

COPIA DI UNA CHIAVE AUTO (PEUGEOT 207)

Ver. 1.0 – By Ptr

Indice

1 Premessa.....	3
2 Elenco chiavi HITAG2.....	4
3 Chiavi Peugeot/Citroen.....	6
4 Attacchi al protocollo HITAG2.....	10
4.1 Algoritmo di attacco.....	12
5 Reader/Writer per tag PCF.....	13
5.1 Collegamento della board RFID con Arduino.....	16
6 AESHitager software.....	18
7 Sniffer.....	20
7.1 Software.....	28
7.2 Compilazione e scrittura sull'ATMEGA.....	35
7.3 Debug del software.....	36
8 Sniffing della comunicazione.....	38
8.1 Utilizzo di un modulo Bluetooth.....	39
8.2 Estrazione dei codici.....	42
9 Copia fisica della chiave.....	44
9.1 VVDI Super Chip.....	46
10 Appendice A – Compilare Crack5 su Windows.....	51

1 Premessa

Recentemente ho dovuto fare una copia della chiave della mia vecchia auto (una Peugeot 207 del 2009) da un ferramenta e sono rimasto incuriosito dal processo di copia. La chiave è basata su un tag PCF della Philips e utilizza il protocollo HITAG2: non basta quindi fare una copia "fisica" della chiave ma è necessario fare anche una copia "elettronica" del contenuto per ottenere una chiave funzionante.

Il ferramenta ha applicato un aggeggio di plastica alla chiave originale, l'ha inserita nel blocchetto di accensione e ha avviato e poi spento la macchina per due/tre volte; ha poi portato la chiave dentro il negozio, ha tolto l'aggeggio di plastica e l'ha inserito nello strumento per la duplicazione. Ha poi proceduto con la fresatura della chiave "verGINE" (le chiavi delle citroen/peugeot hanno un solco su entrambi i lati della chiave) e alla fine l'ha inserita nello strumento per la duplicazione per l'inserimento dei codici (ottenuti, immagino, da quell'aggeggio di plastica utilizzato precedentemente). Tutto per la "modica" cifra di 70 euro (questo solo per una chiave "semplice" senza il radiocomando, con il radiocomando la cifra avrebbe raggiunto anche i 150 euro).

Mi sono chiesto quindi se fosse possibile fare una copia "in casa" senza particolari strumenti iper-costosi ma solo con strumenti autocostruiti o acquistabili a poco prezzo da Aliexpress/Ebay. La risposta è sì e questo documento vi mostrerà tutti i passaggi per fare una copia della vostra chiave senza spendere troppo (la spesa vi verrà sicuramente ripagata se offrite questo servizio a parenti e amici).

Ribadisco una cosa: quello che si sta facendo non è nulla di illegale ed è un servizio che vi potrebbe fare qualsiasi ferramenta con la strumentazione giusta. Per fare una copia di una chiave dovete avere accesso all'automobile: questo documento non vi insegna come rubare un auto.

Nota: nel documento sono presenti dei link a prodotti in vendita su Aliexpress/Ebay/Amazon. Non sono in alcun modo affiliato ai negozi che vendono quei prodotti, sono link lasciati come riferimento delle cose che ho acquistato io.

2 Elenco chiavi HITAG2

In seguito sono elencate alcune (non tutte) automobili che utilizzano le chiavi con il protocollo HITAG2:

Make	Models
Acura	CSX, MDX, RDX, TL, TSX
Alfa Romeo	156, 159, 166, Brera, Giulietta, Mito, Spider
Audi	A8
Bentley	Continental
BMW	Serie 1, 5, 6, 7, all bikes
Buick	Enclave, Lucerne
Cadillac	BLS, DTS, Escalade, SRX, STS, XLR
Chevrolet	Avanlache, Caprice, Captiva, Cobalt, Equinox, Express, HHR Impala, Malibu, Montecarlo, Silverado, Suburban, Tahoe Trailblazer, Uplander
Chrysler	300C, Aspen, Grand Voyager, Pacifica, Pt Cruiser, Sebring Town Country, Voyager
Citroen	Berlingo , C-Crosser, C2, C3 , C4 , C4 Picasso, C5 , C6, C8 Nemo, Saxo, Xsara, Xsara Picasso
Dacia	Duster, Logan , Sandero
Daewoo	Captiva, Windstorm
Dodge	Avenger, Caliber, Caravan, Charger, Dakota, Durango Grand Caravan, Journey, Magnum, Nitro, Ram
Fiat	500, Bravo, Croma, Daily, Doblo, Fiorino, Grande Punto Panda, Phedra, Ulysse, Scudo
GMC	Acadia, Denali, Envoy, Savana, Sierra, Terrain, Volt, Yukon
Honda	Accord, Civic , CR-V, Element, Fit, Insight, Stream, Jazz, Odyssey, Pilot, Ridgeline, most bikes
Hummer	H2, H3
Hyundai	130, Accent, Atos Prime, Coupe, Elantra, Excel, Getz Grandeur, I30 , Matrix, Santa Fe, Sonata, Terracan, Tiburon Tucson, Tuscani
Isuzu	D-Max
Iveco	35C11, Eurostar, New Daily, S-2000
Jeep	Commander, Compass, Grand Cherokee, Liberty, Patriot Wrangler
Kia	Carens, Carnival, Ceed, Cerato, Magentis, Mentor, Optima Picanto, Rio, Sephia, Sorento, Spectra, Sportage
Lancia	Delta, Musa, Phedra
Mini	Cooper
Mitsubishi	380, Colt, Eclipse, Endeavor, Galant, Grandis, L200 Lancer, Magna, Outlander, Outlander, Pajero, Raider
Nissan	Almera, Juke , Micra , Pathfinder, Primera, Qashqai, Interstar Note, Xterra
Opel	Agila, Antara, Astra, Corsa, Movano, Signum, Vectra Vivaro, Zafira
Peugeot	106 , 206 , 207 , 307 , 406, 407, 607, 807, 1007, 3008, 5008 Beeper, Partner, Boxer , RCZ
Pontiac	G5, G6, Pursuit, Solstice, Torrent
Porsche	Cayenne
Renault	Clio , Duster, Kangoo , Laguna II , Logan, Master Megane , Modus, Sandero, Trafic , Twingo
Saturn	Aura, Outlook, Sky, Vue
Suzuki	Alto, Grand Vitara, Splash, Swift, Vitara, XL-7
Volkswagen	Touareg, Phaeton

La tabella è stata estratta da questo paper:

http://www.proxmark.org/files/Documents/125%20kHz%20-%20Hitag/Gone_in_360_Seconds_Hijacking_with_Hitag2-USENIX_2012.pdf

L'elenco e' del 2012 per cui è riferito ai modelli di quell'anno. Non conosco nel dettaglio il tipo di chiavi che vengono utilizzate adesso: io ho fatto una copia per la mia Peugeot 207 del 2009, non sono un autoriparatore per cui non conosco tutti i modelli di chiave in circolazione adesso. Questa guida quindi si riferisce al caso specifico di una particolare chiave, ma in teoria può essere estesa a tutte le chiavi che utilizzano questo protocollo.

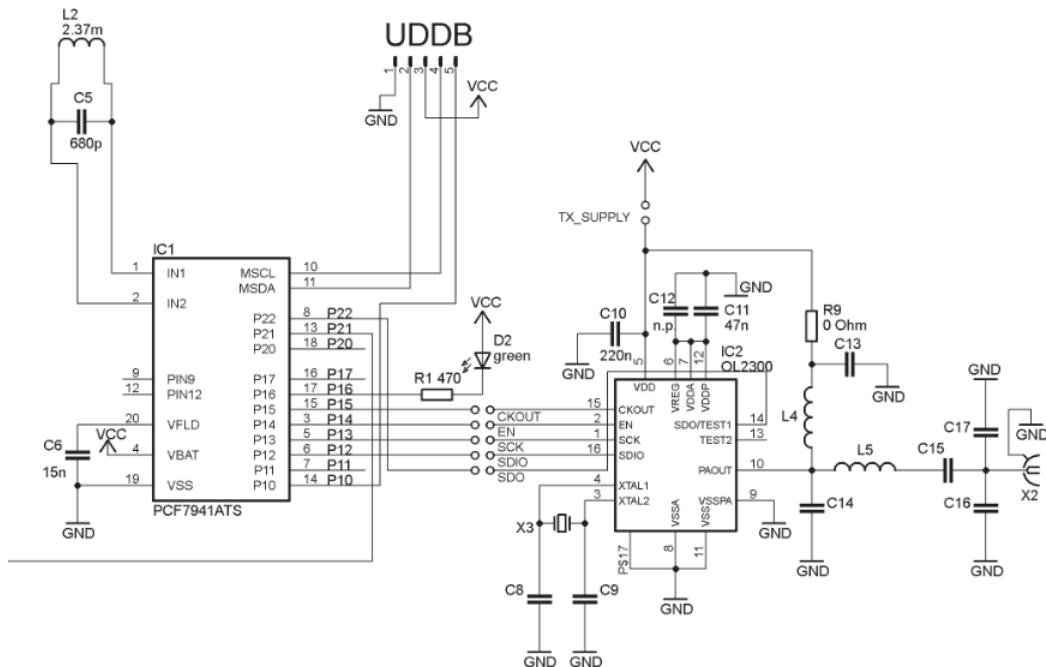
Le chiavi in questione utilizzano un protocollo di comunicazione che e' identico a quello dei tag PCF7936/7, con una chiave di cifratura a 48bit. NOTA: ci sono chiavi in circolazione che utilizzano i PCF7938/PCF7939 che utilizzano una chiave di cifratura non a 48bit e l'algoritmo di cifratura AES. Queste chiavi non possono essere duplicate con il metodo descritto in questo documento.

Qui trovate un elenco di tutti i tag philips con il relativo protocollo:

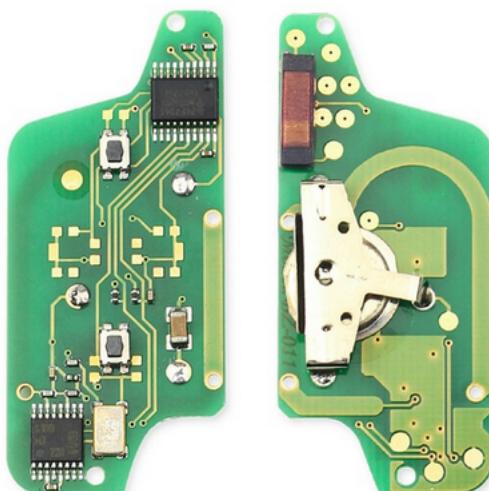
http://www.scorpio-lk.com/downloads/Tango/HITAG_Classification.pdf

3 Chiavi Peugeot/Citroen

Le chiavi auto delle vecchie Peugeot/Citroen che utilizzano il protocollo HITAG2 sono generalmente basate sul chip PCF7941 (datasheet <https://media.digikey.com/pdf/Data%20Sheets/NXP%20PDFs/PCF7x41ATJ.pdf>) che, assieme ad un chip per la trasmissione in radiofrequenza forma una chiave "completa":



Schema di esempio dal datasheet <https://www.mouser.it/datasheet/2/302/UM10473-2489331.pdf>

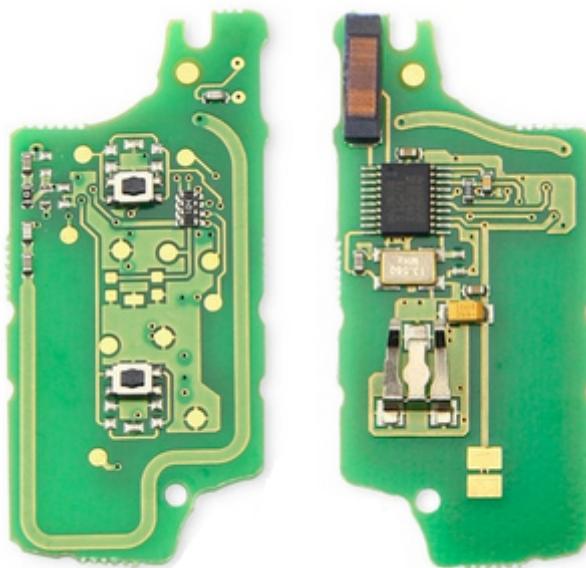


Esempio di una chiave basata su PCF7941 (nell'immagine di sinistra si trova in alto a destra).

La chiave è identificata dal numero 0523

Il PCF7941 si occupa direttamente della comunicazione a 125KHz mentre il chip esterno serve per la gestione del radiocomando (per l'apertura/chiusura dell'auto senza usare la chiave).

Esistono anche delle chiavi basate sul chip PCF7961 che non necessitano di un chip esterno per la gestione del radiocomando:

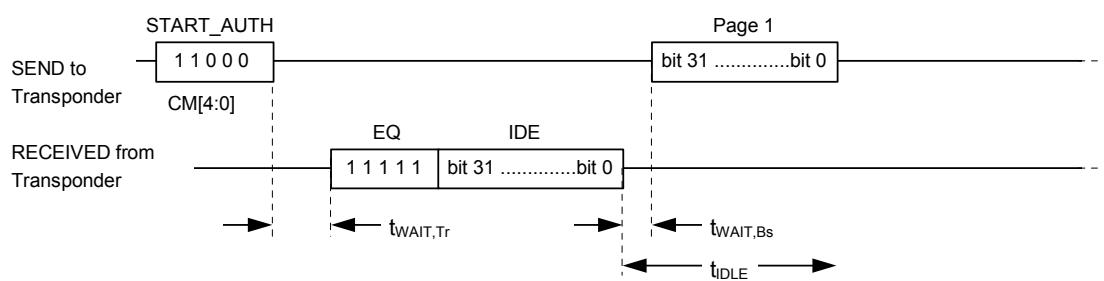


Esempio di una chiave basata su PCF7961

Per capire il protocollo di comunicazione fra la chiave e il lettore possiamo osservare il datasheet del tag PCF7936 (<http://www.proxmark.org/files/Documents/125%20kHz%20-%20Hitag/pcf7936-full%20DS.pdf>).

Questo tag ha due modalità di comunicazione, una "in chiaro" che si chiama **Password Mode** ed è il tipo di comunicazione classica di un tag PCF: per accedere al contenuto della memoria bisogna prima inviare (in chiaro) una password di 32 bit. Ottenere l'accesso per questa modalità è relativamente facile, basta costruire un dispositivo per "sniffare" la comunicazione del lettore e ottenere la password a 32 bit che viene trasmessa in chiaro; a questo punto, conoscendo la password, si può tranquillamente leggere e scrivere tutta la memoria.

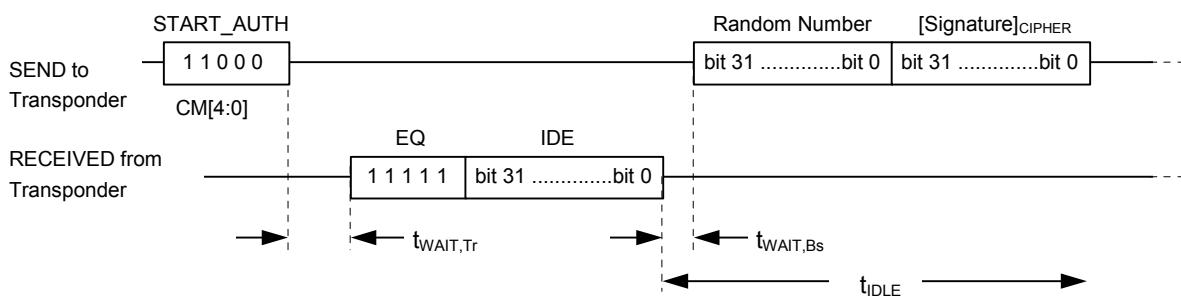
Qui potete vedere il dettaglio del comando AUTH (authenticate), il comando che il lettore invia per ottenere l'accesso alla memoria:



Il codice identificativo del comando (11000) viene inviato dal lettore, il tag risponde con il suo codice identificativo a 32 bit immagazzinato nella Page 0 e infine il lettore invia la password a 32bit (il contenuto della Page 1, come mostrato in figura) per ottenere l'autenticazione.

Peccato che per le chiavi auto non è così facile. L'altra modalità di comunicazione del tag si chiama Cipher Mode e, come si può intuire dal nome, non è "in chiaro".

In questa figura è mostrato il dettaglio del comando AUTH nella modalità **Cipher Mode**:



Il codice identificativo del comando (11000) viene inviato dal lettore, il tag risponde con il suo codice identificativo a 32 bit immagazzinato nella Page 0 proprio come nel caso precedente. Ora il lettore non invia una password a 32bit ma invia un *Random Number* a 32bit più una *Signature* a 32bit che viene calcolata usando l'algoritmo HITAG2 e la *Secret Key* del tag (che il lettore deve necessariamente conoscere).

Per ottenere l'accesso alla memoria non si può fare come nella modalità **Password Mode**: anche se sniffiamo il *Random Number* e la *Signature* non possiamo farci nulla. Dopo il comando AUTH tutta la comunicazione è criptata.

