ISSN 2039-7941
anno 2026 numero 531



RAPPORTI TECNICI INGV

Applicazione Android per la generazione e lettura di codici Quick Response cifrati

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

531

Direttore Responsabile
Daniela VERSACE

Editor in Chief
Milena MORETTI

Editorial Board
Laura ALFONSI
Sofia BARANELLO
Carla BARNABA
Christian BIGNAMI
Simona CARANNANTE
Maria Grazia CIACCIO
Luca COCCHI
Luigi CUCCI
Lorenzo CUGLIARI
Elisabetta D'ANASTASIO
Alessia DI CAPRIO
Roberto DI MARTINO
Domenico DI MAURO
Domenico DORONZO
Filippo GRECO
Alessandro IAROCCI
Mario MATTIA
Daniele MELINI
Anna NARDI
Lucia NARDONE
Marco OLIVIERI
Arianna PESCI
Stefano PINTORE
Pierangelo ROMANO
Paola SBARRA
Davide SCAFIDI
Maurizio SOLDANI
Sara STOPPONI
Umberto TAMMARO
David ZULIANI

REGISTRAZIONE AL TRIBUNALE DI ROMA N.173 | 2014, 23 LUGLIO
© INGV Istituto Nazionale di Geofisica e Vulcanologia
Rappresentante legale: Presidente INGV
Sede: Via di Vigna Murata, 605 | Roma
Tutti i diritti riservati

www.ingv.it

ISTITUTO NAZIONALE **DI GEOFISICA E VULCANOLOGIA**

ISSN 2039-7941
anno **2026** numero **531**

RAPPORTI TECNICI INGV

Applicazione Android per la generazione e lettura di codici Quick Response cifrati

Android application for generating and reading encrypted Quick Response code

Stefano Farroni

INGV | Istituto Nazionale di Geofisica e Vulcanologia, Osservatorio Nazionale Terremoti

Accettato 9 settembre 2026 | *Accepted 9 September 2026*

Come citare | *How to cite* Farroni, S. (2026). Applicazione Android per la generazione e lettura di codici Quick Response cifrati. *Rapporti Tecnici INGV*, *531*, 32 pp. https://doi.org/10.13127/rpt/531

In copertina Rielaborazione grafica a cura di P. Pantani: Come controllare un accesso tramite QR Secure (immagine generata con IA) | *Cover How to store a password with QR Secure (AI generated image)*

531

INDEX

Riassunto 7

Abstract 7

Introduzione 8

1. Fondamenti teorici 9

2. Architettura del sistema e tecnologie utilizzate 9

3. Flusso operativo dell'applicazione 10

4. Funzionalità dell'applicazione 18

5. Sicurezza implementata 20

6. Casi d'uso 22

7. Conclusioni ed evoluzioni future 24

Bibliografia 25

Appendice 27

Riassunto

Il presente documento descrive una piattaforma denominata QR Secure, realizzata con l'obiettivo di consentire la generazione, la condivisione e la lettura di Quick Response Code (di seguito QR Code) contenenti informazioni che si vuole mantenere segrete.
La piattaforma impiega tecniche crittografiche moderne, basate su cifratura simmetrica AES-256 e su un'infrastruttura *backend* dedicata alla gestione sicura delle chiavi crittografiche.
L'architettura del sistema è composta da tre componenti principali:

- applicazione mobile sviluppata in Delphi [Buchanan W., 1998];
- *backend* realizzato con Python FastAPI;
- database PostgreSQL per la memorizzazione sicura delle chiavi.

L'intero processo prevede la generazione di una chiave casuale AES-256 (256 bit) e di un vettore di inizializzazione (IV) da 128 bit, la cifratura dei dati sul dispositivo, la memorizzazione protetta delle chiavi sul server mediante doppia cifratura con Master Key AES-256, e la successiva decifratura autorizzata in fase di lettura del QR Code.
Questo documento illustra i principi teorici adottati, l'architettura software, le soluzioni implementative, il flusso operativo completo con esempi pratici, e le funzionalità offerte dall'applicazione.
Il documento è organizzato distinguendo progressivamente obiettivi e fondamenti teorici (capitolo 1), architettura del sistema e tecnologie utilizzate (capitolo 2), flusso operativo dell'applicazione (capitolo 3), funzionalità dell'applicazione (capitolo 4), sicurezza implementata (capitolo 5), casi d'uso con relativa discussione critica (capitolo 6) e conclusioni ed evoluzioni future (capitolo 7); la procedura di attivazione dell'account e installazione dell'app, di natura puramente operativa, è riportata in appendice.

