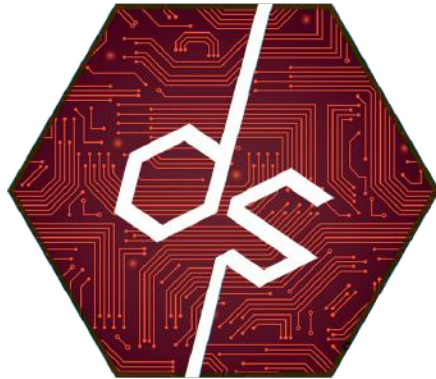


Arduino Steganografo

Progetto Architettura degli Elaboratori

Group Project: **DekketScrib-Team**

Documento del progetto



Corso L-31
a.a.2020/2021
Università degli studi di Camerino
Italia, Camerino (MC)
Data 2021-06-07

DekketScrib-Team:

CISCA SIMONE, Università degli Studi di Camerino
CALVARI MAURIZIO, Università degli Studi di Camerino
BIALOWAS PATRYK, Università degli Studi di Camerino
BURDIMOV DUMITRU, Università degli Studi di Camerino
CRUCIANI FEDERICO MARIA, Università degli Studi di Camerino
SALPINI LEONARDO, Università degli Studi di Camerino

Abstract

Il linguaggio Assembly è il linguaggio di programmazione più vicino al linguaggio macchina.

Il vantaggio principale è l'utilizzo diretto dei registri e, di conseguenza, una maggiore velocità di lettura e scrittura sugli stessi rispetto agli altri linguaggi, prenderemo come riferimento per le nostre comparazioni il linguaggio C.

L'obiettivo del progetto è di dimostrare, tramite l'utilizzo di un Arduino Uno, la differenza temporale di esecuzione. Testeremo le due funzioni Hiding e Unveiling con il codice sorgente scritto completamente in C e con alcune istruzioni sostituite in linguaggio Assembly.

I test verranno eseguiti su dei file di dimensioni diverse.

INDICE

INDICE	2
1 Introduzione	3
1.1 Bitmap	3
1.2 Limitazioni d'uso	5
1.3 Utenza Finale	6
1.4 Utilizzo del Sistema	6
2 Funzionalità del Sistema	7
2.1 Scelta: Hiding o Unveiling	7
2.2 Micro SD	7
3 Schema Circuitale	7
3.1 Circuito logico	7
3.2 Case	8
3.3 LED RGB	8
3.4 Pulsante	9
3.5 Switch	9
3.6 SD-R	9
4 Approccio Fallimentare	9
5 Funzione Hiding	9
5.1 Algoritmo	10
5.2 Codice	10
6 Funzione Unveiling	12
6.1 Algoritmo	12
6.2 Codice	12
7 Problemi	13
7.1 Formati Bitmap	13
7.2 Formati Testo	14
7.3 Collo di bottiglia	15
8 Grafici	15
8.1 Hiding	15
8.2 Unveiling	16
9 Conclusioni	17
10 Impatto Covid-19 sul progetto	17
11 Riferimenti	18

1 INTRODUZIONE

La possibilità di utilizzare un Arduino ed eventuali componenti da implementare ha posto, a noi membri, l'interesse nel creare un progetto utile e divertente su cui lavorare. In principio pensammo di creare un telecomando per la gestione della domotica domestica, cioè un telecomando universale utilizzabile su apparecchiature elettroniche tramite l'utilizzo di infrarossi. In seguito alla discussione di altre idee, concludemmo che lo steganografo iniettivo sarebbe stato un ottimo strumento per raggiungere il nostro obiettivo. La steganografia (2) ha come scopo principale quello di rendere sicura la comunicazione, "invisibile" l'atto comunicativo in sé. L'origine del termine risale all'antica Grecia e viene tradotto come "scrittura nascosta".

Un semplice ma efficace metodo di steganografia nell'antichità, visto il frequente uso delle tavolette di cera, era quello di incidere il messaggio nel fondo legnoso della tavoletta, per poi scrivere sulla cera un messaggio diversivo.

L'evoluzione tecnologica ha portato l'Uomo ad ingegnarsi per poter utilizzare questo potente strumento spostandosi da mezzi fisici a digitali sia per la trasmissione di messaggi sia per il watermarking dei prodotti.

Esistono varie forme di steganografia, tra queste: iniettiva, generativa, sostitutiva, selettiva e costruttiva. La steganografia da noi applicata è quella iniettiva, poiché si adatta meglio al nostro compito, ovvero nascondere informazioni nei pixel delle immagini.

Dopo varie ricerche, abbiamo scelto di utilizzare come formato del file immagine il **BMP Version 3 (Microsoft Windows NT)** (3) e versioni successive in quanto in questa versione, in particolare nelle codifiche a 24 e 32 bit, le componenti dei colori sono descritte da un Byte ciascuna, rendendo possibile modificare i bit meno significativi senza alterare l'aspetto dell'immagine. Le immagini bitmap supportate sono quindi quelle a 24 bit e a 32 bit, poiché modificandone una a 8, 4 o 1 bit il cambiamento sarebbe palese.