4 Attacchi al protocollo HITAG2

Non ho le competenze necessarie per descrivere i vari attacchi che sono stati utilizzati per "craccare" il protocollo HITAG2. Vi basti sapere che il protocollo è stato completamente "reversato" e il codice è disponibile online (vedi link <http://web.archive.org/web/20120127012528/http://cryptolib.com/ciphers/hitag2/hitag2.c>) e sono stati realizzati diversi tipi di attacchi per ottenere la Secret Key del tag.

Qui trovate alcuni link per i più interessati che spiegano il reverse dell'algoritmo e alcuni attacchi:

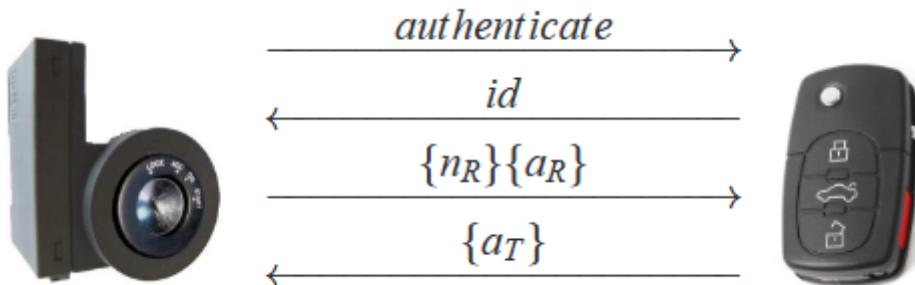
<http://www.proxmark.org/files/Documents/125%20kHz%20-%20Hitag/>

[Breaking_Hitag_2_Revisited.pdf](#)

<http://www.proxmark.org/files/Documents/125%20kHz%20-%20Hitag/>

[Gone_in_360_Seconds_Hijacking_with_Hitag2-USENIX_2012.pdf](#)

Copio qui l'immagine dell'autenticazione in modalità Cipher Mode mostrata in uno dei link riportati sopra:



Il lettore (in questo caso il blocchetto di accensione dell'automobile) invia il comando *authenticate* (11000) al tag, questo risponde con il suo codice identificativo a 32 bit (in chiaro, indicato come *id*). In seguito il lettore invia il *Random Number* (indicato come *{nr}*) e la *Signature* (indicata come *{ar}*). Ho ripetuto qui questa sequenza perché uno degli attacchi utilizzati prevede l'utilizzo del codice identificativo del tag (*id*) più due coppie di *{nr}{ar}* per ottenere la Secret Key. In questo modo si può sniffare la comunicazione fra blocchetto di accensione dell'automobile e la chiave, come si faceva nel caso semplice in modalità **Password Mode**, per ottenere tutto quello che ci serve. L'*id* lo possiamo ricavare usando un normale lettore di tag PCF (quando si invia il comando 11000 la risposta del tag che contiene il codice identificativo e' in chiaro), per le due coppie di *{nr}{ar}* dobbiamo necessariamente costruire uno sniffer.

Quindi, per ottenere la Secret Key ci servono tre cose:

- Una implementazione dell'algoritmo per "craccare" il protocollo HITAG2.
- Un lettore di tag PCF che supporta il protocollo HITAG2 (non basta un lettore "semplice").
- Uno sniffer per captare la comunicazione fra il blocchetto di accensione dell'automobile e la chiave.

4.1 Algoritmo di attacco

Sono stati implementati diversi tipi di attacchi, il loro codice lo potete trovare su github nella repository del Proxmark3 di Iceman:

<https://github.com/RfidResearchGroup/proxmark3/tree/master/tools/hitag2crack>

Gli attacchi che ci interessano sono l'**Attack 5** e l'**Attack 5opencl**. Quest'ultimo è una versione ottimizzata dell'**Attack 5** che utilizza la GPU della scheda grafica, dato che non ne ho una a disposizione faccio riferimento all'**Attack 5** (gli input sono gli stessi).

Da linux, è necessario fare il clone della repository in una cartella del vostro sistema:

```
git clone https://github.com/RfidResearchGroup/proxmark3.git
```

Entrare nella cartella "crack5":

```
cd proxmark3/tools/hitag2crack/crack5/
```

Eseguire le istruzioni per il build:

```
make clean  
make
```

Per l'utilizzo, riporto qui le istruzioni:

```
You'll need just two nR aR pairs. These are the encrypted nonces and challenge response values. They should be in hex.
```

```
./ht2crack5 <UID> <nR1> <aR1> <nR2> <aR2>
```

```
UID is the UID of the tag that you used to gather the nR aR values.
```

Mostreroò in seguito come utilizzare il comando.

È possibile anche compilare l'attacco su windows, le istruzioni le trovate nell'[Appendice A](#).

5 Reader/Writer per tag PCF

In commercio sono presenti diversi lettori per la programmazione delle chiavi auto che utilizzano il protocollo HITAG2 (ne elenco alcuni)

- Una versione del Gambit (il case è praticamente lo stesso) specifica per il protocollo HITAG2, venduta intorno agli 80/100 euro.



- Lo ZedBull mini venduto a circa 50 euro.



- L'IPROG pro venduto a circa 60/100 euro



Ce ne sono sicuramente altri che mi sono dimenticato/non conosco, ma come potete vedere hanno tutti comunque dei prezzi molto alti.

L'IPROG pro è un lettore molto particolare, ha tutta una serie di schede che si possono collegare per utilizzare diverse funzioni: una di queste schede serve proprio a leggere i tag a 125KHz/134KHz e può essere acquistata separatamente (al costo di circa 16 euro qui <https://it.aliexpress.com/item/1005003735929997.html>)



Su questa scheda è presente il chip PCF7991, il lettore di tag PCF che generalmente viene usato nel blocchetto di accensione dell'automobile. Sulla scheda i pin per la comunicazione con il PCF7991 vengono portati fuori sul connettore a 44pin.

Un utente del forum digital-kaos è riuscito ad interfacciare questa scheda con Arduino e ha realizzato un software per la lettura/scrittura dei tag PCF che utilizzano il protocollo HITAG2 (e molte altre chiavi).

Link alla pagina del forum: <https://www.digital-kaos.co.uk/forums/showthread.php/876570-Open-Source-Hitag-Programmer/>

Link al sito personale dell'utente: <https://kivijakola.fi/projektit/2021/01/27/hitag-open-source-tool/>

Link alla repository di github che contiene lo sketch di arduino e il software per la lettura/scrittura di tag: <https://github.com/kivijakola/hitager/>

Ricapitolando: acquistando una scheda su Aliexpress al costo di 16 euro e usando Arduino (va bene anche un Arduino Nano) si riesce a leggere e scrivere sui tag PCF. Ad oggi credo che questa sia la soluzione più economica che ci sia in giro per leggere/scrivere questo tipo di tag.

5.1 Collegamento della board RFID con Arduino

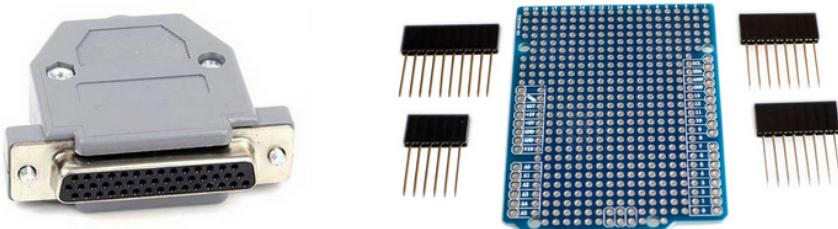
Una volta acquistata la board, dovete collegarla correttamente al vostro Arduino. Potete fare in diversi modi:

- Usare dei connettori Dupont, per intenderci, questi:

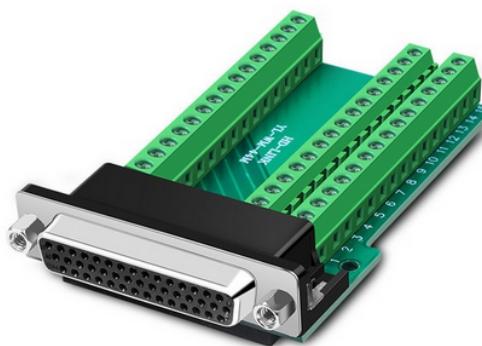


per collegare direttamente i pin di Arduino ai pin del connettore a 44pin.

- Acquistare un connettore femmina a 44pin (questo <https://it.aliexpress.com/item/1005003656001383.html>, selezionare l'opzione "Classic Black Female") e usare una prototype shield (questa <https://it.aliexpress.com/item/32846515603.html>) per saldare direttamente dei fili dal connettore femmina alla prototype shield.



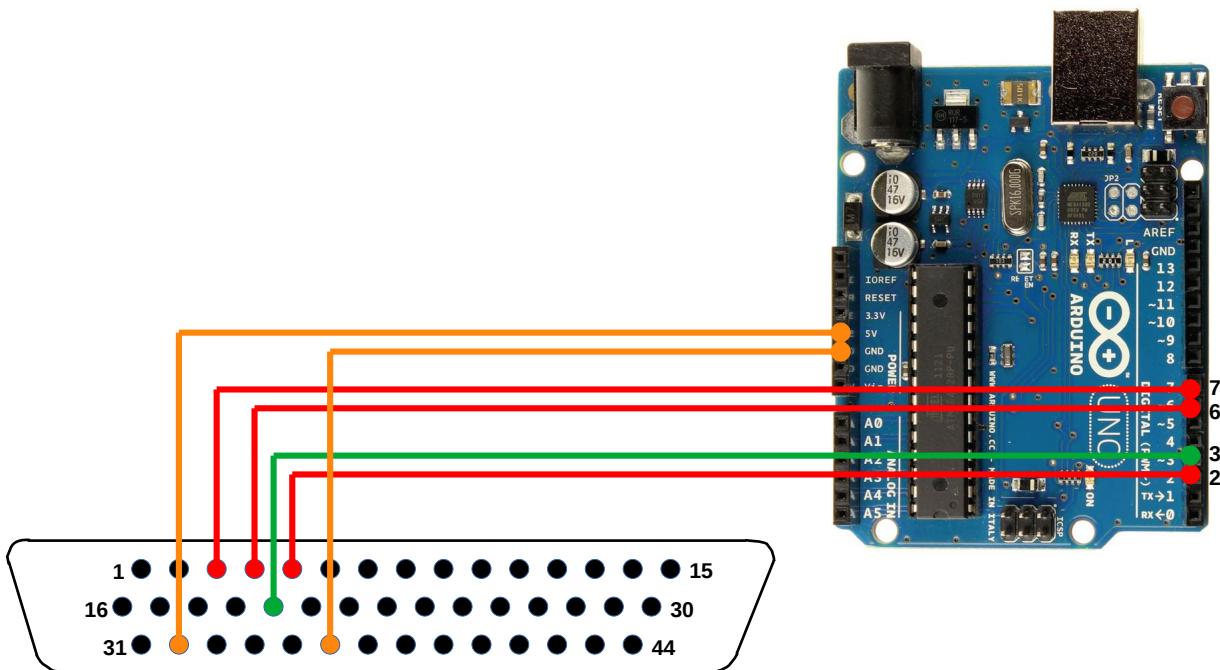
- Usare una breakout board (questa <https://it.aliexpress.com/item/1005001924810029.html>, selezionare "Female Nut") per facilitare la connessione ed avere i pin numerati direttamente in bella vista.



Le connessioni da fare fra board e Arduino sono le seguenti:

Arduino Uno/Nano	PIN Connettore scheda
D2	5
D7	3
D6	4
D3	20
GND	36
VCC	32

Io ho utilizzato i connettori Dupont, le connessioni da fare (quelle che ho elencato sopra) sono illustrate in figura.



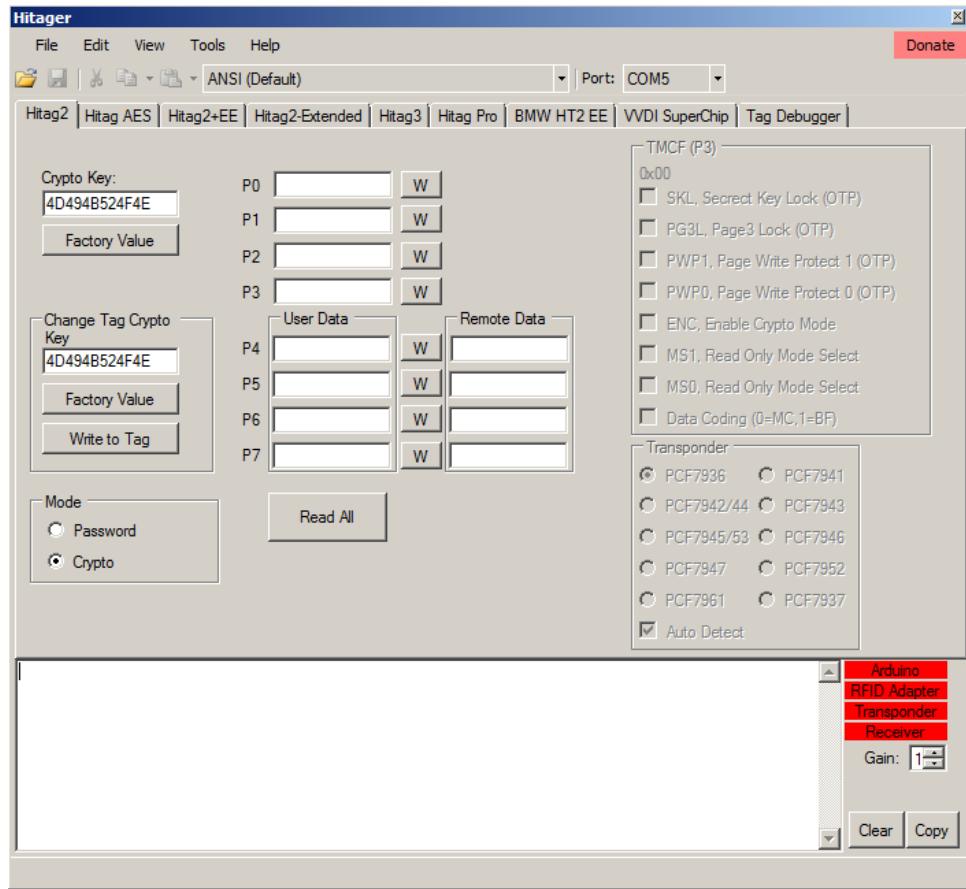
Fatti i collegamenti, è necessario scaricare il software e lo sketch di Arduino da qui:

<https://github.com/kivijakola/hitager/releases>

La pagina è da tenere d'occhio (assieme al thread su digital-kaos) perchè il software e lo sketch vengono aggiornati. Ora che sto scrivendo, l'ultima release è la v1.3.2: il software si trova nell'archivio **AESHitager_v132.7z** e lo sketch si trova dentro **hitaguino_v211.7z**. Vanno semplicemente estratti e nel caso dello sketch è sufficiente aprirlo con Arduino IDE e fare l'upload.

