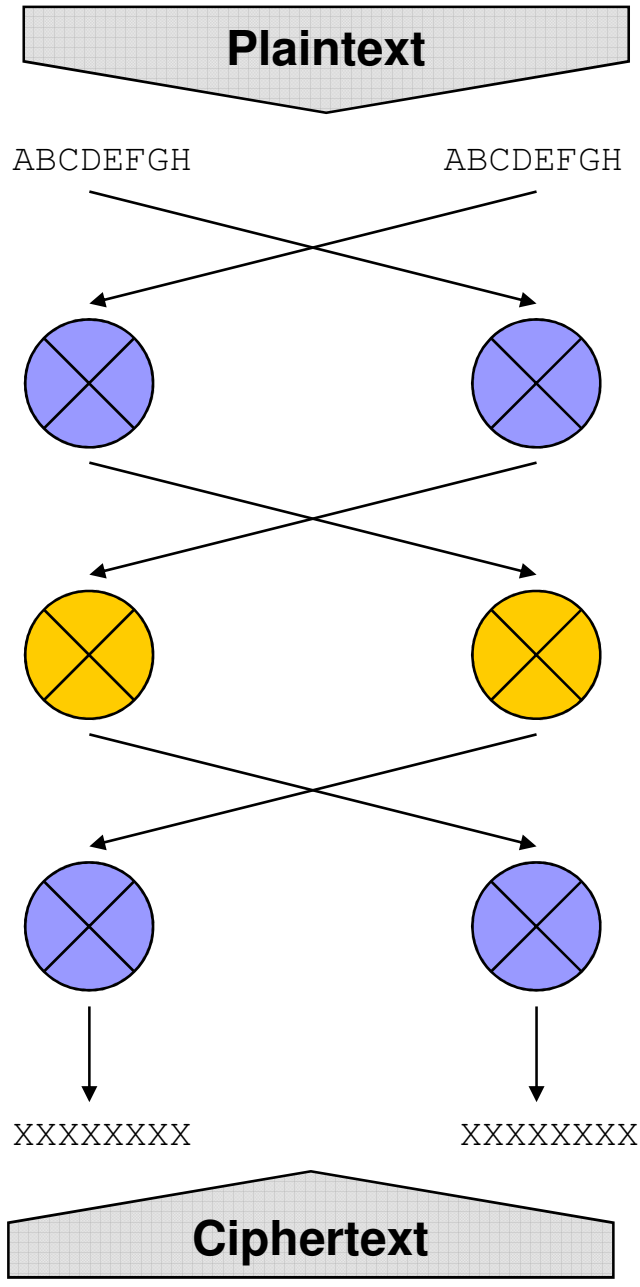
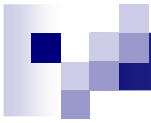
buruCrypto è stato sviluppato come una libreria esterna:

- riutilizzabile in altri ambiti
- sostituibile all'interno di spaceHider con algoritmi di cifratura più sicuri

Si è preferito non utilizzare DES o Triple-DES a causa del chiper-block di dimensioni relativamente grandi (*64bit*)

Interfaccia di buruCrypto

```
void b_init(void);  
void b_encrypt(const byte*, byte*, int);  
void b_decrypt(const byte*, byte*, int);  
void b_encrypt_string(const char*, char*, int len)  
void b_decrypt_string(const char*, char*, int len)  
void b_setkey(const char*);  
chipper_block b_getsubkey(int);
```





Cifratura e codifica di Huffman

Il codice di Huffman è a lunghezza variabile, quindi in generale produce serie di bit di lunghezza diversa da 8 (1 byte)

Le permutazioni dell'algoritmo di cifratura coinvolgono più caratteri, rimescolandoli tra loro

Un algoritmo di tipo *streaming* avrebbe generato un bitstream criptato con bit incorrelati tra di loro

Il codice di Huffman utilizzato, può essere mantenuto segreto ed essere considerato un *cifrario* di una ulteriore fase crittografica