1.1 Bitmap

Il formato BMP è uno dei formati più semplici, sviluppato congiuntamente da Microsoft e IBM. Un file bitmap registra un'immagine sotto forma di tabella di punti (pixel). Gestisce i colori sia in RGB oppure mediante una palette indicizzata.

Il file contenente un'immagine bitmap può essere scomposto in tre parti principali:

- **Intestazione:** contiene informazioni utili per interpretare il file e costruire l'immagine, come le dimensioni, il numero di colori utilizzati, il numero di pixel, dove cominciano le informazioni riguardanti l'immagine vera e propria.
- **Palette dei colori:** contiene informazioni sui colori dell'immagine a bassa profondità di colore (la profondità di colore indica il numero di bit usati per indicare il colore di un singolo pixel).
- **Bitmap:** i dati riguardanti l'immagine vera e propria.

Lo steganografo che abbiamo sviluppato, come detto in precedenza, è compatibile con i formati bitmap a 32 e 24 bit questo perché abbiamo deciso di utilizzare 4 Byte dell'immagine per nascondere un 1 Byte del testo. Di questi 4 Byte i primi tre vengono alterati mentre l'ultimo deve rimanere invariato.

Nel caso del formato a 32 bit (scritto solitamente nella notazione R.G.B.A.X, dove R rappresenta il byte del rosso, G il byte del verde, B il byte del blu e A.X. il byte dedicato al canale alpha) verrà modificato il bit meno significativo dei byte RGB. mentre verranno lasciati invariati A.X.

Questo perché se il canale alpha viene modificato l'immagine in output sarà corrotta o alterata rispetto all'immagine in input.

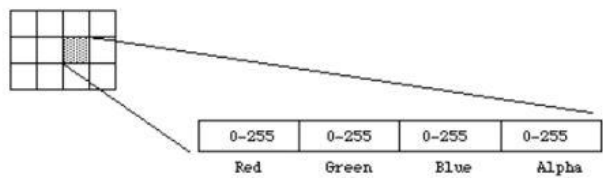


Fig. 1. Esempio di un pixel formato bitmap 32 bit

(4)

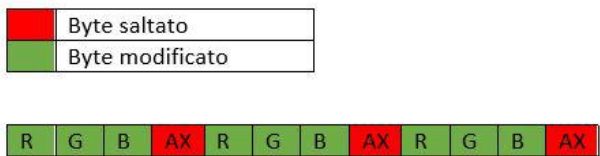


Fig. 2. Esempio grafico della modifica dei byte

Di conseguenza il formato a 24 bit funzionerà comunque anche se non è presente il Byte dedicato al canale alpha. Questo perché il Byte che verrà saltato sarà sempre il quarto che in questo caso sarà un Byte dedicato ad un colore tra R.G.B.

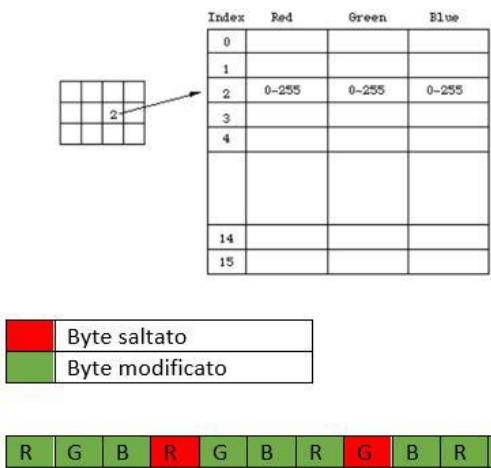


Fig. 3. Esempio di un pixel formato 24 bit e della modifica dei byte

(5)

1.2 Limitazioni d'uso

Come visto in precedenza per nascondere un carattere necessitiamo di 4 Byte, con la figura 4 rappresentiamo graficamente il nostro obiettivo, scomporre il byte del carattere per inserire ogni suo bit nel bit meno significativo dei byte RGB.

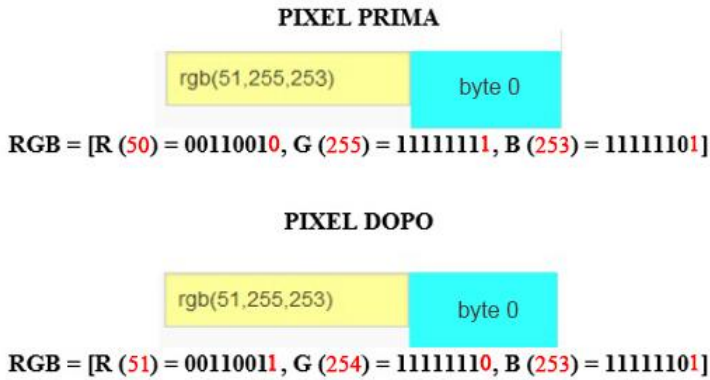


Fig. 4. Spiegazione grafica su come avviene la modifica dei byte
(6)

Ogni carattere del testo occupa al più 4 Byte dell'immagine.