6 AESHitager software

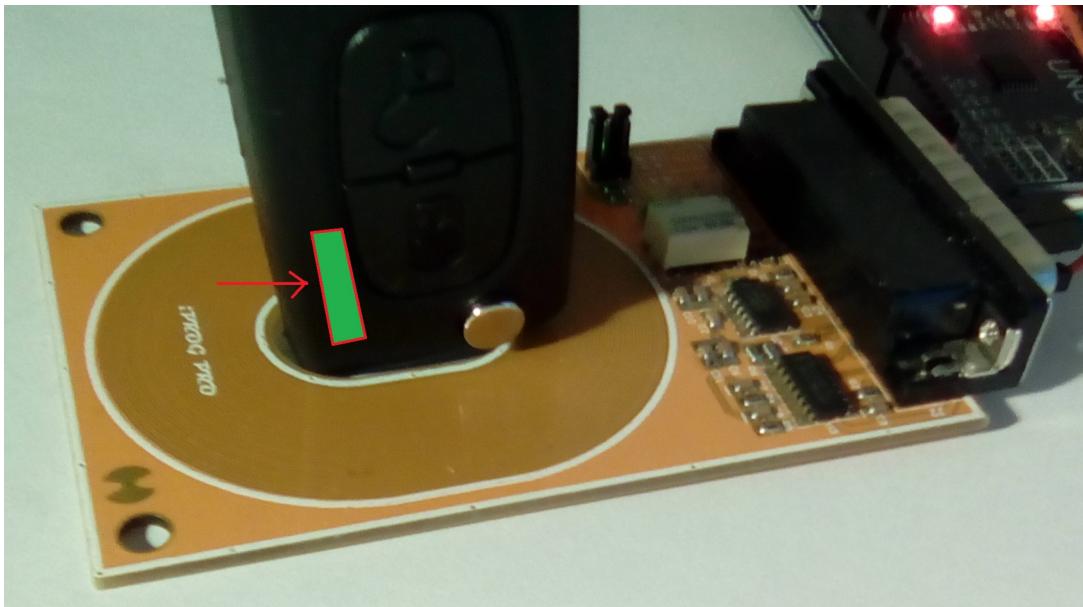
Una volta che l'Arduino è stato programmato, potete aprire il software AESHitager:



Una volta aperto, portate a 2 il valore di “Gain” che si trova in basso a destra. Nella casella di testo vi troverete dei messaggi come mostrato nella figura qua sotto; se tutto va bene e i collegamenti con la scheda sono corretti, la scritta “Arduino” diventerà verde:



A questo punto potete posizionare la chiave (o il tag) da leggere/scrivere sulla scheda. Nel caso di una chiave, questa deve essere posizionata in verticale al centro della scheda (dove si trova il buco) come illustrato in figura:



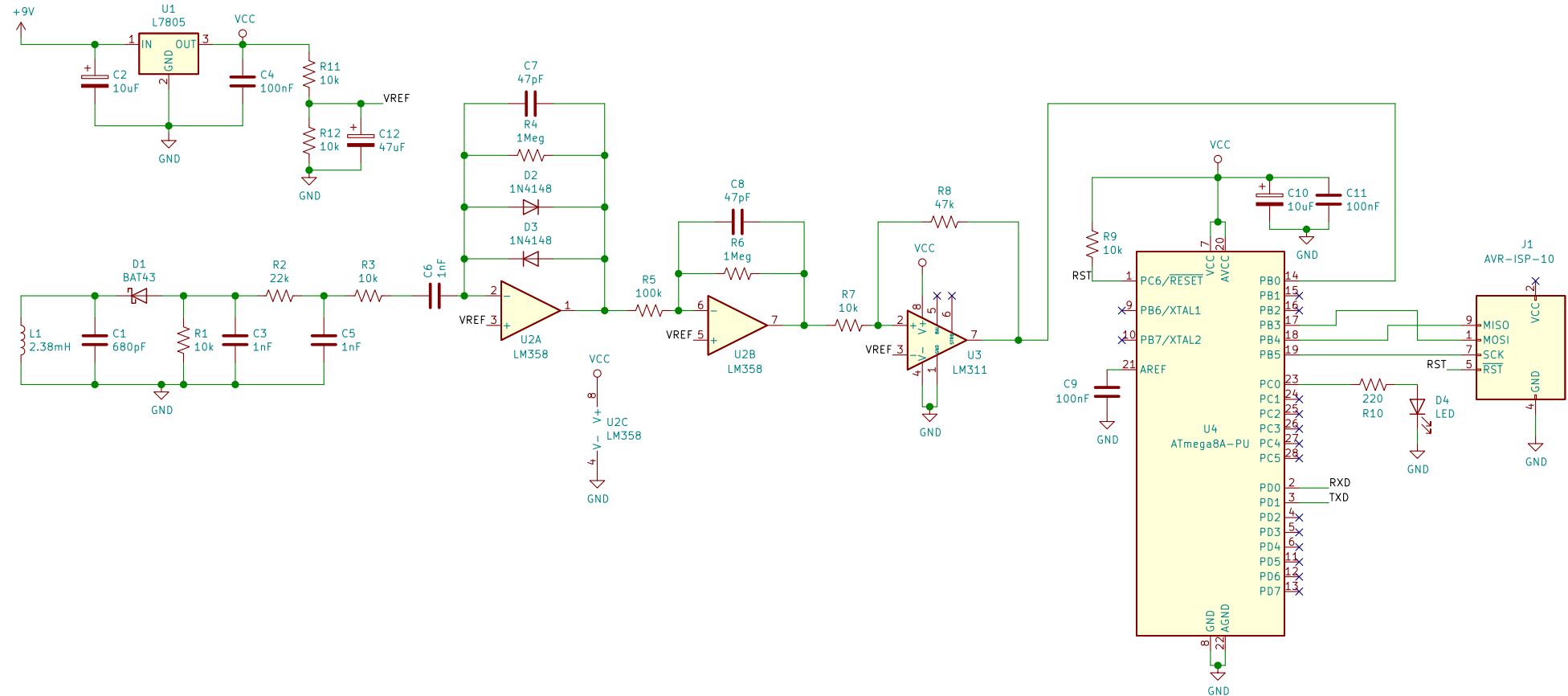
La freccia rossa indica la posizione dell'antenna all'interno della chiave

Nel caso di un tag, anche questo deve essere posizionato al centro della scheda e **il gain deve essere portato a 1 nel caso si abbiano problemi di lettura**:



Il resto del software è semplice e intuitivo da usare: vedremo poi quali passaggi seguire per leggere correttamente una chiave e modificarne il contenuto.

7 Sniffer



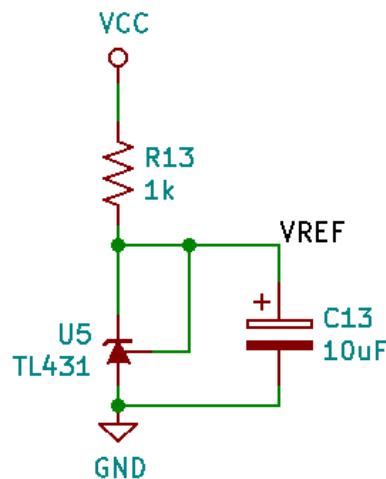
Schema dello sniffer

In seguito verranno descritte alcune parti del circuito (il suo schema è stato preso da questo documento <https://ww1.microchip.com/downloads/en/devicedoc/51115f.pdf> nella sezione “ASK READER SCHEMATIC”, con alcune modifiche).

- *Alimentazione*

Il circuito viene alimentato da una batteria a 9V in ingresso a un integrato 7805 per generare la tensione di alimentazione a 5V per l'OP-AMP, per il comparatore e per il microcontrollore.

Viene anche generato il riferimento a VCC/2 necessario per il funzionamento dell'OP-AMP e del comparatore. In alternativa, potete usare un integrato TL431 per generare il riferimento a 2,5V



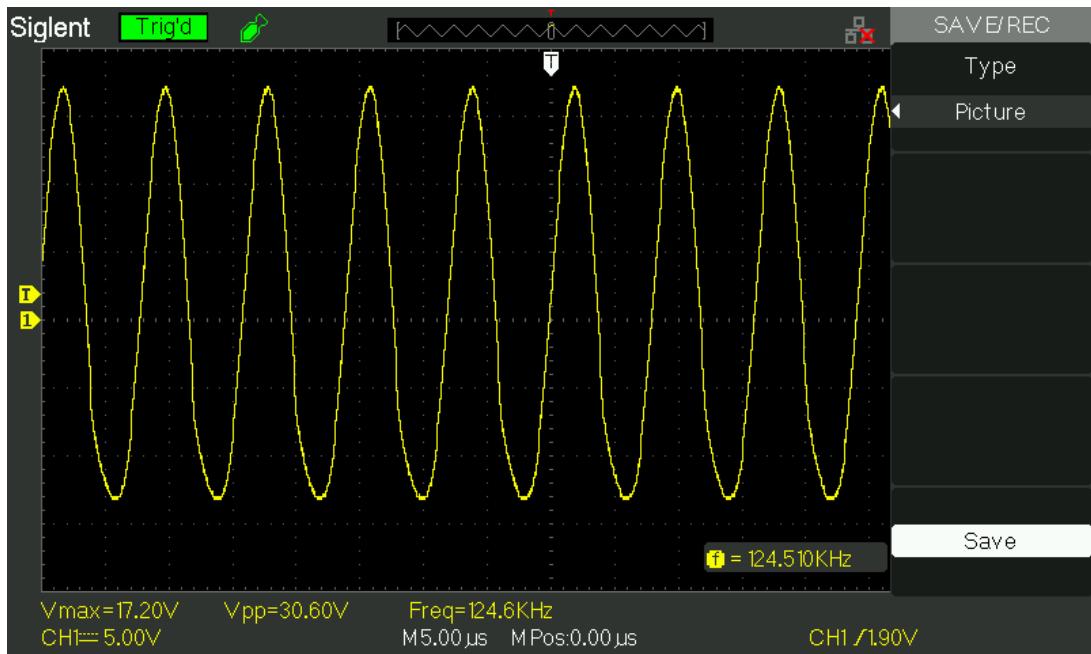
- *L1-C1 Circuito risonante*

Per il circuito risonante viene utilizzata un'antenna che viene utilizzata nelle chiavi auto e si trova “già pronta” in commercio (ad esempio qui <https://it.aliexpress.com/item/1005002940462306.html>, oppure qui <https://www.ebay.it/itm/254301759219>).



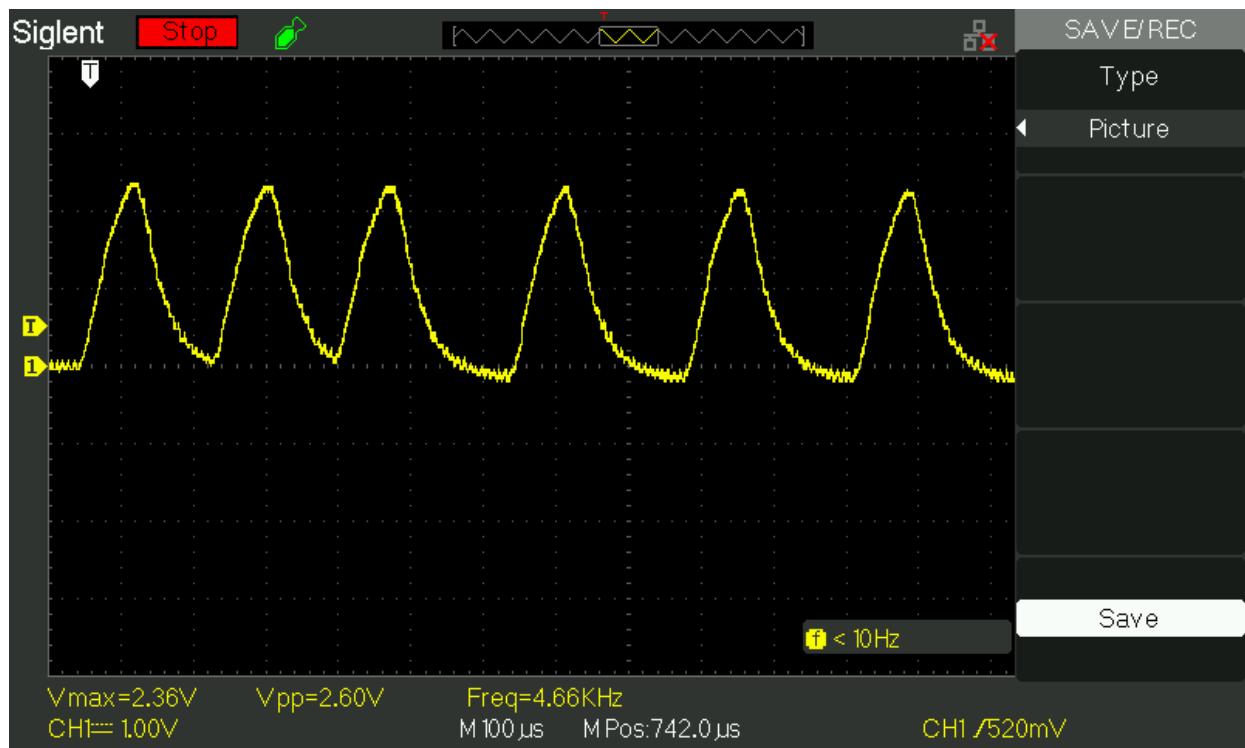
È un'antenna lunga 1cm che ha un'induttanza di 2,38mH : il condensatore scelto per avere la risonanza a 125KHz è 680pF.

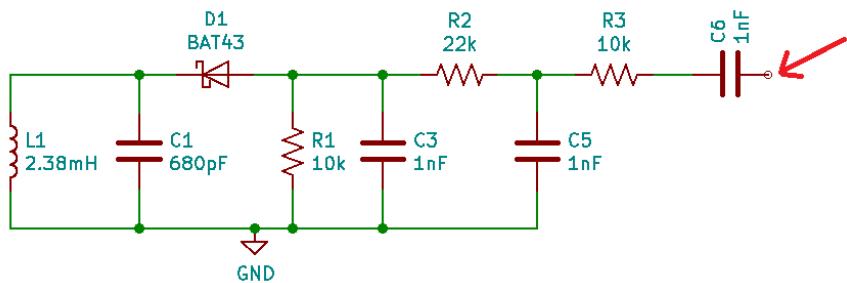
Questo è lo screenshot della tensione sull'antenna quando è appoggiata sulla scheda del nostro lettore:



- *D1-R1-C3-R2-C5-R3-C6 Rivelatore di inviluppo + filtro*

La variazione di tensione impressa dal tag sull'antenna viene rivelata e filtrata da questo circuito. Come riferimento riporto lo screenshot della tensione presa fra un terminale di C6 e la massa quando il circuito dell'OP-AMP non è collegato:

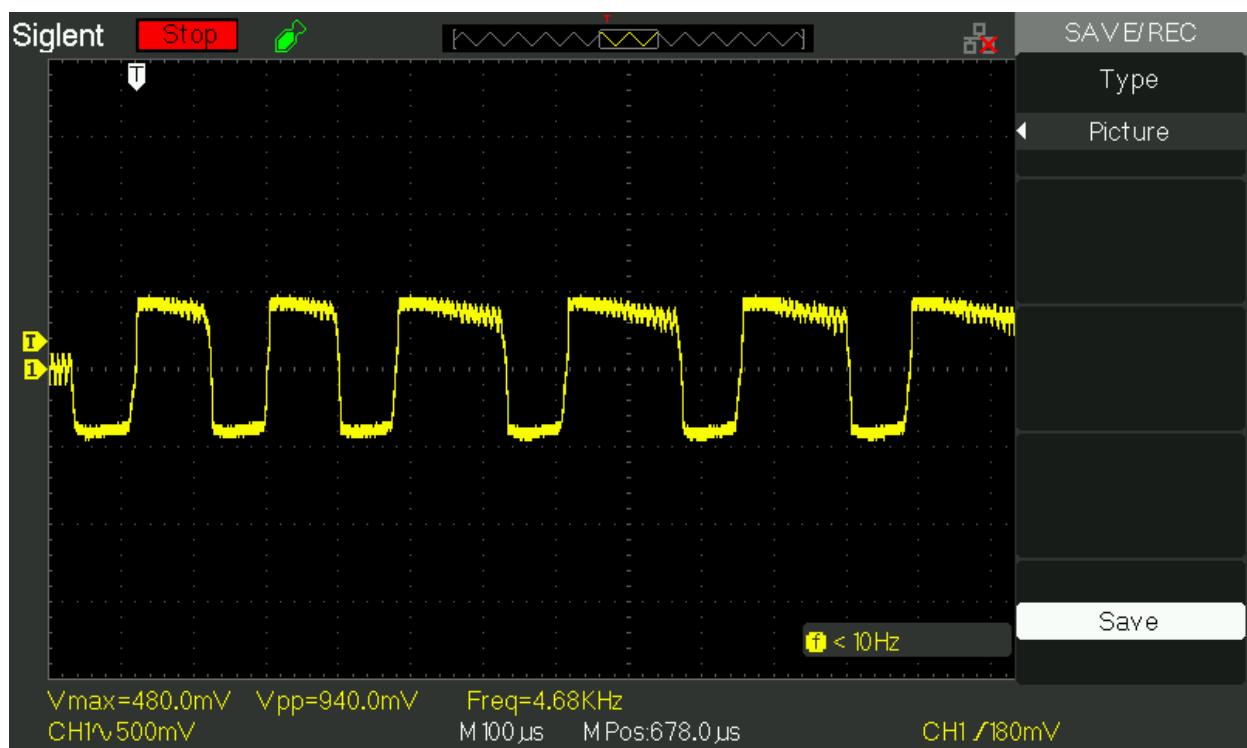




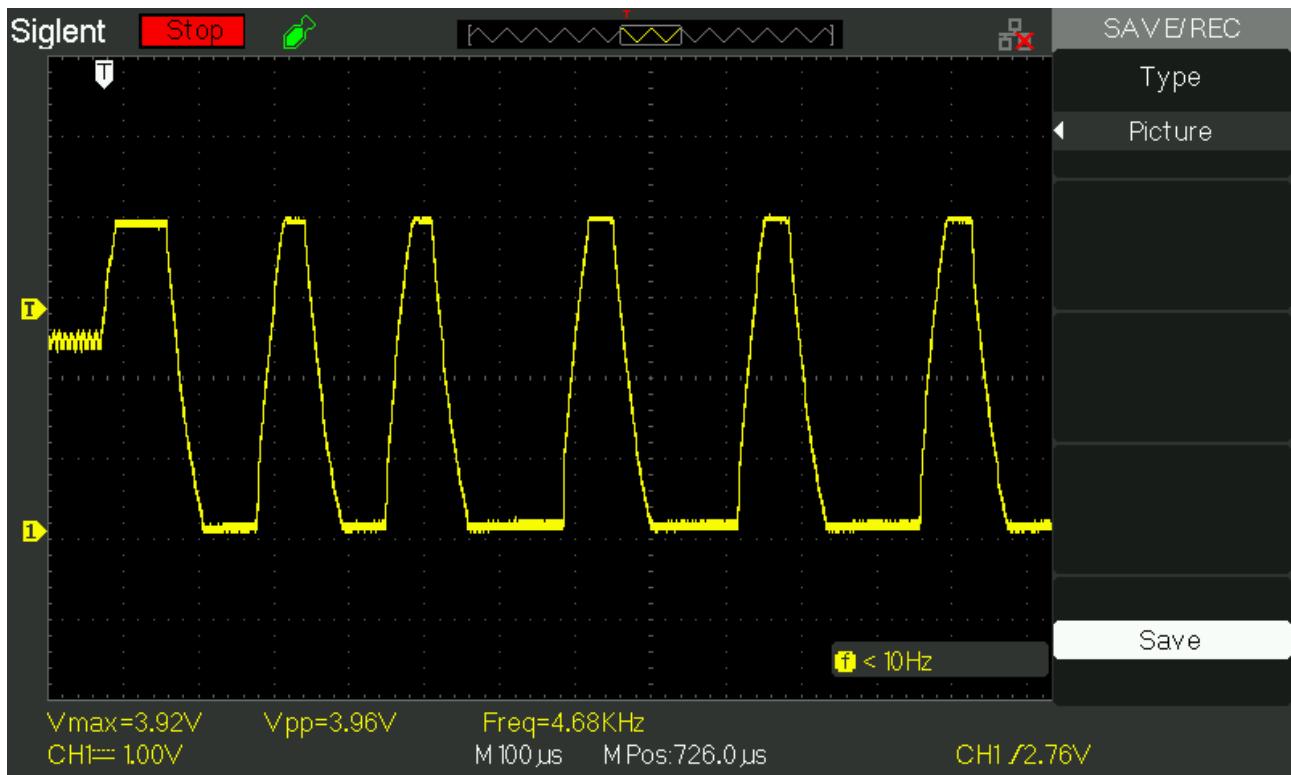
La misura è stata fatta usando questo circuito e posizionando la sonda fra la freccia rossa e la massa.