Abstract

This document describes a platform called QR Secure, designed to enable the generation, sharing, and reading of Quick Response Codes (hereinafter QR Codes) containing information that must be kept secret. The platform uses modern cryptographic techniques based on AES-256 symmetric encryption and a backend infrastructure dedicated to securely managing cryptographic keys. The system architecture is composed of three main components:

- *a mobile application developed in Delphi;*
- *a backend built with Python FastAPI;*
- *a PostgreSQL database for securely storing keys.*

The entire process involves generating a random AES-256 (256-bit) key and a 128-bit initialization vector (IV), encrypting the data on the device, storing the keys securely on the server using double encryption with an AES-256 master key, and subsequently authorizing decryption when reading the QR Code. This document illustrates the theoretical principles adopted, the software architecture, the implementation solutions, the complete operational flow with practical examples, and the features offered by the application.
The document is organized by progressively distinguishing objectives and theoretical foundations (Chapter 1), system architecture and technologies used (Chapter 2), application operational flow (Chapter 3), application functionality (Chapter 4), implemented security (Chapter 5), use cases with their critical discussion (Chapter 6), and conclusions and future developments (Chapter 7). The account activation and app installation procedure, which is purely operational in nature, is reported in the appendix.

Keywords Codice QR; Crittografia | QR Code; Cryptography; Android

Introduzione

Per comprendere il valore del QR Code, è necessario ripercorrere brevemente l'evoluzione dei sistemi di codifica delle informazioni. Se il codice a barre lineare (codice a barre 1D), introdotto negli anni '70, ha rivoluzionato la logistica permettendo la lettura veloce di brevi sequenze numeriche, esso presentava un limite intrinseco: la capacità di immagazzinare dati era direttamente proporzionale alla sua lunghezza fisica. Il salto di qualità avvenne nel 1994 in Giappone, quando l'azienda Denso Wave sviluppò il QR Code. A differenza dei suoi predecessori, il QR Code è bidimensionale (2D) e utilizza una matrice di punti bianchi e neri. Questa struttura consente di immagazzinare centinaia di volte più informazioni rispetto a un codice a barre tradizionale, inclusi caratteri alfabetici, numerici, binari e persino ideogrammi come i Kanji. Inoltre, grazie ai pattern di allineamento e ai codici di correzione degli errori (Reed-Solomon), il QR Code è estremamente resiliente: può essere letto anche se parzialmente danneggiato o oscurato. Tuttavia, questa potenzialità ha sempre celato un paradosso: il QR Code è nato per essere pubblico e leggibile da chiunque possegga una fotocamera. Questa caratteristica, se da un lato ne ha decretato il successo per il marketing e la condivisione di link, dall'altro ne ha limitato l'uso in contesti sensibili, dove la riservatezza dei dati è prioritaria.

La necessità di proteggere le informazioni non è un'invenzione dell'era digitale, ma accompagna l'umanità sin dall'antichità. Gli Spartani utilizzavano la Scitala (un bastone attorno al quale avvolgevano una striscia di cuoio per scrivere il messaggio in orizzontale) per trasmettere ordini militari. Giulio Cesare adottava il cifrario di Cesare, un semplice spostamento alfabetico, per comunicare con i suoi generali. Durante il Rinascimento, cifrari sempre più complessi, come quello di Vigenère, tentavano di resistere alle tecniche di crittanalisi emergenti.

Con l'avvento dell'era informatica, la crittografia ha subito una rivoluzione copernicana. Si è passati dalla crittografia simmetrica (una sola chiave per cifrare e decifrare) alla crittografia asimmetrica (chiave pubblica e chiave privata), gettando le basi per la sicurezza di Internet, dell'e-commerce e delle comunicazioni moderne. Oggi, la crittografia non è più solo uno strumento bellico o diplomatico, ma un diritto fondamentale della privacy del cittadino digitale. Protegge i nostri messaggi, i dati sanitari, le transazioni bancarie e la nostra identità online. In un mondo iperconnesso, cifrare significa riappropriarsi del controllo sui propri dati.

Unire il QR Code alla crittografia risponde a una precisa esigenza del mondo contemporaneo: la necessità di trasferire dati sensibili in canali fisici o visivi in totale sicurezza. Tradizionalmente, se stampiamo un QR Code contenente una password o un token di accesso, chiunque lo fotografi potrà leggerlo. La soluzione proposta in questo lavoro ribalta questo paradigma. Invece di memorizzare l'informazione "in chiaro" all'interno della matrice, il QR Code diventa un contenitore opaco, un semplice "biglietto" che, da solo, è incomprensibile. Ma qual è il vantaggio concreto di questa architettura?