Di conseguenza sorgono limitazioni riguardanti le dimensioni dell'immagine e del testo in input.

Di seguito sono riportate le formule per il corretto funzionamento. Siano:

- M la memoria SD,
- I_{txt} il file input.txt,
- I_{bmp} il file input.bmp,
- O_{txt} il file output.txt,
- O_{bmp} il file output.bmp,
- HP l'header e la palette di I_{bmp} .

Funzione di Hiding:

$$I_{txt} < \frac{1}{12}(I_{bmp} - HP)$$

Ogni byte di I_{txt} richiede almeno 4 pixel, quindi 12 Byte.

Ogni bit di I_{txt} prenderà il posto del bit meno significativo di alcuni byte di **I_{bmp}** (esclusi HP e il byte x -esimo quando x è un multiplo di 4) e inoltre verrà creato il file O_{bmp} .

$$I_{txt} + I_{bmp} < \frac{1}{2}M$$

Questo perché O_{bmp} dovrà coesistere con I_{txt} e I_{bmp} .

Funzione di Unveiling:

$$I_{bmp} < \frac{11}{12}M$$

In quanto dopo aver effettuato l'unveiling verrà creato il file O_{txt} che dovrà coesistere con il file I_{bmp} .

1.3 Utenza Finale

Il prodotto finale è rivolto a chi necessita uno scambio di informazioni.

La steganografia si pone come obiettivo di mantenere nascosta l'esistenza di dati a chi non conosce il protocollo di occultamento, creando così una rete dove solo chi possiede uno steganografo (che utilizza lo stesso algoritmo) può ottenere il messaggio in chiaro. Invece la crittografia consiste nel rendere inaccessibili i dati nascosti a chi non conosce la chiave, cioè di nascondere il contenuto del messaggio.

L'invio del file contenente il testo celato deve essere effettuato evitando qualsiasi tipo di compressione, che potrebbe annullare i bit meno significativi del file e quindi il messaggio.

1.4 Utilizzo del Sistema

È importante ricordare che per il corretto funzionamento del sistema è necessario utilizzare solamente immagini a 32 o 24 bit, mentre per quanto riguarda il file di testo si può utilizzare qualsiasi codifica, ma quella consigliata è UTF-8, che è la più comune e tutti i sistemi operativi sono in grado di interpretarla correttamente.

Lo steganografo è anche dotato di un led con il quale può mostrare lo stato in cui si trova.

- LED rosso: errore, file errati all'interno della M



- LED giallo: attesa input Hiding



- LED magenta: attesa input Unveiling



- LED blu-verde: elaborazione in corso



- LED verde intermittente: fine elaborazione, possibilità di rimozione M



ISTRUZIONI D'USO

- (1) Formattare M in FAT32.
- (2) Inserire in M i file di input: *Itxt* e *Ibmp* per l'Hiding, *Ibmp* per l'Unveiling.
- (3) Scegliere la modalità Hiding o Unveiling tramite lo switch; per la funzione Hiding il led diventerà **Giallo**, per la funzione Unveiling il led diventerà **Magenta**.
- (4) Inserire M nel lettore SD dello steganografo, in caso di illuminazione rossa da parte del LED, rimuovere M e tornare al passo 2. In caso di persistenza del problema tornare al passo 1.
- (5) Attendere che il LED blu passi al colore verde.
- (6) Al termine dell'esecuzione del programma, il LED verde lampeggerà, quindi rimuovere M .
- (7) Per riavviare l'esecuzione e tornare al passo 3 tenere premuto il pulsante fino a quando il led smetterà di lampeggiare.

2 FUNZIONALITÀ DEL SISTEMA

2.1 Scelta: Hiding o Unveiling

Durante la fase di ideazione e concezione del progetto abbiamo deciso di inserire un interruttore per consentire all'utente la scelta di nascondere o rivelare il contenuto di *M*. In caso di selezione impossibilitata dal contenuto di *M* l'Arduino comunica l'errore (es. la sola presenza del file *Ibmp* non consente il processo di Hiding).

Si ipotizza come aggiunta futura, la possibilità di rendere i processi di Hiding e Unveiling automatici: se presenti *Itxt* e *Ibmp*, inizierà l'Hiding; se presente solo il file *Ibmp* avverrà invece l'Unveiling.

2.2 Micro SD

La micro SD dovrà avere una capienza pari o inferiore a 64 GB, poiché è la dimensione massima supportata dal lettore SD.

Per il corretto funzionamento dello steganografo, la scheda di memoria dovrà essere in formato **FAT32**, essendo uno standard di formattazione per i file system. Ai fini dei test è stata utilizzata una micro SD con 128 MB di memoria.