- ***U2 (LM358) - Amplificatore***

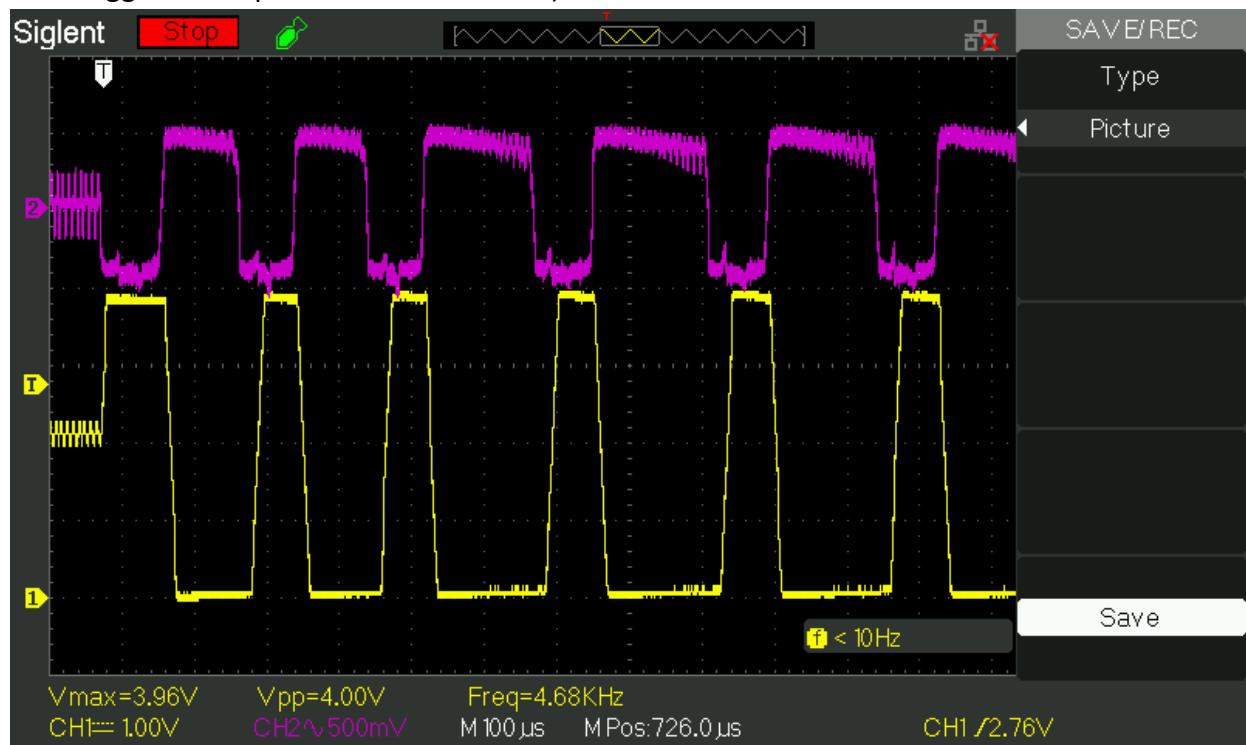
La tensione rivelata viene amplificata e filtrata da U2 (nell'LM358 ci sono due OP-AMP nello stesso package). Questa è la tensione di uscita di U2A (pin 1, accoppiamento in AC):



Questa è la tensione di uscita di U2B (pin 7):



È possibile aumentare il guadagno di U2B diminuendo la resistenza R5. Usando $22\text{k}\Omega$ invece di $100\text{k}\Omega$, si ottengono dei fronti più ripidi della tensione di uscita (ma il circuito diventa leggermente più sensibile ai disturbi):

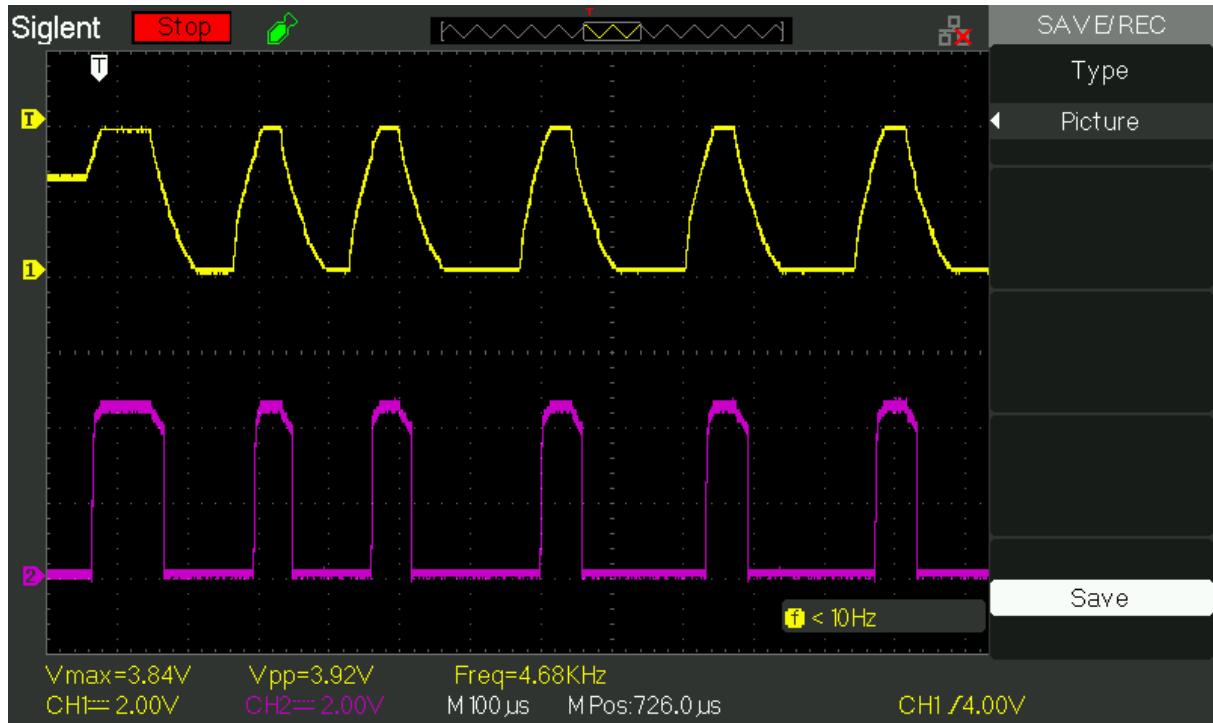


Ho lasciato la resistenza da $100\text{k}\Omega$ perché si ottengono comunque risultati ottimi. Potete sperimentare anche voi e decidere di conseguenza.

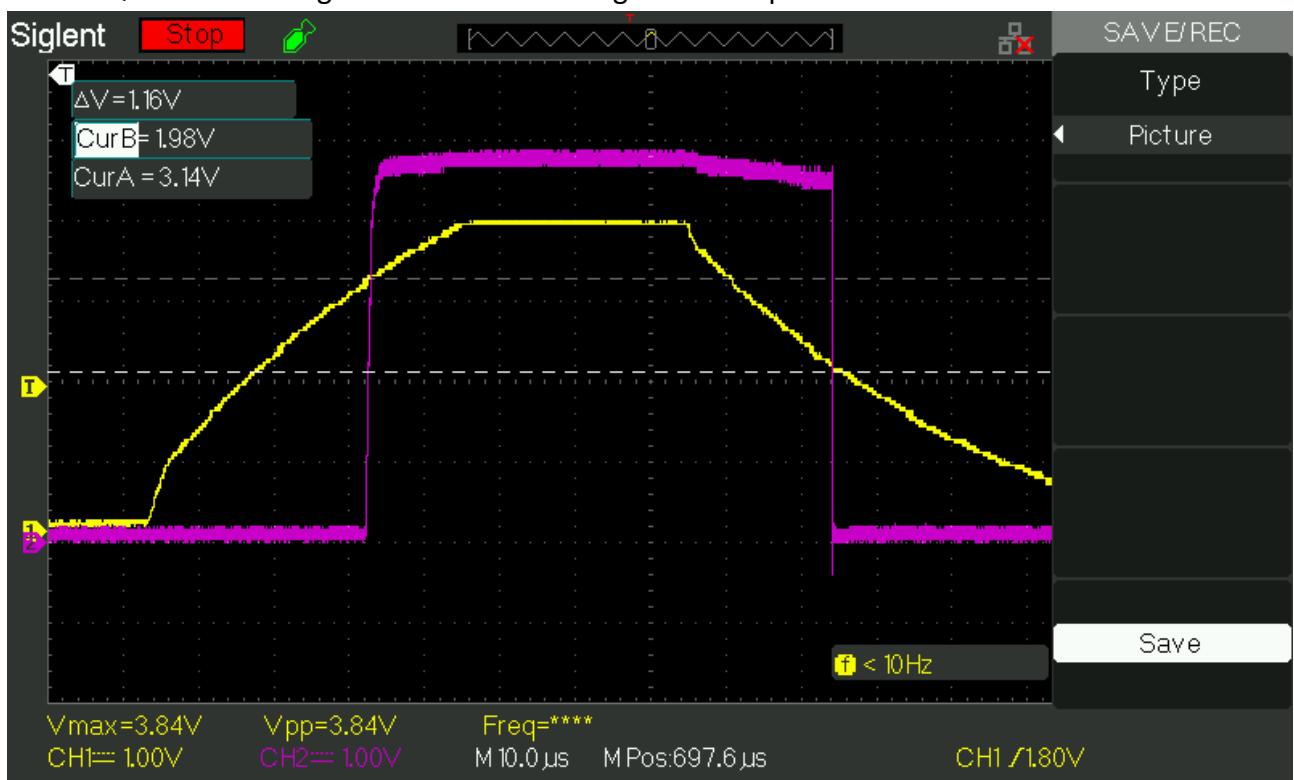
- *R7-R8-U3 Comparatore con isteresi*

Prima di fornire la tensione in ingresso al microcontrollore ho utilizzato un comparatore (LM311) per squadrare la tensione in uscita all'OP-AMP e per avere un minimo di isteresi per la reiezione dei disturbi.

La forma d'onda viola è la tensione in uscita al comparatore (pin U3), quella gialla è l'uscita di U2B (pin 7):



Questo è il dettaglio delle tensioni di soglia del comparatore con isteresi:



La tensione di soglia superiore è di circa 3V e la tensione di soglia inferiore è di circa 2V.
È possibile sperimentare con tensioni di soglia diverse e con una tensione di riferimento diversa (la tensione del pin 3 di U3) per avere una isteresi più “larga”.
Potete usare questo calcolatore online <https://www.allaboutcircuits.com/tools/hysteresis-comparator-calculator/> per le vostre sperimentazioni

Input

Positive Supply Voltage: V

Negative Supply Voltage: V

Calculate V_{th} and V_{tl} given $R (=R_2/R_1)$ and V_{ref}

Ratio of Resistors (R_1/R_2):

Reference Voltage (V_{ref}): V

High Threshold Voltage (V_{th}): V

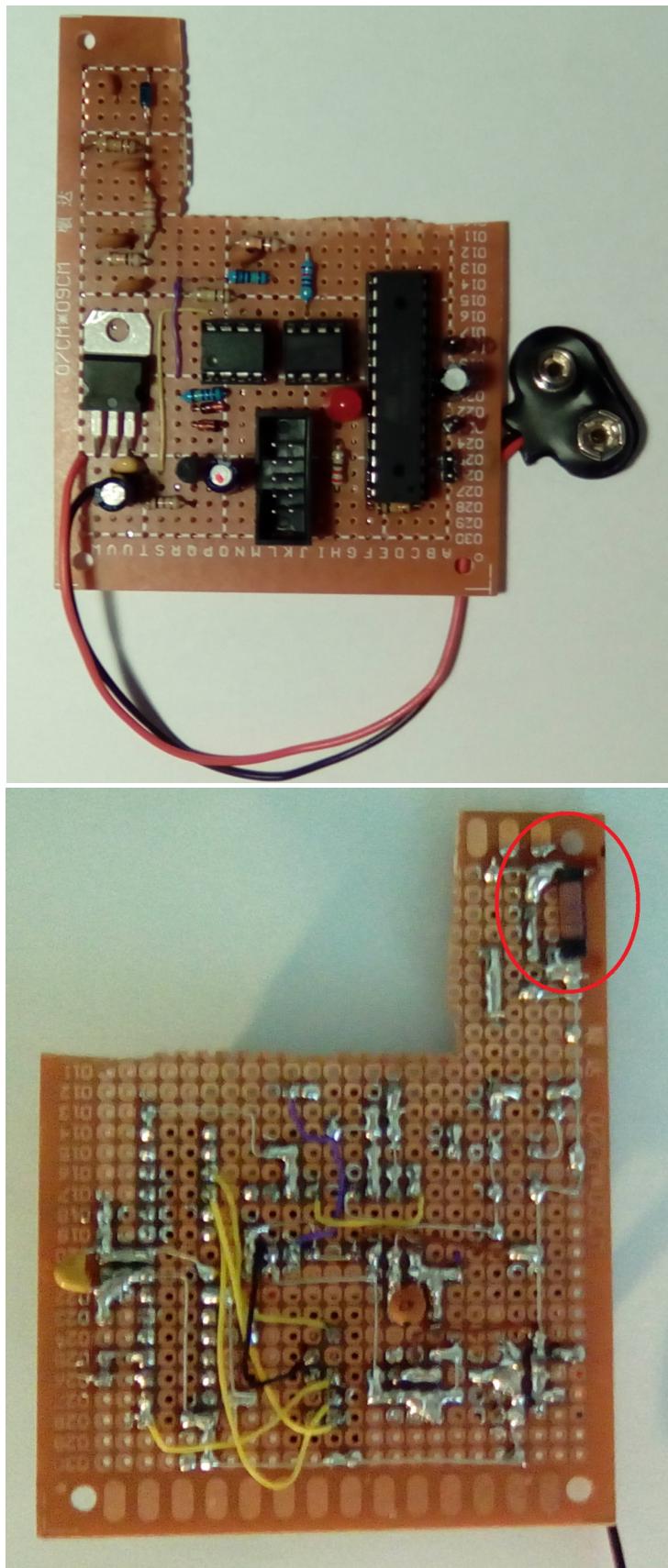
Low Threshold Voltage (V_{tl}): V

Hysteresis Voltage:

Calculate

Inserendo il valore di “Positive Supply Voltage”, “Negative supply voltage”, “Ratio of Resistors” (attenzione che nel sito c’è un errore, in realtà è R_2/R_1 in base allo schema che propongono) e “Reference Voltage”, cliccando su “Calculate” vi compariranno i valori di entrambe le soglie del vostro comparatore con isteresi (se vedete con i valori che ho utilizzato, i livelli delle soglie tornano con quanto si vede nel circuito).