Message Hiding

Il *bitstream* viene diviso in blocchi da 3 bit ciascuno ed ognuno dei blocchi viene codificato come

$$n^{\circ}\text{spazi}_{(\text{dec})} + '\backslash t'$$

Es: la tripletta di bit 010 è codificata come “. . . \t”, 3 spazi seguiti da una tabulazione



Message Hiding

I blocchi codificati vengono inseriti in coda alle righe del testo fino al raggiungimento di un valore critico di lunghezza della riga (default = 70 car)

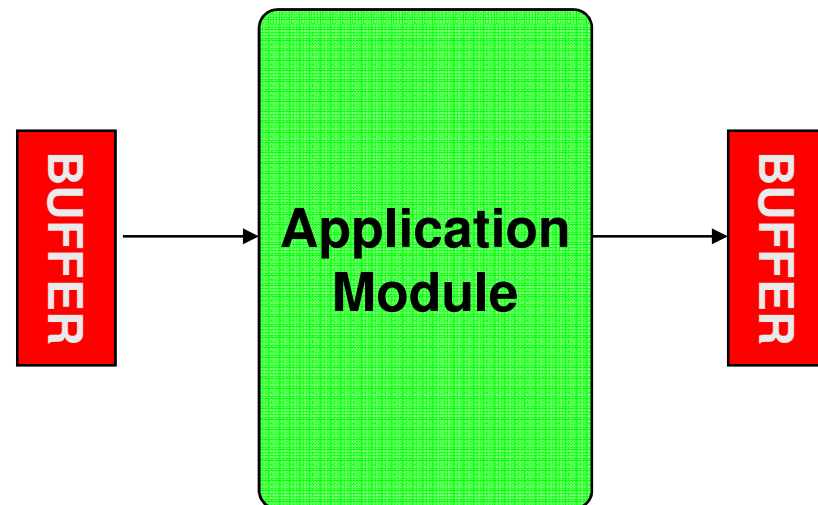
Ogni file di testo ha quindi un valore massimo di informazioni che è in grado di contenere.

NOTA: E' possibile specificare il valore critico di riga in fase di "iniezione" del messaggio nel cover

Interfacce

Le interfacce tra i moduli hanno dei buffer di ingresso e dei buffer di uscita, in quanto le operazioni sono svolte su gruppi di bit

- *Codifica: 3bit;*
decodifica: 3bit;
- *Cifratura: 16bit;*
decifratura: 16bit;
- *Compressione: 8bit*
Decompressione: variabile





Utilizzo di sh

Il tool viene utilizzato da riga di comando

```
C:\SH\> sh
Usage: sh [-C] [-Q] [-S] [-p passwd] [-l line-len]
                                     [-f file | -m message]
                                     [infile [outfile]]
```

- C**: Abilita compressione
 - Q**: Quiet mode: non mostra messaggi durante l'esecuzione
 - S**: Calcola una stima dello spazio libero in un file
 - p** <password>: se specificata cripta il bitstream
 - l** <line-length>: specifica la lunghezza critica di linea
 - f** <file> | -**m** <message>: Sorgente del messaggio (file di testo o stringa)
- infile**: file di input
outfile: file di output

[DIMOSTRAZIONE](#)



E se...

Il messaggio supera la capienza del file *cover*?

SpaceHider aggiunge delle righe in fondo al file

Il file contiene già spazi bianchi alla fine delle righe?

*SpaceHider provvede a toglierli prima di inoculare il
messaggio*



Immagini in un file di testo?

E' facile vedere che è possibile estendere il metodo di steganografia sviluppato in modo da nascondere all'interno di un *cover* di testo file binari.

Sarebbe necessario:

- Differenziare le modalità di lettura/scrittura dei file
 - Sviluppare un codice di Huffman efficiente per i binari
- non sarebbe comunque possibile nascondere file troppo "pesanti"