3 SCHEMA CIRCUITALE

Elenco componenti hardware:

- Arduino Uno R3 ATMEGA 328P
- Lettore SD - AZDelivery SPI Reader Micro SD-R
- SD micro (128 MB) - SD
- Powerbank (20 Ah) - PB
- LED RGB
- 2x Resistenza da 220 Ω
- 2x Resistenza da 330 Ω
- 1x Resistenza da 1000 Ω
- Cavi in quantità sufficiente a collegare le componenti
- Breadboard

3.1 Circuito logico

Il circuito si compone di 4 elementi montati su una breadboard 400 pin con 4 barre di alimentazione: un pulsante a pressione, un led RGB, uno switch a carrello e un modulo per la lettura di una micro SD. I cavi presenti nel circuito hanno colori specifici per la propria funzione. Il cavo ARANCIONE corrisponde al pin del led RGB rosso, il cavo BLU al pin del led RGB blu e il cavo VERDE al pin del led RGB verde. A seguire per i componenti attivi come ad esempio il pulsante e lo switch fanno utilizzo di cavi gialli. I cavi rossi indicano la VCC, mentre quelli neri la GND

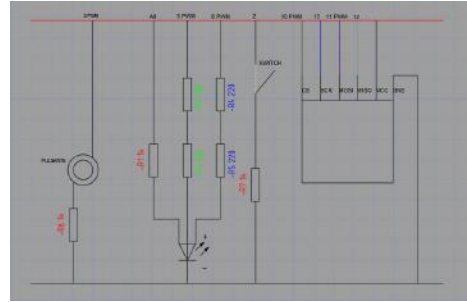
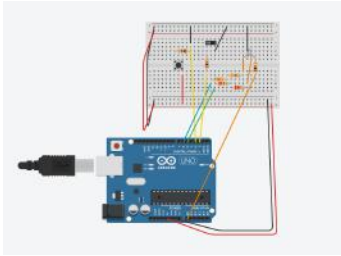


Fig. 5. Schema Logico
(9)

3.2 Case

Il case è il componente da noi usato per proteggere il circuito.

Per realizzarlo sono stati utilizzati principalmente pannelli in *Plexiglass*, con spessore di 2,5mm ciascuno e colla a caldo.

È dotato di uno sportellino a chiusura magnetica per poter sostituire/rimuovere il PB e/o l'arduino.

Le dimensioni sono all'incirca 9cm di larghezza, 18cm di profondità e 7,5cm di altezza, per un volume totale di 1215cm³.

Il case è suddiviso in due parti:

- **Layer 0 (bot):** Power bank.
- **Layer 1 (middle):** Circuito e Arduino.
- **Layer 2 (top):** I/O

Il Layer 2 è composto da:

- **LED RGB**
- **Pulsante**
- **Switch a carrello**
- **Lettore MicroSD**

3.3 LED RGB

Il Diodo è un componente elettronico passivo, non-lineare a 4 terminali (quadripolo), la cui funzione ideale è quella di permettere il flusso di corrente elettrica in un verso e di bloccarlo quasi totalmente nell'altro.

Il LED RGB è composto al suo interno da tre led, uno rosso, uno verde e uno blu, per questo dovrà essere collegato a tre differenti pin dell'Arduino e a gnd. Tra i pin dell'Arduino e i pin dei led sono state inserite delle resistenze per evitare che i componenti si danneggino.

- Il led rosso è collegato al pin A0, un pin analogico, con una resistenza da 1k ohm.
- Il led verde è collegato al pin 5 PWM con due resistenze da 330 ohm ciascuna per un totale di 660 ohm.
- Il led blu è collegato al pin 6 PWM con due resistenze da 220 ohm per un totale di 440 ohm.

Il led RGB è l'unico output da parte dello steganografo, poiché ad ogni colore è attribuito uno stato.

3.4 Pulsante

La funzione del pulsante è di resettare Arduino dopo che un'elaborazione è terminata oppure in caso di errore.

Collegato a 5V e al pin 3 tramite una resistenza pull-down, come illustrato nel circuito. In questo modo il pulsante è in uno stato "normalmente aperto", ossia premendolo si chiude il circuito permettendo il passaggio della corrente, mentre quando si trova in posizione di riposo il circuito è aperto e la corrente non circola.

3.5 Switch

Lo switch a carrello è un elemento attivo che serve per creare un'interruzione nel flusso di corrente elettrica che viene recepita da Arduino per farlo entrare in modalità unveiling viceversa sarà in modalità hiding.

3.6 SD-R

Il lettore di schede micro SD ha sei pin, due per l'alimentazione (5V e gnd) e quattro per la comunicazione con Arduino, che avviene tramite il protocollo SPI.

SPI (Serial Peripheral Interface) è un protocollo seriale sincrono per lo scambio di dati tra un microcontrollore (master) e una o più periferiche (slaves), comprende i seguenti pin:

- **MISO** (Master In Slave Out), per inviare dati dallo slave al master - pin 11;
- **MOSI** (Master Out Slave In), per inviare dati dal master allo slave - pin 12;
- **SCK** (Serial Clock), per sincronizzare la comunicazione tra i dispositivi attraverso il clock - pin 13;
- **CS** (Chip Select), utilizzato dal master per abilitare il dispositivo, utile nel caso ne sia connesso più di uno - pin 10;

4 APPROCCIO FALLIMENTARE

Come primo tentativo, si pensò di modificare i tre byte che determinano il colore del pixel (rosso (**R**), verde (**G**) e blu (**B**)), sostituendoli con i byte dei caratteri che compongono il testo del messaggio da nascondere nell'immagine. **Vantaggi:** Tempo di elaborazione molto breve, messaggi con dimensione maggiore:

$$I_{txt} < \frac{3}{4}(I_{bmp} - HP)$$

in quanto ogni pixel assumeva nei 3 byte RGB i 3 caratteri. Il risultato era un'immagine completamente diversa.