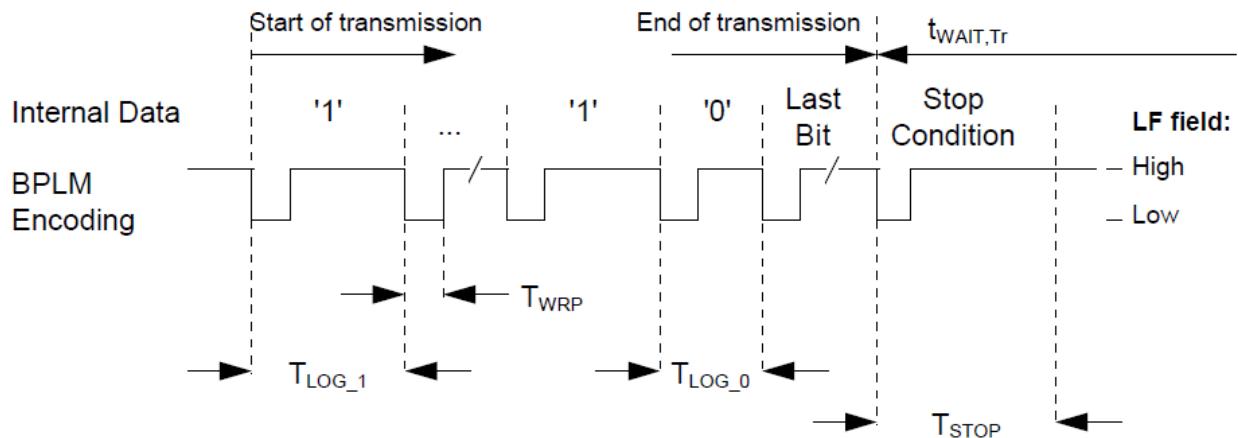
Queste sono le immagini del prototipo che ho realizzato su scheda millefori:



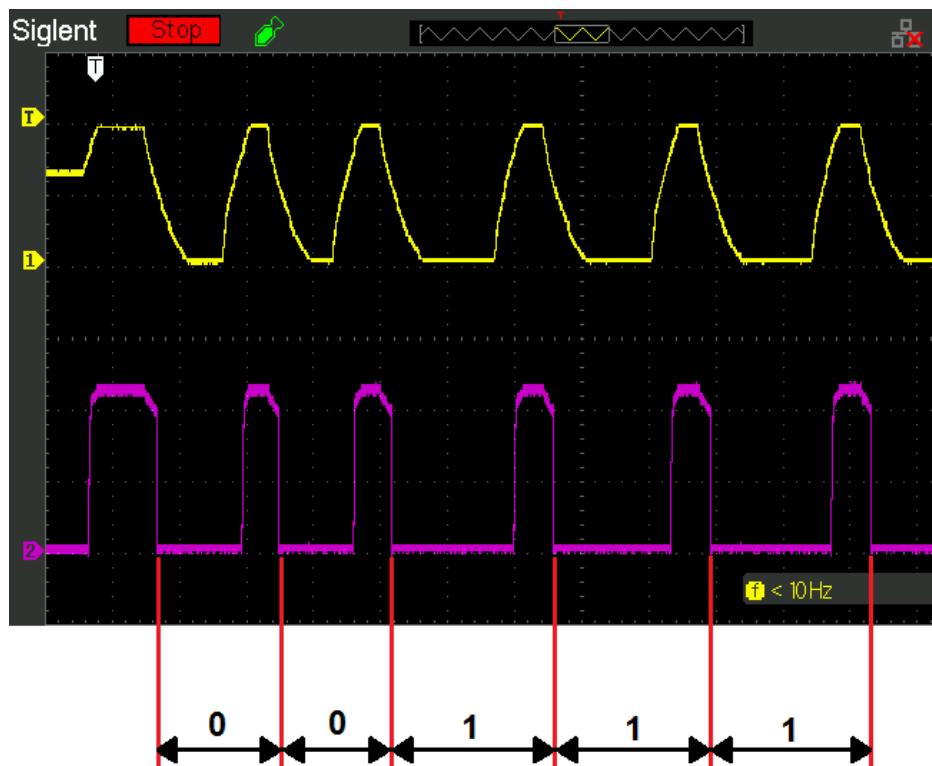
L'antenna è quella cerchiata in rosso. L'ho saldata sul retro perché il componente è a montaggio superficiale.

7.1 Software

In ingresso al microcontrollore arriva un treno di impulsi che corrisponde al comando inviato dal lettore. La codifica usata è chiamata Binary Pulse Length Modulation, uno "0" viene rappresentato da un impulso "corto" e un "1" viene rappresentato da un impulso "lungo":



Ad esempio, questa forma d'onda corrisponde all'invio del comando 00111 da parte del lettore:



È sufficiente misurare il periodo del treno di impulsi in ingresso per rilevare i bit inviati.

Per misurare il periodo viene utilizzata la modalità “Input Capture Unit” del Timer1. In questa modalità ogni volta che un “evento” (nel nostro caso un fronte di discesa) si verifica sul pin ICP1 (il pin a cui è collegata l’uscita del comparatore), il valore del Timer1 viene copiato all’interno del registro ICR1 il quale può essere usato per determinare se il bit trasmesso è uno zero o è un uno.

L’ATMEGA8 viene programmato per utilizzare il clock interno a 8MHz (125ns), il Timer1 viene configurato senza prescaler nella modalità “Fast PWM” in cui si può impostare nel registro OCR1A il valore di overflow del contatore. Da datasheet si ricavano le durate minime/massime degli impulsi zero/uno inviati dal lettore:

Bit inviato	Durata minima	Durata massima
Zero	144μs	176μs
Uno	208μs	256μs

Considerando il tempo di durata massima di uno “zero” e il tempo di durata minima di un “uno”, si può ricavare il valore di soglia per la decodifica:

Tempi	Valore Timer1
Durata massima bit Zero	$\frac{176 \mu s}{125 ns} = 1408$
Durata minima bit Uno	$\frac{208 \mu s}{125 ns} = 1664$

Si può impostare quindi 1500 come valore di soglia: da varie prove ho notato che è meglio aumentare leggermente il valore di soglia a 1700, che tiene conto dei vari ritardi che si vengono ad accumulare nel sistema. Questi sono i valori del Timer1 che ho misurato nel caso di una trasmissione del comando 00111:

Bit inviato	Valore Timer1
Zero	1493
Zero	1446
Uno	1911
Uno	1970
Uno	1895

Come vedete è molto più sicuro portare la soglia a 1700 piuttosto che lasciarla a 1500.

Il valore di overflow del Timer1 è impostato a 5000 (625μs), ben oltre la durata massima del bit “uno”.

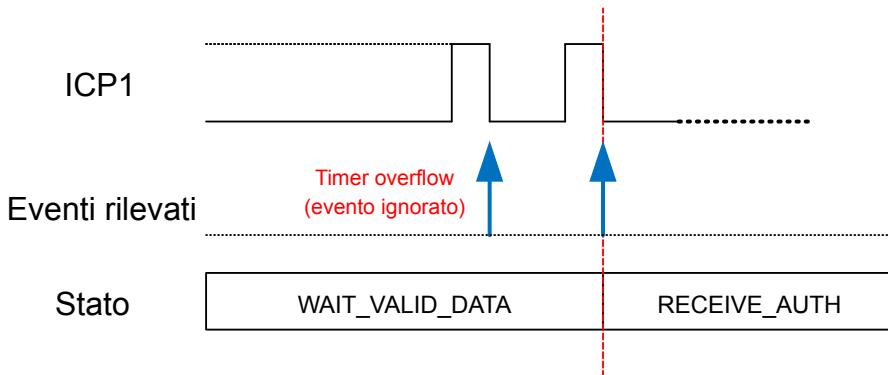
Lo scopo del software è quello di captare la comunicazione iniziale che avviene fra il blocchetto di accensione e la chiave: inizialmente viene inviato un comando AUTH, la chiave invia al lettore il proprio codice identificativo trasmesso in chiaro (che non ci interessa, dato che lo possiamo ricavare con l'AESHitager) e successivamente il lettore invia un dato a 64 bit che ci serve per ricavare la Secret Key usando lo script "Crack 5"; tutto il resto della comunicazione non ci interessa. Dobbiamo quindi identificare la trasmissione del comando AUTH, ignorare la risposta della chiave e salvare il dato a 64 bit trasmesso dal lettore.

La gestione della comunicazione avviene usando una "macchina a stati", che utilizza i seguenti stati:

- ***WAIT_VALID_DATA***

In questo stato viene atteso il primo bit valido di una comunicazione. Il Timer1 viene resettato e viene atteso il primo fronte di discesa del segnale in ingresso; quando questo evento arriva, il Timer1 viene resettato e viene salvato il dato di ICR1 che contiene l'informazione della durata del bit. Se il Timer1 è andato in overflow significa che siamo in presenza del primo fronte di discesa che appare sul pin: la comunicazione viene ignorata e nello step successivo si riparte dall'inizio dello stato.

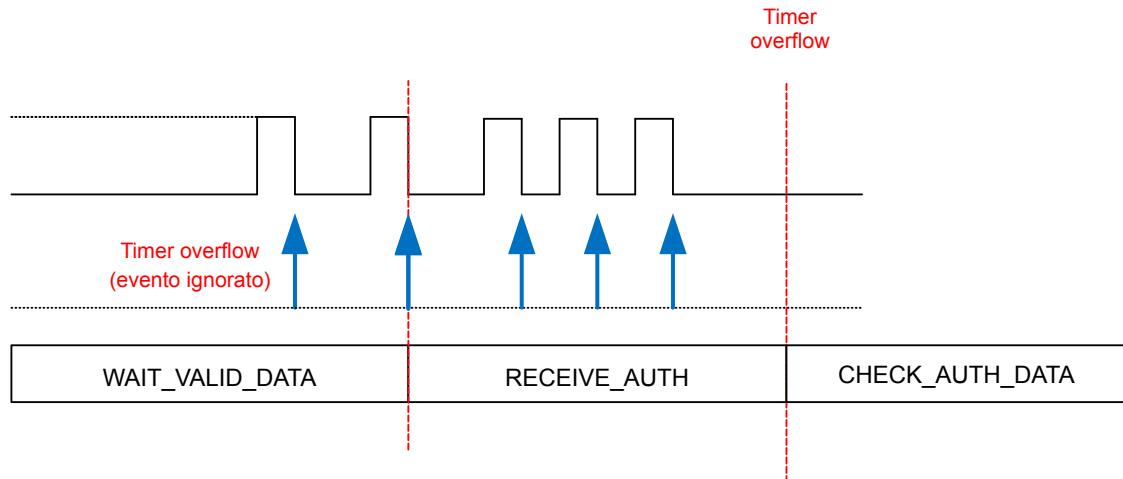
Se il Timer1 non è andato in overflow significa che siamo in presenza di un bit valido: la durata di esso viene confrontata con la soglia impostata e viene salvato il bit all'interno di rx_data. Lo stato successivo viene impostato a *RECEIVE_AUTH*, lo stato in cui continua la ricezione dei bit.



- ***RECEIVE_AUTH***

In questo stato il Timer1 viene resettato e viene atteso il fronte di discesa sul segnale di ingresso. Se durante questa attesa il Timer1 va in overflow, significa che la trasmissione del primo comando è terminata e il ciclo di attesa viene interrotto. Solo se vengono ricevuti cinque bit si passa allo stato successivo (*CHECK_AUTH_DATA*), in caso contrario significa che è stato captato un disturbo (o un'altra comunicazione) e si ritorna allo stato precedente.

Se il Timer1 non va in overflow, viene salvato il bit all'interno di rx_data come nello stato precedente.

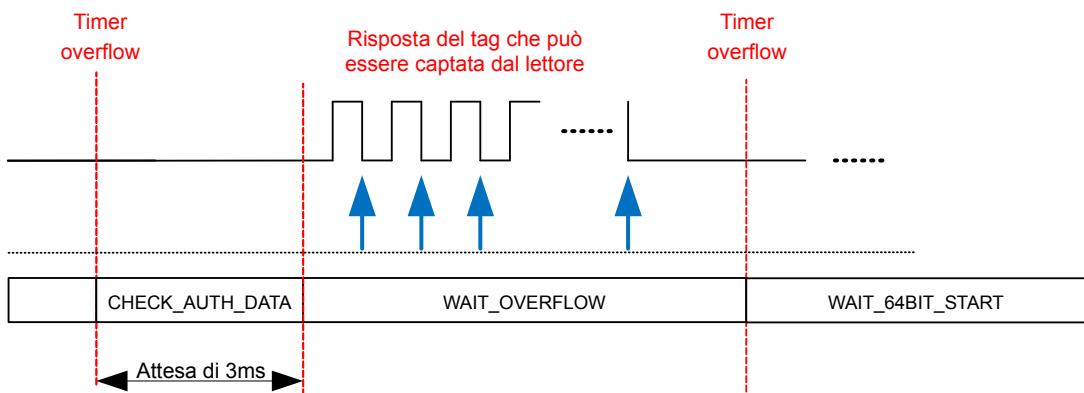


- **CHECK_AUTH_DATA**

Questo stato è molto semplice, viene fatto il check su rx_data: se questo è uguale a 11000 (comando AUTH) vengono attesi 3ms e lo stato successivo viene posto uguale a *WAIT_OVERFLOW*. In caso contrario, allo step successivo si ritorna allo stato *WAIT_VALID_DATA*, in attesa del comando AUTH.

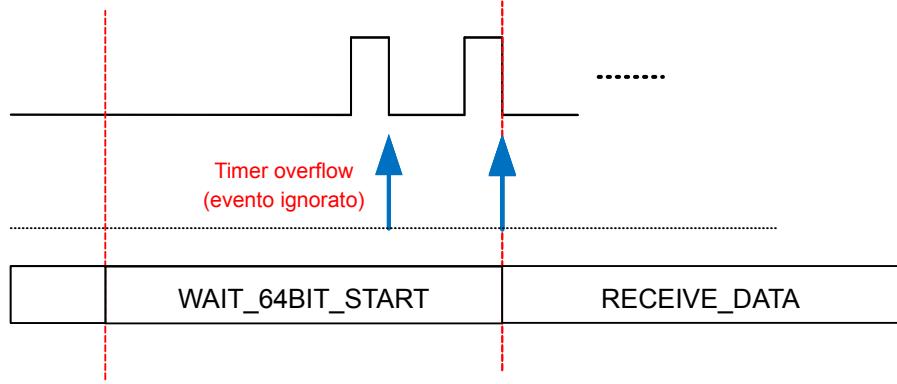
- **WAIT_OVERFLOW**

Nello stato precedente vengono attesi 3ms. In questo modo, quando arrivo in questo stato, mi trovo nel mezzo della risposta che viene inviata dal tag. Il Timer1 viene inizializzato e viene atteso il fronte di discesa del segnale in ingresso: se durante questa attesa il Timer1 va in overflow, significa che il tag ha finito di inviare la sua risposta e posso passare allo stato successivo in cui viene atteso l'invio del comando a 64bit da parte del lettore (*WAIT_64BIT_START*). In caso contrario, rimango in questo stato in attesa dell'overflow.



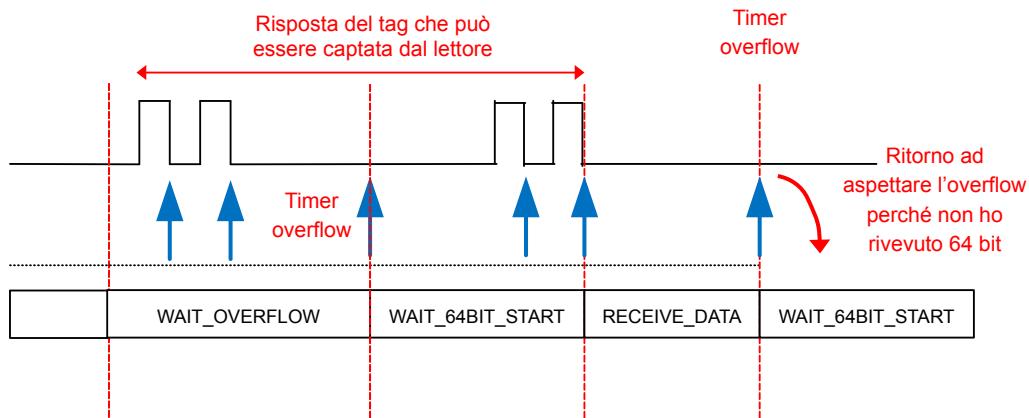
- **WAIT_64BIT_START**

Questo stato è analogo allo stato *WAIT_VALID_DATA*: il Timer1 viene inizializzato e viene atteso il primo fronte di discesa del segnale in ingresso: il primo overflow viene ignorato mentre il successivo bit valido viene salvato all'interno del dato data_h (vengono usate due variabili a 32 bit per salvare il dato a 64bit inviato dal lettore). Una volta ricevuto il primo bit valido, viene impostato lo stato successivo a *RECEIVE_DATA*.