5 FUNZIONE HIDING

Il problema del primo approccio era l'evidente modifica dell'immagine. Quindi, come soluzione, si è scelto di modificare solo una parte del byte di colore, cioè la parte meno rilevante: il bit meno significativo.

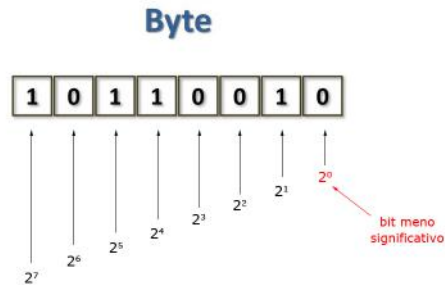


Fig. 6. Rappresentazione grafica di un byte

5.1 Algoritmo

La funzione “hiding” legge l’offset per analizzare l’header e la palette dell’immagine così da copiarli invariati sul file di output.

La reale codifica avviene copiando i bit del testo nei bit meno significativi dei byte che compongono i pixel (vedere sezione 1.1).

Per azzerare il bit meno significativo di ogni byte viene effettuata l’operazione di AND bit a bit tra il byte dell’immagine e la costante 0xFE.

Successivamente viene preso ogni singolo bit del carattere da nascondere, per farlo viene eseguito l’AND bit a bit con la variabile “x”, contenente il valore 0x01 (00000001), shiftata a sinistra di “n” volte, dove “n” indica il numero del bit del carattere preso in considerazione. Il valore ottenuto da questa operazione deve essere shiftato a destra di “n” posizioni, per portare il bit del testo nella posizione meno significativa.

Infine viene effettuata l’operazione di OR tra i due risultati ottenuti in precedenza. In questo modo si ottiene il byte dell’immagine con al suo interno, nel bit meno significativo, un bit di testo.

5.2 Codice

```

1 #include <SPI.h>
2 #include <SD.h>
3
4 void hiding(File image, File text, File out) {
5     image.seek(10);
6     int offset = 0;
7     image.read(&offset, sizeof(offset));
8
9     image.seek(0);
10    for(int j = 0; j < offset; ++j){
11        char x;
12        image.read(&x, sizeof(char));
13        out.write(x);
14    }
15
16    image.seek(offset);
17    while(image.available()){
18        setRgb(image.size(), image.position());

```

```

19     if(text.available()){
20         char text_byte = text.read();
21
22         char x = 0x01;
23         for(int i = 0; i < 8; ++i){
24             char img_byte;
25
26             image.read(&img_byte, sizeof(char));
27             char to_write;
28             asm(
29                 //lds carica in r24 dalla SRAM la variabile img_byte
30                 "lds r24, (img_byte)"
31                 //lds carica in r26 (registro puntatore di indirizzamento
32                 //indiretto) dalla SRAM la variabile text_byte
33                 "lds r26, (text_byte)"
34                 //AND tra il registro r24 e il byte 0xFE per impostare
35                 //il bit meno significativo a 0
36                 "and r24, 0xFE"
37                 //AND tra il registro r24 e il byte 0x01,
38                 //il risultato sar  inserito nel registro 24
39                 "and r24, 0x01"
40                 //lds carica in r25 la variabile i
41                 "lds r25, (i)"
42                 "loop:"
43                 //Spostamento del puntatore all'interno del registro r26,
44                 //effettua lo shift logico a destra
45                 "lsr r26"
46                 //Il comando dec decrementa la variabile
47                 //contenuta nel registro r25
48                 "dec r25"
49                 //brne termina il loop se r25 = 0, quindi
50                 //quando il flag z di dec torna al valore true
51                 "brne loop"
52                 //OR tra i registri r24 e r26 per inserire il byte nel immagine
53                 "or r24, r26"
54                 //lds carica in r28 dalla SRAM la variabile to_write
55                 "lds r28, (to_write)"
56                 //r28 assume il valore di r24
57                 "mov r28, r24"
58                 //Clobber List: specifichiamo i registri
59                 //da riservare per il programma
60                 ::: "r24", "r25", "r26", "r28"
61             );
62             x <<= 1;
63             out.write(to_write);
64
65             if ((i + 1) % 3 == 0){
66                 char x = image.read();
67                 out.write(x);
68             }
69         }
70         char y = image.read();

```

```

71         out.write(y);
72     }
73     else {
74         char x = image.read();
75         x &= 0xFE;
76         out.write(x);
77     }
78 }
79 }

```

6 FUNZIONE UNVEILING

6.1 Algoritmo

La funzione “unveiling” legge l’offset per posizionarsi dove iniziano i byte di colore ed iniziare il processo di costruzione del carattere.