- ***RECEIVE_DATA***

Questo stato è analogo allo stato *RECEIVE_AUTH*: il Timer1 viene inizializzato e viene atteso il fronte di discesa sul segnale di ingresso. Se durante questa attesa il Timer1 va in overflow, significa che possono essersi verificate due condizioni: o la trasmissione del comando a 64bit del lettore è terminata oppure la risposta captata dal lettore nello stato *WAIT_OVERFLOW* è stata tale da portarmi erroneamente in questo stato. Un esempio:



In base alla posizione dell’antenna rispetto alla chiave, la risposta del tag potrebbe non essere captata in maniera “continua” ma potrebbero esserci dei “buchi” fra i bit inviati come in figura oppure la larghezza dei bit stessi potrebbe essere tale da causare un overflow: a questo punto la macchina a stati si ritroverebbe nello stato *WAIT_64BIT_START* in attesa di un bit valido. Un’altra risposta captata erroneamente potrebbe portare la macchina stati nello stato *RECEIVE_DATA*: in questo caso, appena ricevo un overflow viene fatto un controllo sui bit ricevuti. Se questi sono 64, allora passo allo stato successivo (*CHECK_RX_DATA*), in caso contrario ritorno allo stato *WAIT_64BIT_START* in attesa di un overflow e di un dato valido (come mostrato nella figura qui sopra).

In questo modo vado a filtrare le comunicazioni spurious che possono essere captate dall’antenna.

- ***RECEIVE_DATA***

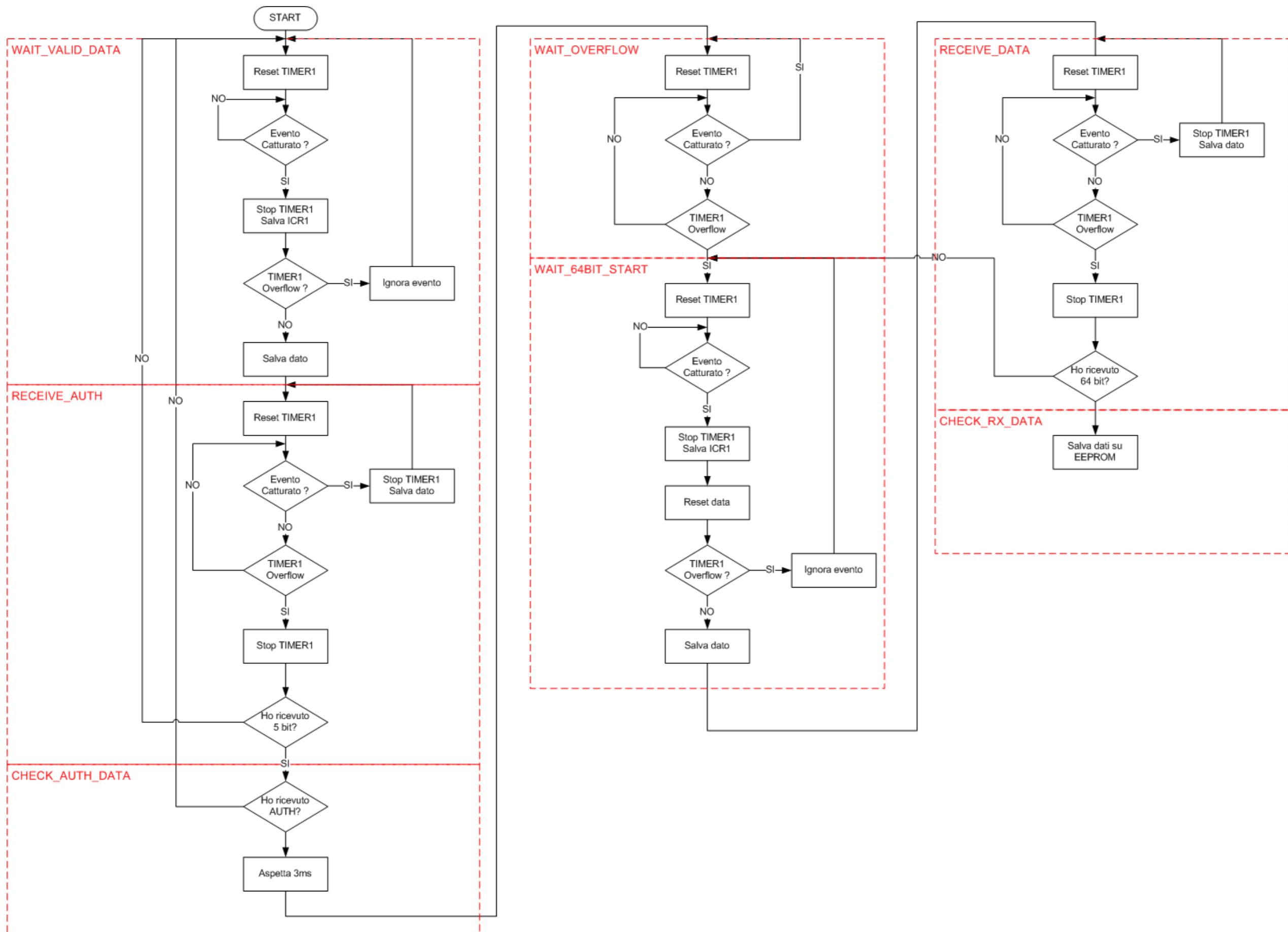
In questo stato vengono salvati all'interno della EEPROM i dati a 64bit ricevuti; dopo il salvataggio, la macchina a stati ritorna nello stato *WAIT_VALID_DATA* in attesa di un altro comando AUTH.

Solo due comunicazioni a 64bit vengono salvate nella EEPROM: è questo il numero di comunicazioni che ci serve per estrarre la Secret Key con lo script. È possibile anche disabilitare il salvataggio sulla EEPROM e comunicare via seriale il dato a 64bit ricevuto scrivendo

```
#define USE_SERIAL
```

all'inizio del main. Usando un modulo bluetooth è possibile inviare via seriale le comunicazioni sul proprio smartphone (questa modalità verrà mostrata più avanti).

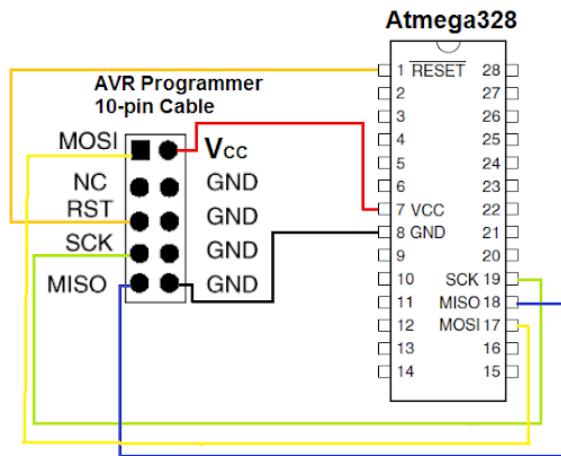
Nella pagina successiva è mostrato il diagramma di flusso della macchina a stati implementata.



7.2 Compilazione e scrittura sull'ATMEGA

Per la compilazione e la scrittura sull'ATMEGA vengono utilizzati WINAVR (scaricabile qui <https://sourceforge.net/projects/winavr/>) e il programmatore USBASP, acquistabile su internet a pochi euro.

Il collegamento del programmatore all'ATMEGA avviene in questo modo:



Per compilare il codice, utilizzare l'istruzione seguente al prompt dei comandi di Windows:

```
avr-gcc -Wall -g -Os -mmcu=atmega8 -std=gnu99 -o main.bin main.c
```

Per generare il file .hex

```
avr-objcopy -j .text -j .data -O ihex main.bin main.hex
```

Prima di scrivere il file .hex è necessario scrivere i fuse bit (NOTA: questa operazione è da fare una volta sola, non si deve ripetere ogni volta che si scrive il .hex)

```
avrdude -p atmega8 -c usbasp -U lfuse:w:0xC4:m  
avrdude -p atmega8 -c usbasp -U hfuse:w:0xD9:m
```

Per scrivere il file .hex nel microcontrollore:

```
avrdude -p atmega8 -c usbasp -U flash:w:main.hex:i -F -P usb
```

Per chi ha linux, può scaricare tutti i tool usando il comando

```
sudo apt-get install gcc-avr avr-libc avrdude
```

Le istruzioni per la compilazione e la scrittura sono le stesse usate su windows.

7.3 Debug del software

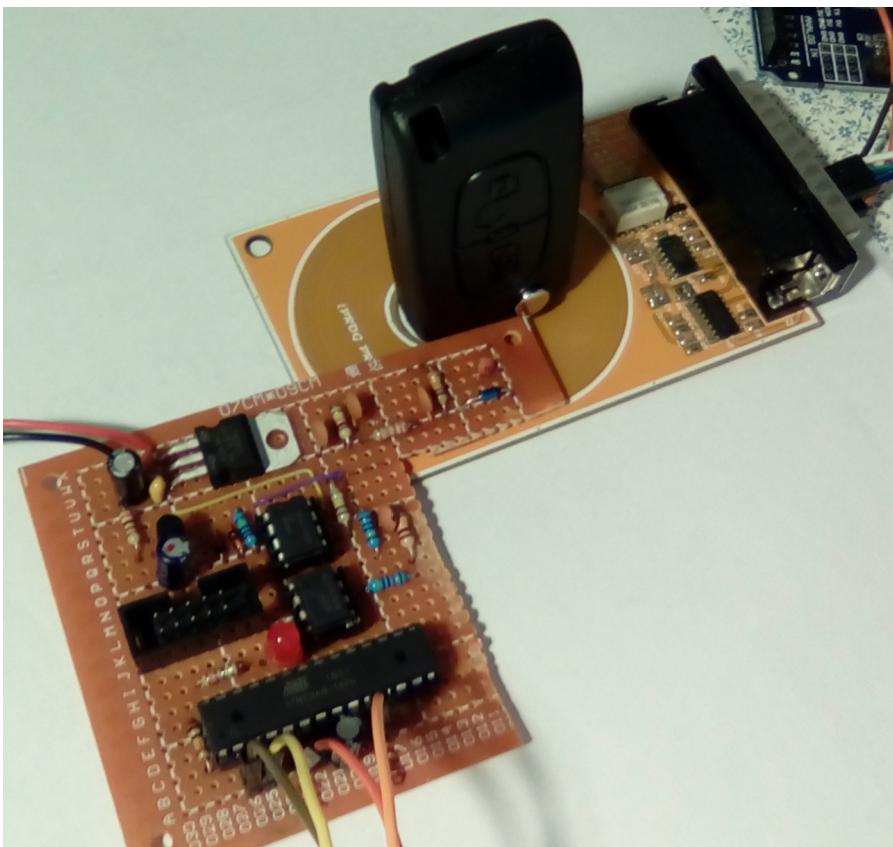
È possibile abilitare la modalità di debug scrivendo all'inizio del main.c

```
#define USE_SERIAL
```

In questo modo ogni volta che viene rilevata una comunicazione valida AUTH + 64bit, lo sniffer invierà il dato a 64bit sulla seriale (pin TXD).

Per testarlo con l'AESHitager, i passaggi sono i seguenti:

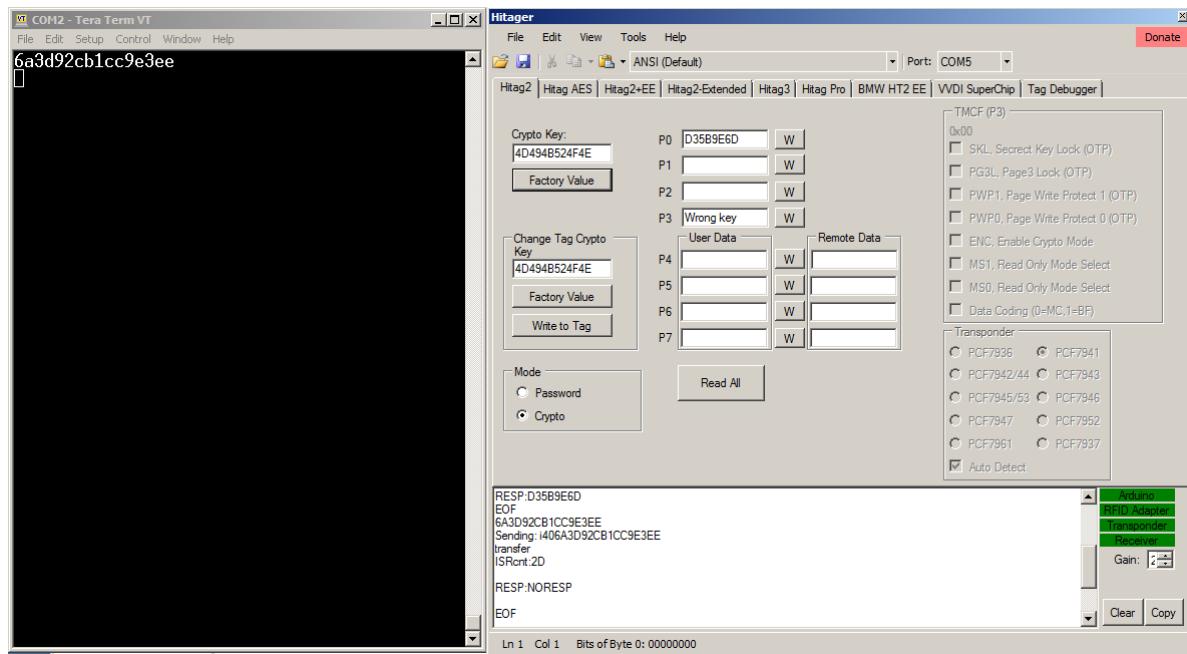
1. Aprire l'AESHitager e aumentare il "Gain" a 2.
2. Posizionare la chiave e lo sniffer come indicato in figura:



3. Collegare il pin TXD dell'Atmega al pin RX del modulo FTDI (o equivalente).
4. Aprire TeraTerm (o un altro terminale) ed usare questi settaggi per la seriale:

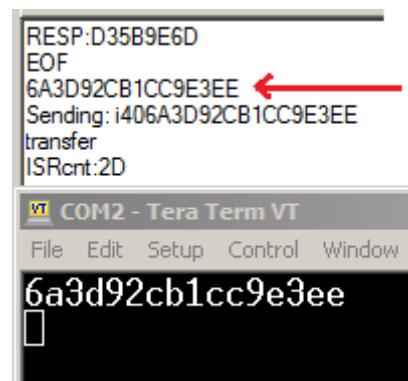
Baud rate	9600
Data	8 bit
Parity	None
Stop	1 bit

- Da AESHitager selezionare la modalità “Crypto” e premere “Read All” (nota: la crypto key deve essere sbagliata).
- Nel terminale della seriale deve comparire il dato a 64bit inviato dal lettore, come in figura:



- Per controllare che il dato inviato dal lettore sia stato riconosciuto correttamente dallo sniffer, è sufficiente guardare la casella di testo che si trova nella parte inferiore di AESHitager, dove sono loggati tutti i comandi inviati.

In figura, dopo RESP:D35B9E6D (è la risposta inviata dalla chiave, il suo codice identificativo che si trova nella Page 0) viene inviata una stringa di 64bit: è proprio questo che lo sniffer deve intercettare e dal terminale si può notare che la comunicazione è stata captata nella maniera corretta.

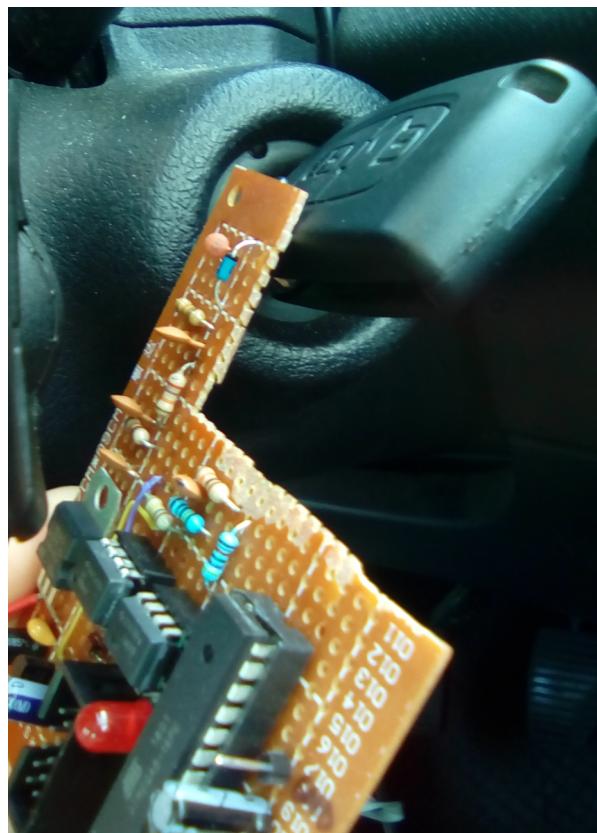


- Si può anche testare il range di funzionamento dello sniffer provando ad allontanarlo dal lettore. Generalmente la comunicazione riesce ad essere captata anche ad una distanza di 4/5 cm.