Nella **riga 24** il byte dell’immagine viene messo in AND con la costante 0x01 (00000001) annullando tutti i bit tranne l’ultimo, che viene spostato poi a sinistra di “i” posizioni. Il byte così ottenuto viene messo in OR con il byte di testo da generare, aggiungendo un bit per volta.

Come per l’Hiding, ogni quarto byte viene saltato.

Una volta generato il carattere 0x00 (‘\0’, cioè il terminatore di stringa) la funzione arresta la lettura dell’immagine e termina.

6.2 Codice

```

1 void unveiling(File image, File text) {
2
3     image.seek(10);
4     int offset = 0;
5     image.read(&offset, sizeof(offset));
6
7     image.seek(offset);
8
9     while (image.available()) {
10         setRgb(image.size(), image.position());
11
12         for (i = 0; i < 8; ++i) {
13             // Lettura di un byte dell'immagine
14             image.read(&img_byte, sizeof(char));
15
16             // Le seguenti istruzioni eseguono lo shift di ogni bit
17             // del carattere all'interno dei byte dei pixel
18             asm(
19                 // caricamento del byte immagine
20                 "lds r24, (img_byte);"
21                 // caricamento del byte carattere
22                 "lds r26, (text_byte);"
23                 // AND dei registri r24 e r26
24                 "and r24, 0x01;"
25                 // Caricamento della variabile i nel registro r25
26                 "lds r25, (i);"
27                 "loop:"
28                 // Spostamento del puntatore all'interno di r24
29                 "lsl r24;"

```

```

30 //Decremento di 1 del valore
31 //memorizzato in r25
32 "dec r25;"
33 //Ritorno a riga 24 per continuare
34 //l'esecuzione del loop
35 "brne loop;"
36 //OR tra r26 e r24 per memorizzare il
37 //bit del txt
38 "or r26, r24;"
39 //Clobber List: specifichiamo i registri
40 //da riservare per il programma
41 ::: "r24", "r26", "r25"
42 );
43
44 if ((i + 1) % 3 == 0) {
45     image.read();
46 }
47 }
48
49 image.read();
50 text.write(text_byte);
51
52 if (text_byte == 0) {
53     break;
54 }
55 }
56 }

```

7 PROBLEMI

7.1 Formati Bitmap









Formato	Num. Colori	Pixel/Byte
1 bit	2	8/1
2 bit	4	4/1
4 bit	16	2/1
8 bit	256	1/1
16 bit	65536	1/2
24 bit	16,777,216	1/3
32 bit	4,294,967,296	1/4

Una limitazione dello steganografo è la compatibilità con i soli formati a 32 e 24 bit, questo è dovuto alla struttura dei formati bitmap.

Infatti nelle immagini a 32 e 24 bit ogni colore è descritto come miscela di tre colori primari: 1 Byte rosso, 1 Byte verde e 1 Byte blu, quindi cambiando il bit meno significativo di ogni componente è possibile alterare l'immagine in modo impercettibile per nascondere un testo al suo interno.

Nei casi per esempio a 8 o 4 bit cambiare un singolo bit significherà cambiare completamente colore, portando ad un'alterazione dell'immagine di output rispetto all'input.

Come illustrato dalla tabella seguente, tentando di effettuare l'Hiding su un'immagine in formato non supportato l'output risulta visibilmente alterato.

INPUT	OUTPUT	Formato
		4 bit
		8 bit
		24 bit
		32 bit

7.2 Formati Testo

La codifica consigliata per il testo è UTF-8, poiché è un formato comune che tutti i sistemi operativi sono in grado di interpretare, utilizzare altre codifiche potrebbe causare problemi durante la lettura del testo ricavato dalla funzione di Unveiling se il sistema non le supporta.

Ad esempio, utilizzando la codifica ANSI, esclusiva di Windows, il testo non viene interpretato correttamente dai sistemi macOS.

Tali inconvenienti non sono dovuti allo steganografo, che si limita a scomporre il testo in bit e a scriverli nell'immagine, ma piuttosto alle differenze tra i vari sistemi operativi e, soprattutto, tra i programmi di lettura del testo (es. Blocco Note, Notepad++, TextEdit per macOS), che potrebbero non supportare alcune codifiche.

7.3 Collo di bottiglia

La funzione di Hiding legge da due file e scrive su uno nuovo, creando così del traffico di dati per la lettura/scrittura *M*.

Essendo la scrittura lenta rispetto alla lettura, c'è la possibilità che si venga a creare un collo di bottiglia aumentando il tempo di esecuzione necessario per completare la funzione.