8 Sniffing della comunicazione

Ora che lo sniffer è pronto, dobbiamo sniffare la comunicazione fra il blocchetto di accensione e la chiave della nostra automobile. Per fare questo dobbiamo:

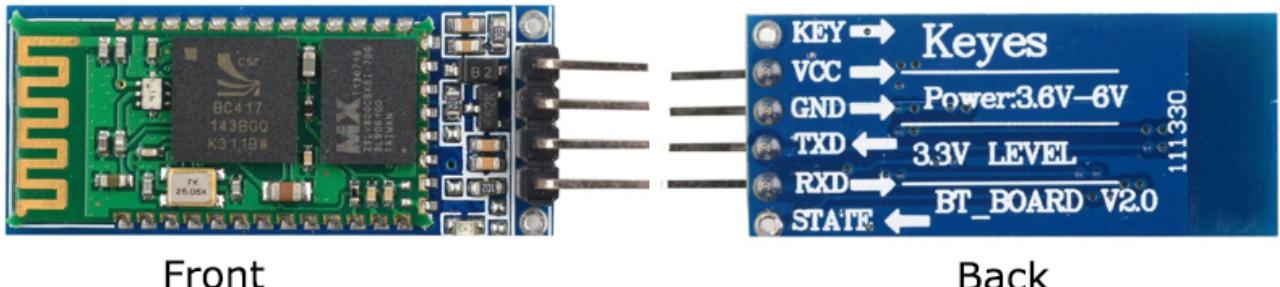
1. Aprire la nostra auto, accenderla e poi spegnerla. Ho notato che alla prima accensione successiva all'apertura dell'automobile, il blocchetto di accensione invia due comandi identici di autenticazione (almeno questo è quello che sembra dallo sniffer) appena viene inserita la chiave (senza girarla per accendere). Questo causa il salvataggio di due dati a 64bit identici nella EEPROM, per cui evitiamo di sniffare questa comunicazione.
2. Togliere la chiave dal blocchetto e poi ri-infilarla. Successivamente, accendere lo sniffer e posizionarlo come in figura:



3. Girare la chiave per accendere l'automobile. Il led collegato al pin PC0 dell'Atmega deve lampeggiare.
4. Spegnere l'automobile, togliere la chiave dal blocchetto e poi ri-infilarla. Posizionare lo sniffer come nella posizione precedente e poi girare la chiave per accendere l'automobile. Il led collegato al pin PC0 dell'Atmega deve lampeggiare.
5. A questo punto le due comunicazioni a 64bit sono state captate correttamente.

8.1 Utilizzo di un modulo Bluetooth

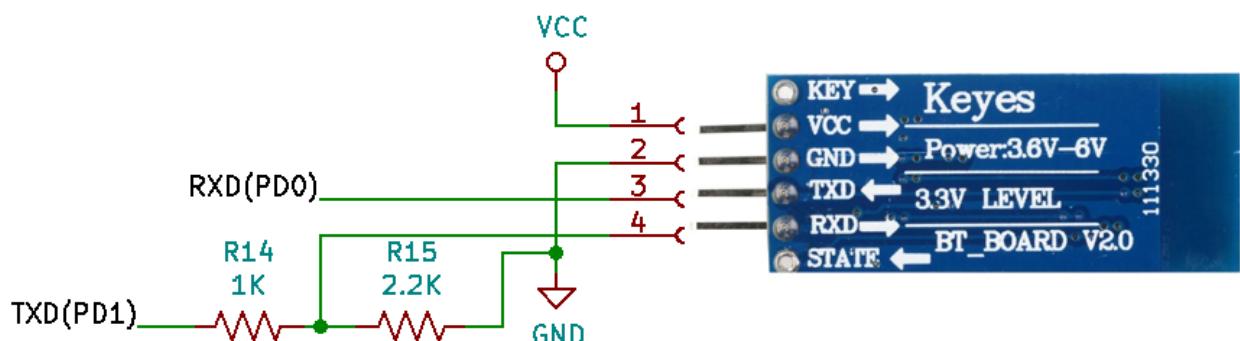
In alternativa al salvataggio sulla EEPROM interna del microcontrollore, è possibile abilitare la comunicazione tramite porta seriale (come descritto nel paragrafo [7.3.Debug del software](#)) ed utilizzare il modulo HC-06 per la comunicazione bluetooth.



Il modulo è acquistabile qui <https://it.aliexpress.com/item/32342784842.html>

Questi moduli permettono di realizzare un “link” seriale fra un terminale Bluetooth e l'ATMEGA: in questo modo è possibile visualizzare la comunicazione sniffata sul proprio smartphone, senza dover passare dalla lettura della EEPROM interna del microcontrollore.

Il collegamento da effettuare con l'ATMEGA è il seguente:

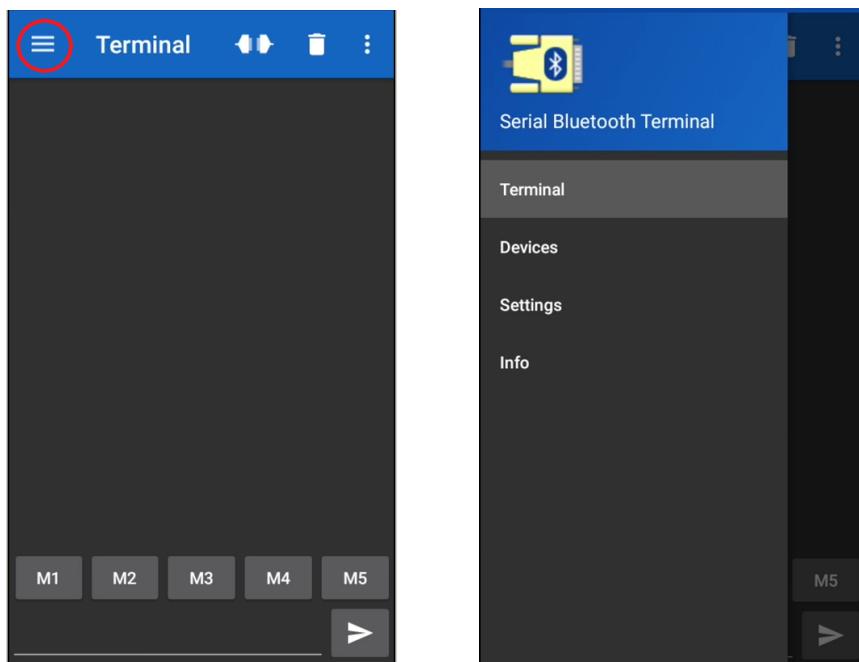


Come applicazione da usare su Android consiglio “Serial Bluetooth Terminal”
https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal

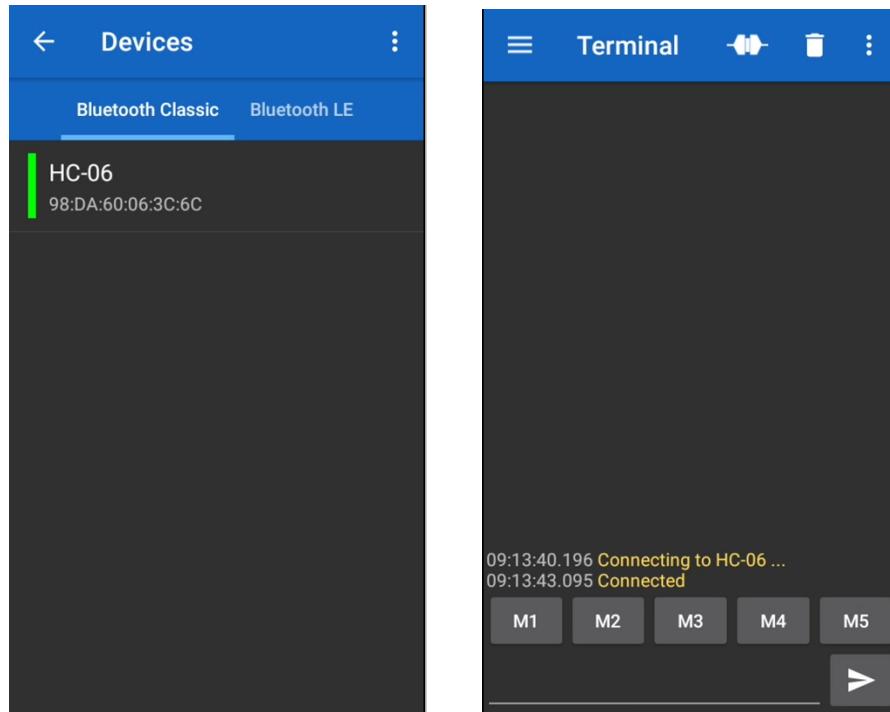
Una volta che il modulo HC-06 è alimentato, compare fra i dispositivi Bluetooth con il nome **HC-06**: dovete fare il pairing con il vostro smartphone usando come pin 1234



Ora dovete aprire il “Serial Bluetooth Terminal” e selezionare il menù a tendina:



Da “Devices” selezionate il modulo HC-06 dalla tab “Bluetooth Classic”. Una volta selezionato il dispositivo, comparirà nel terminale la scritta “Connected”:



Sempre dal menù a tendina selezionate “Settings” e in “Display mode” impostate “Terminal”.

A questo punto il link seriale è stabilito e i dati trasmessi dall’ATMEGA verranno inviati correttamente al terminale.

Potete salvare i dati ricevuti selezionando dal menù in alto a destra “Data” → “Save”

I file salvati li trovate dentro **Android\data\de.kai_morich.serial_bluetooth_terminal\files**

Ecco l’esempio di un log:

```
07:50:19.899 Connecting to HC-06 ...
07:50:20.348 Connected
07:50:34.056 6561da17ba1d4249
07:50:34.056 ♦5ae959e1c08a0165
07:50:55.934 ♦0b0b40e0d75192a0
07:51:07.576 ♦
```

In giallo sono evidenziate le comunicazioni a 64bit captate dallo sniffer.

8.2 Estrazione dei codici

Per estrarre i due dati a 64bit e per ottenere il comando da usare con lo script “Crack 5” ho realizzato uno script in Python (se utilizzate il modulo Bluetooth non è necessario estrarre i dati dalla EEPROM).

Gli step per utilizzarlo sono:

1. Eseguire il comando:

```
avrdude -c usbasp -p m8 -P usb -U eeprom:r:"EEPROM.hex":r
```

con l’USBASP collegato allo sniffer. Viene letta la EEPROM del micro e il suo contenuto viene salvato in un file binario chiamato EEPROM.hex

2. Scaricare lo script “create_command.py” che si trova nella repository.
3. Inserire dentro “id_code” il codice identificativo della vostra chiave che ottenete con l’AESHitager (il dato viene trasmesso in chiaro per cui basta premere su “Read All” anche se la Crypto Key non la conosciamo). È il dato che si trova nella Page 0 (P0).
Esempio: id_code = "AABBCCDD"
4. Eseguire lo script scrivendo:

```
python create_command.py
```

5. Lo script vi mostra i due dati a 64bit che sono stati captati e vi mostra il comando da utilizzare con lo script “Crack 5”.

```
First sniff = 7e8cd9f388fabc95
Second sniff = 5d4f2b1d34056db1

Command to use :
./ht2crack5 D35B9E6D 7e8cd9f3 88fabc95 5d4f2b1d 34056db1
```

6. Eseguire lo script ht2crack5 compilato con le istruzioni presenti nella sezione [Algoritmo di attacco](#).

```
:crack5$ ./ht2crack5 D35B9E6D 7e8cd9f3 88fabc95 5d4f2b1d 34056db1
Thread 0 slice 1/512
Thread 1 slice 1/512
Thread 2 slice 1/512
Thread 3 slice 1/512
```

Lo script ci può mettere dalle due alle cinque ore per estrarre la Crypto Key, usando la CPU al 100% (io uso un computer portatile con Linux per evitare di avere il computer principale "bloccato" dallo script).

7. Quando lo script ha concluso le sue operazioni vi comparirà a video la Crypto Key come in figura:

```
Thread 3 slice 224/512
Thread 1 slice 220/512
Thread 0 slice 222/512
Thread 2 slice 224/512
Key: 4F4E4D494B52
      :crack5$
```

Attenzione che la Crypto Key ricavata dallo script deve essere inserita in maniera diversa all'interno del software AESHitager: i primi due byte devono essere spostati in fondo, in questa maniera:

Key ricavata dallo script	Key da inserire nel software
4F4E4D494B52	4D494B524F4E

Questo è dovuto ad un errore presente nell'AESHitager che considera i 2 byte più significativi della chiave come i meno significativi.

8. Una volta inserita la Crypto Key correttamente nel software AESHitager si può leggere tutto il contenuto della chiave. La Page 1 e la Page 2 che contengono la Crypto Key, generalmente sono lockate in lettura, tutti gli altri campi sono leggibili senza problemi.

Se utilizzate il modulo Bluetooth avete già a disposizione i due dati a 64bit, esempio:

```
07:50:19.899 Connecting to HC-06 ...
07:50:20.348 Connected
07:50:34.056 6561da17ba1d4249
07:50:34.056 5ae959e1c08a0165
07:50:55.934 0b0b40e0d75192a0
07:51:07.576 ?
```

Una volta ricavato il codice identificativo come indicato al passo 3, il comando da inviare è il seguente (i due dati a 64bit vanno "spezzati" in quattro dati a 32bit):

```
./ht2crack5 AABBCCDD 5ae959e1 c08a0165 0b0b40e0 d75192a0
```

Dove con AABBCCDD si intende il contenuto della Page 0.

9 Copia fisica della chiave

Per realizzare una copia fisica della chiave possiamo acquistare una chiave “vergine” da Aliexpress e utilizzare il servizio di fresatura CNC offerto dallo stesso venditore.

Qui il link della chiave vergine in vendita:

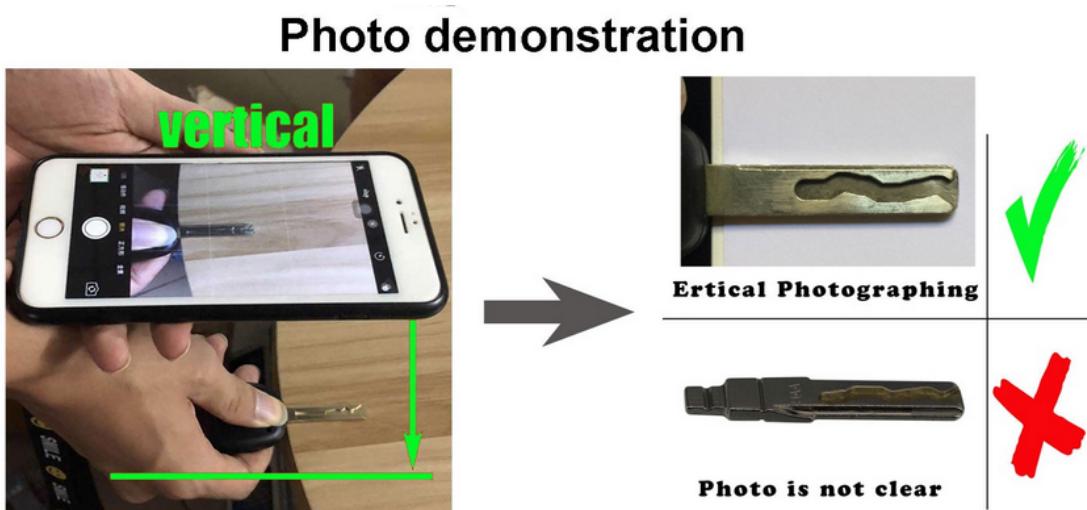
<https://it.aliexpress.com/item/33036657360.html>

Qui il link del servizio offerto di fresatura CNC:

<https://it.aliexpress.com/item/33030748490.html>

Per fare una copia della propria chiave gli step sono:

- Fare una foto fronte/retro della propria chiave, come indicato nel link del servizio di fresatura:



- Inviare al venditore della chiave le foto, chiedendogli se sono state scattate correttamente per la copia. In caso di risposta affermativa, effettuare l'ordine della chiave vergine dal primo link che ho inserito sopra.
- Per determinare quale chiave acquistare dovete osservare quella della vostra auto. Ci sono di fatto quattro tipi di chiavi per le Peugeot: cambiano in base al codice presente a lato quando la chiave è aperta (può essere **CE0536** oppure **CE0523**) e in base al tipo di taglio (solco laterale presente nelle chiavi **HCA** e assente in quelle **VA2**). Nel link sono presenti delle immagini di esempio per cui sarà facile identificare quale chiave acquistare.

Una nota importante:

Anche se la chiave presenta al suo interno un circuito “completo” con tag e radiocomando, non è possibile utilizzarlo: il tag ha la Page 0 in sola lettura (come tutti i tag) per cui non sarà possibile clonare completamente la propria chiave. La vostra automobile non riconosce altri tag con codici identificativi diversi, a meno di riprogrammare direttamente la centralina, operazione abbastanza difficile (che non so minimamente come effettuare).

Stessa cosa per il radiocomando: i datasheet che si trovano in rete dei vari PCF7941/PCF7961 non sono completi e risulta difficile capire come programmare correttamente il radiocomando.

Dobbiamo accontentarci: la copia della chiave auto che andremo a realizzare non avrà il radiocomando (l'apertura/chiusura dell'auto va fatta necessariamente con la chiave) e per il cloning completo dovremo acquistare un chip che permette la copia 1:1 di un tag PCF: il VVDI Super Chip.

9.1 VVDI Super Chip

In commercio esistono diversi tipi di tag per clonare le chiavi auto: molti di questi non hanno un datasheet e le informazioni a disposizione sono scarne. Generalmente vanno acquistati dispositivi commerciali per programmare questo tipo di tag.

Ho scelto il VVDI Super Chip della Xhorse perché questo tag può essere scritto con il software AESHitager. Complimenti all'autore del software per essere riuscito a capire come scrivere dentro questi tag, dato che non esistono molte informazioni dettagliate in rete.

Il tag lo potete acquistare su Aliexpress qui:

<https://it.aliexpress.com/item/1005003020409304.html>

Oppure su Amazon qui:

<https://www.amazon.it/GEEKEN-pezzo-XT27A01-XT27A66-transponder/dp/B08XK6DRVG/>

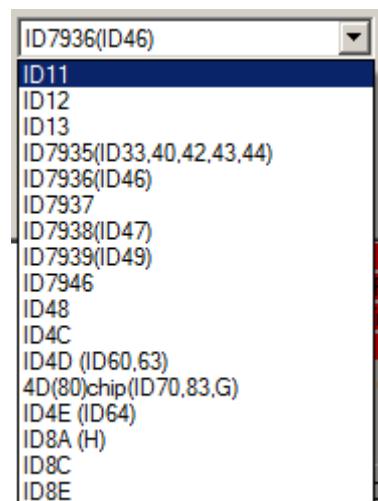
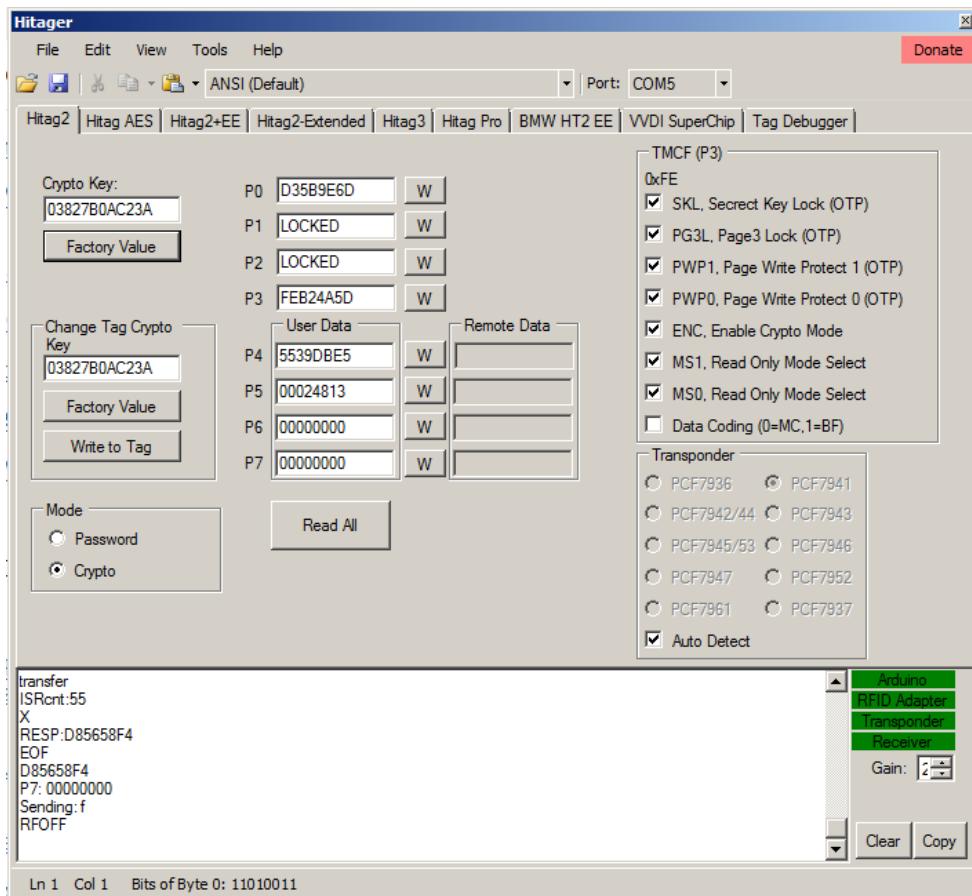


Foto del tag / Elenco dei tag che il chip riesce ad emulare

Il tag riesce ad emulare correttamente i PCF7935/7936/7937/7938/7939/7946 (nell'immagine sono indicati con ID*).

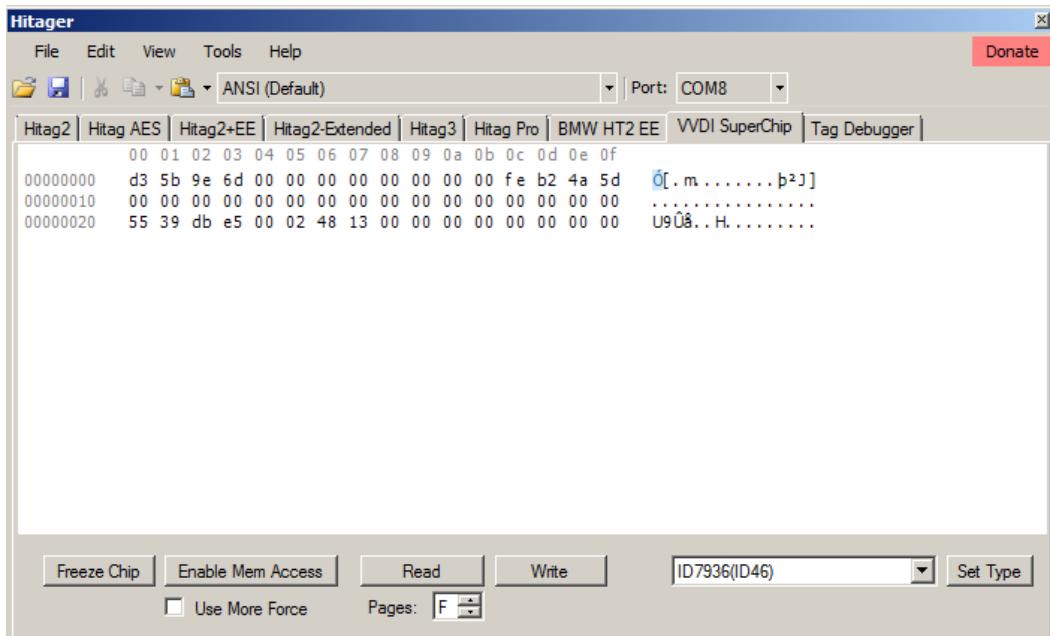
Una chiave auto letta con AESHitager, si presenta generalmente così:



Tutti le pagine vengono lette tranne la Page 1 e la Page 2: in queste due pagine è scritta la Crypto Key. Noi però la conosciamo (avendola ricavata nei passi precedenti) per cui il contenuto delle due pagine sarà il seguente:

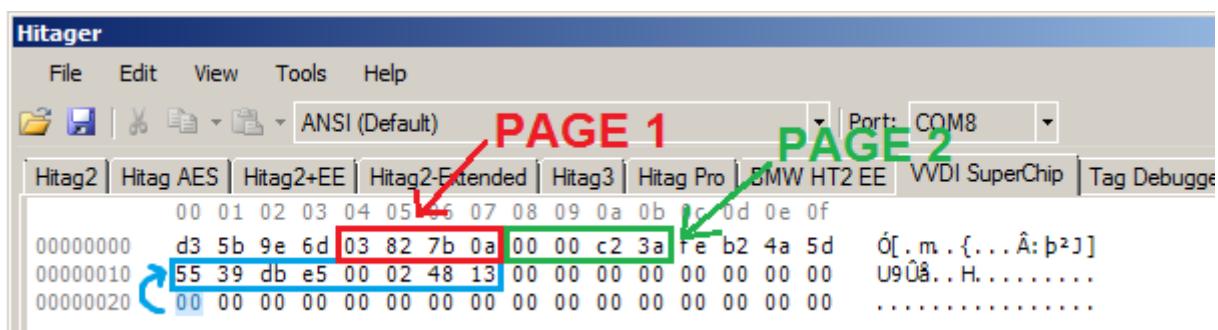
Crypto Key usata dal software	03827B0AC23A
Page 1	03827B0A
Page 2	0000C23A

Ora dobbiamo selezionare il tab VVDI SuperChip nel software. La schermata che compare è la seguente:



Prima di scrivere sul tag bisogna modificare i dati presenti nell'editor esadecimale del software:

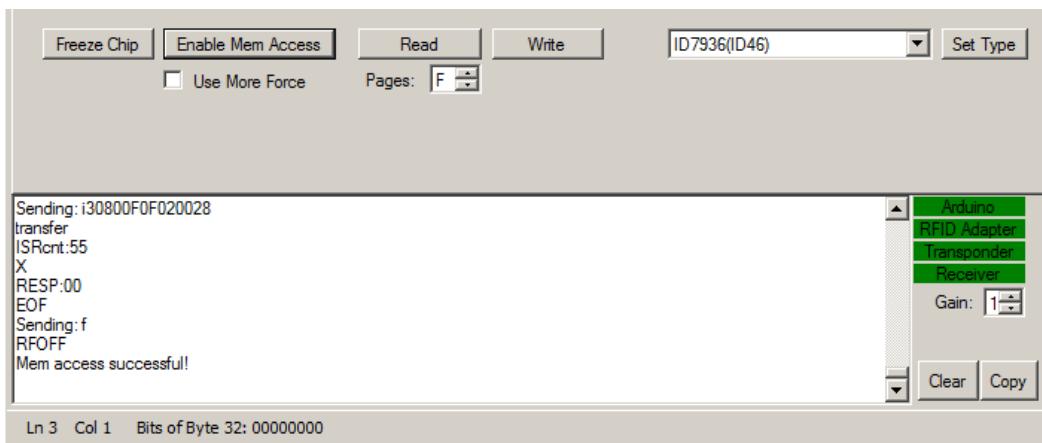
- Nella prima riga, nelle colonne 04 / 05 / 06 / 07 va inserita la Page 1.
- Nella prima riga, nelle colonne 08/ 09 / 0a / 0b va inserita la Page 2.
- La terza riga deve essere copiata nella seconda riga.
- La terza riga va messa a zero.



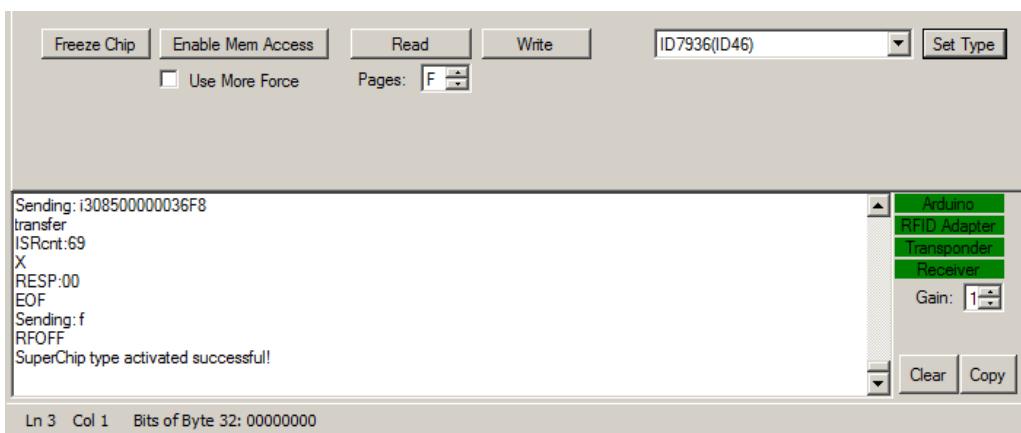
Il tag VVDI Super Chip va posizionato al centro del lettore, come mostrato in figura:



Il Gain va portato a 1 e poi dovete cliccare su “Freeze Chip”. Dopo che ha finito le operazioni, dovete cliccare su “Enable Mem Access”. Se l’operazione è andata a buon fine compare la scritta “Mem access successful!”

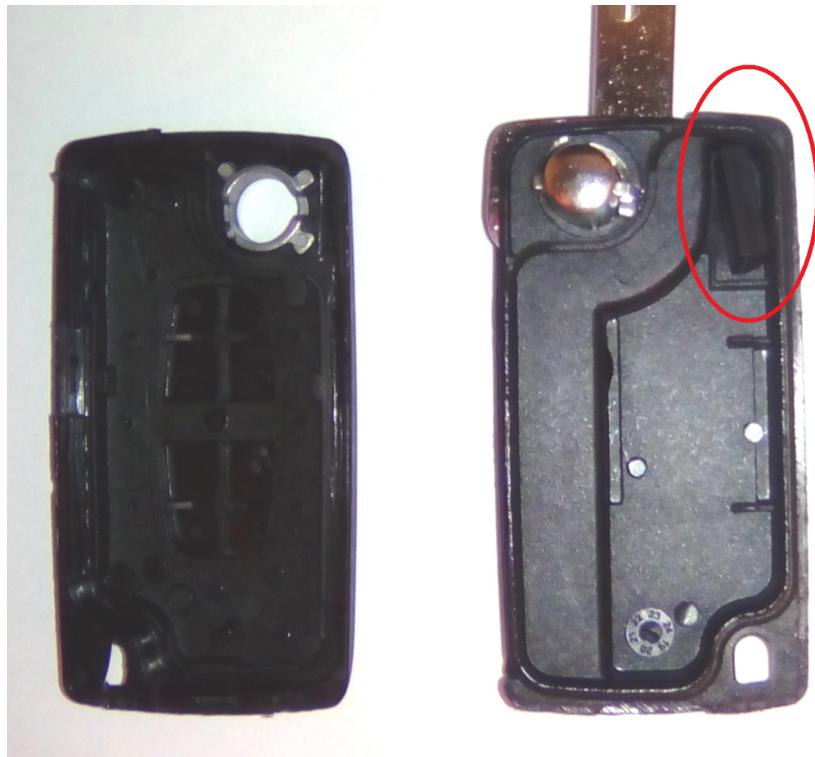


Ora dovete cliccare su “Write” per scrivere la memoria interna del tag. Fatto ciò, selezionate dal menù a tendina “ID7936(ID46)” e cliccate poi su “Set Type”. Se tutto è andato a buon fine, comparirà la scritta “SuperChip type activated successfully!”.



A questo punto potete tornare nel tab “Hitag2” e verificare che tutto sia stato scritto correttamente.

Ora non ci rimane che aprire la chiave che abbiamo ordinato da Aliexpress, rimuovere il circuito interno e piazzare il tag nella posizione mostrata in figura:



Consiglio di fissare il tag con una punta di colla a caldo.

L'operazione di cloning è conclusa, non rimane che testare la chiave sulla vostra auto.

10 Appendice A – Compilare Crack5 su Windows

Per compilare il codice su Windows è necessario scaricare MSYS2 (<https://www.msys2.org/>) seguendo tutte le istruzioni presenti nel sito.

Per prima cosa installate il package git per MSYS2:

```
pacman -S git
```

Fate il clone della repository nella vostra home di MSYS2:

```
git clone https://github.com/RfidResearchGroup/proxmark3.git
```

Entrate nella cartella "crack5":

```
cd proxmark3/tools/hitag2crack/crack5/
```

Prima di procedere con la compilazione, dovete modificare a mano il file **ht2crack5.c**

Alla riga 98 è presente questa funzione:

```
static int num_CPs(void) {
#if defined(_WIN32)
#include <sysinfoapi.h>
    SYSTEM_INFO sysinfo;
    GetSystemInfo(&sysinfo);
    return sysinfo.dwNumberOfProcessors;
#else
#include <unistd.h>
    int count = sysconf(_SC_NPROCESSORS_ONLN);
    if (count < 2)
        count = 2;
    return count;
#endif
}
```

Il compilatore installato con MSYS2 non permette di avere gli include all'interno di una funzione, la cosa si risolve mettendo gli include in testa al file (riga 20):

```
#include <pthread.h>
#include "ht2crackutils.h"

#if defined(_WIN32)
    #include <sysinfoapi.h>
#else
    #include <unistd.h>
#endif
```

Alla riga 89 la funzione deve essere modificata così (vanno tolti gli include):

```
static int num_CPUs(void) {
#if defined(_WIN32)
    SYSTEM_INFO sysinfo;
    GetSystemInfo(&sysinfo);
    return sysinfo.dwNumberOfProcessors;
#else
    int count = sysconf(_SC_NPROCESSORS_ONLN);
    if (count < 2)
        count = 2;
    return count;
#endif
}
```

A questo punto si possono eseguire le istruzioni per il build:

```
make clean
make
```