Questo problema, inoltre, potrebbe essere causato dalla funzione **setRGB** e dalla funzione **RGB_color**. Queste due funzioni vengono ripetute *N* volte quanto il numero di Byte da analizzare:

Per la funzione di Hiding:

$N = I_{bmp} - (HP)$ Per la funzione di Unveiling verranno ripetute:

$N = O_{txt}$

```
1 void setRgb(unsigned long file_size , unsigned long current_position) {
2     double x = ((double) current_position / (double) file_size) * 255.0 * 2.0;
3     if (x <= 255) {
4         RGB_color(0, x, 255);
5     } else {
6         RGB_color(0, 255, (255 * 2) - x);
7     }
8 }
```

8 GRAFICI

Per confrontare i due codici sorgente abbiamo raccolto il tempo di esecuzione delle funzioni di Hiding e Unveiling con file di dimensioni diverse.

Sono stati eseguiti tre test per ogni caso e infine è stata fatta una media tra i due tempi con valori più prossimi, in quanto si è visto che alcuni dati ottenuti risultavano incongruenti con il resto dei dati raccolti.

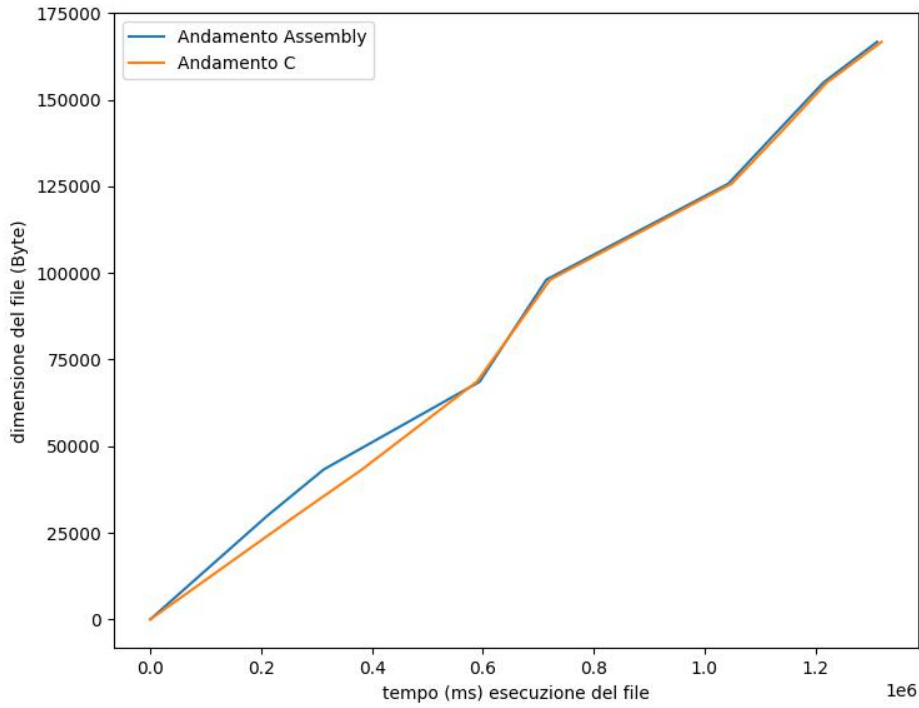
8.1 Hiding

File (Byte)	Assembly (ms)	C (ms)	Differenza (ms)
0	0	0	0
29.930	211.242	262.490	51.248
43.294	313.110	382.199	69.089
68.598	594.224	589.096	-5.128
98.074	714.763	721.741	6.978
125.790	1.043.108	1.049.203	6.095
140.754	1.130.812	1.138.523	7.711
154.890	1.213.920	1.220.143	6.223
166.710	1.311.339	1.319.517	8.178

Come si è visto, l'esecuzione di questa funzione prevede un ampio periodo di tempo in base alla dimensione dell'immagine in quanto, anche dopo aver nascosto l'intero testo, dovrà continuare a scrivere l'intera immagine. Ciò è dovuto principalmente all'hardware utilizzato.

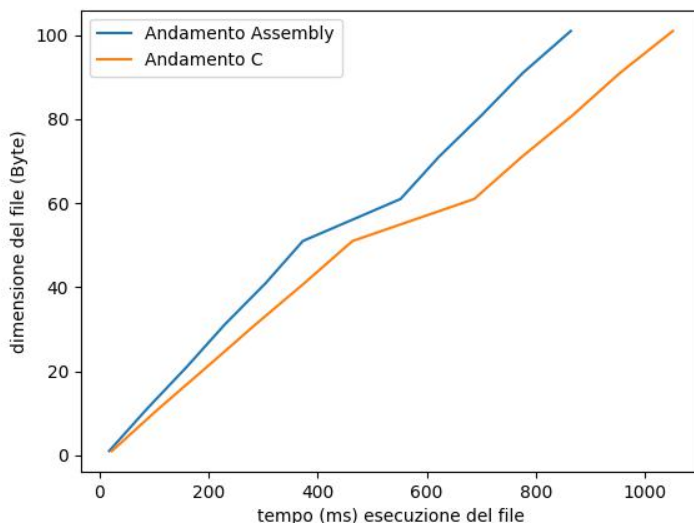
Il file con dimensione maggiore da noi testato, ovvero 166 KB, ha impiegato quasi 22 minuti.

In un file di queste dimensioni si possono nascondere circa 13.833 caratteri.



8.2 Unveiling

File (Byte)	Assembly (ms)	C (ms)	Differenza (ms)
1	18	23	5
11	87	108	21
21	160	197	37
31	229	285	56
41	305	376	71
51	373	464	91
61	552	687	135
71	662	774	112
81	701	868	167
91	776	954	178
101	864	1.051	187



9 CONCLUSIONI

Come volevasi dimostrare: il codice implementante le istruzioni Assembly, che lavora prevalentemente a livello MV2, ha un tempo di esecuzione minore rispetto il codice implementante il codice in solo linguaggio C, MV4.

Di fatto, entrambe le funzioni contenenti le istruzioni Assembly hanno ridotto il tempo di esecuzione del programma, come si può denotare dalle colonne "Differenza".

In conclusione, i tempi di Hiding sono troppo estesi. Ciò è dovuto prevalentemente all'hardware, utilizzando un **Raspberry Pi2 B** le tempistiche verrebbero sicuramente ottimizzate.

Ad esempio: avendo una maggiore quantità di RAM, si potrebbe anche rielaborare l'algoritmo utilizzando i thread per minimizzare i tempi di lettura e scrittura, una CPU con maggiore potenza di calcolo permetterebbe di lavorare direttamente sull'immagine di input modificando solo i byte necessari, quindi senza creare una nuova immagine.

10 IMPATTO COVID-19 SUL PROGETTO

Il progetto è nato dopo un anno di pandemia, per non correre rischi abbiamo innanzitutto seguito le direttive del governo unite al nostro buonsenso. Principalmente abbiamo lavorato da remoto con incontri saltuari per analizzare il punto della situazione.

Date le circostanze, abbiamo usufruito di **Github** (1), un potente strumento per il controllo delle versioni e di suddivisione degli incarichi per lo sviluppo software che, oltre a condividere i file del progetto, tiene traccia delle modifiche apportate durante il progresso del progetto.

In particolare, permette di creare delle **Issues**, da noi utilizzate per suddividere il lavoro.

Per quanto riguarda la parte di sviluppo del codice abbiamo utilizzato i seguenti IDE: **ArduinoIDE** (11), **CLion** (12), **Microchip Studio 7** (13).

Tra le alternative a nostra disposizione per la creazione del documento, abbiamo scelto **Overleaf** (14) \LaTeX , un editor online semplice da utilizzare con controllo di versione, condivisione istantanea e ampia possibilità di personalizzazione.

11 RIFERIMENTI

- (1) **Github Repository del Progetto DekketScri:**
<https://github.com/SpaceCowboyS01/DekketScri>
Accessed 2021-05-03
- (2) **Steganografia:**
<https://it.wikipedia.org/wiki/Steganografia>
Accessed 2021-05-18
- (3) **Informazioni sul bitmap:**
<https://www.fileformat.info/format/bmp/egff.htm>
https://it.wikipedia.org/wiki/Windows_bitmap
<http://www.brescianet.com/appunti/varie/tracciati/bmpformat.htm>
Accessed 2021-05-18
https://en.wikipedia.org/wiki/BMP_file_format
http://paulbourke.net/dataformats/bitmaps/index_it.html
Accessed 2021-06-07
<https://www.glgprograms.it/?p=prog/bitmap#introduzione>
Accessed 2021-06-16
- (4) **Fig.1: Esempio di un pixel formato bitmap 32 bit:**
http://paulbourke.net/dataformats/bitmaps/index_it.html
Accessed 2021-06-07
- (5) **Fig.3: Esempio di un pixel formato bitmap 24 bit**
http://paulbourke.net/dataformats/bitmaps/index_it.html
Accessed 2021-06-07
- (6) **Fig.4: Esempio modifica byte:**
<https://www.ictsecuritymagazine.com/wp-content/uploads/pixel.jpg>
Accessed 2021-05-03
- (7) **Arduino R3 ATMEGA 328P:**
Available: <https://www.microchip.com/wwwproducts/en/ATmega328P#datasheet-toggle>
Dispense Prof. Trojani Accessed 2021-05-17
- (8) **DataSheet Led RGB:**
<https://www.arduino.cc/documents/datasheets/LEDRGB-L-154A4SURK.pdf>
Accessed 2021-06-14
- (9) **Autodesk Tinkercad (Fig.5):**
<https://www.tinkercad.com/>
Accessed 2021-06-14
- (10) **Autodesk Inventor:**
<https://www.autodesk.it/products/inventor/overview?term=1-YEAR>
Accessed 2021-05-18
- (11) **Arduino IDE:**
<https://www.arduino.cc/en/software>
Accessed 2021-05-03
- (12) **C-Lion:**
<https://www.jetbrains.com/clion>
Accessed 2021-05-03
- (13) **MicrochipStudio7:**
<https://www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices>
Accessed 2021-05-03
- (14) **Overleaf:**
<https://it.overleaf.com>
Accessed 2021-05-18
- (15) Tutte le immagini non presenti in questo elenco sono state create dai membri del DekketScri-Team.