1. Trasporto sicuro su canali insicuri: Il QR Code può essere esposto in una vetrina, inviato via e-mail o stampato su un biglietto da visita senza timore di esposizione. Chi lo intercetta vede solo un insieme apparentemente casuale di dati.
2. Decentralizzazione del controllo: A differenza dei sistemi "chiave in chiaro" o dei classici link di recupero password, il presente sistema separa il dato (la password) dal supporto (il QR) e dalla chiave di decifratura (conservata nel *backend*). L'utente deve possedere sia il

QR fisico che l'applicazione autenticata per risalire al dato originale.
3. Esperienza utente fluida: Per l'utente finale, il processo rimane semplice come scansionare un normale QR Code, ma il risultato è profondamente diverso: l'app non mostra un link, ma rivela un segreto.

In sintesi, questo progetto fonde la potenza visiva e la resilienza del QR Code con la robustezza della crittografia moderna, realizzando un sistema di "condivisione sicura di credenziali" dove la sicurezza non è un'opzione, ma un requisito architetturale di base. Il *backend*, custode delle chiavi, agisce come un notaio digitale che autorizza la decifrazione solo se l'app richiedente è legittima, creando un ecosistema chiuso dove l'informazione sensibile viaggia invisibile agli occhi di chiunque non sia autorizzato.

1. Fondamenti teorici

Il QR Code è un codice bidimensionale capace di memorizzare grandi quantità di informazioni rispetto ai tradizionali codici a barre lineari. Le sue principali caratteristiche sono l'elevata capacità di memorizzazione, la rapida scansione, la correzione automatica degli errori e l'ampia diffusione e compatibilità. Nonostante tali vantaggi, il QR Code standard non fornisce alcuna forma di protezione intrinseca dei dati. Per ovviare a tale limitazione, nel progetto QR Secure è stato adottato l'algoritmo AES-256 in modalità CBC (*Cipher Block Chaining*), considerato uno degli standard più sicuri e diffusi a livello mondiale. La crittografia simmetrica, su cui si basa AES-256, utilizza una singola chiave sia per la cifratura sia per la decifratura dei dati; in particolare, AES-256 impiega una chiave di 256 bit (32 byte), che offre una sicurezza computazionalmente impraticabile da violare con la tecnologia attuale. I vantaggi principali di questo algoritmo sono l'elevata sicurezza (256 bit = 2^{256} combinazioni possibili), la velocità di esecuzione ottimale su *hardware* moderno, l'ampia disponibilità di librerie certificate e il riconoscimento come standard NIST (National Institute of Standards and Technology). A ciò si aggiunge l'utilizzo di un Vettore di Inizializzazione (IV), un valore casuale di 128 bit che garantisce che due messaggi identici producano *ciphertext* differenti, impedendo così attacchi basati sull'analisi statistica dei dati cifrati [Wong D., 2021]. Nel sistema QR Secure, l'IV viene generato tramite un generatore crittografico sicuro ad ogni operazione di cifratura, assicurando l'unicità e l'imprevedibilità necessarie per la sicurezza del protocollo.

2. Architettura del sistema e tecnologie utilizzate

L'infrastruttura si compone di tre moduli distinti: un'applicazione mobile sviluppata in Delphi, che gestisce l'interfaccia utente e le operazioni di generazione e lettura dei QR Code nonché la cifratura e decifratura lato client; un *backend* REST API realizzato in Python con FastAPI, preposto alla gestione di autenticazione, autorizzazione e storage sicuro delle chiavi; e un database PostgreSQL, deputato a memorizzare chiavi cifrate, metadati dei QR Code, utenti e log di accesso. Ogni componente svolge un ruolo specifico all'interno del processo di cifratura e decifratura, garantendo la separazione dei livelli di sicurezza. Il flusso di generazione di un QR Code prende avvio quando l'utente inserisce il messaggio nell'app mobile; quest'ultima genera localmente una chiave AES-256 random di 32 byte e un IV di 16 byte, quindi cifra il messaggio con AES-256-CBC usando chiave e IV, e infine genera il QR Code contenente il *payload* cifrato e un *hash* identificativo. Contestualmente, l'app invia al *backend* l'*hash*, la chiave, l'IV, i parametri di visibilità e la scadenza; il *backend*, a sua volta, cifra chiave e IV con la Master Key AES-256-CBC (realizzando una doppia cifratura) e memorizza nel database l'*hash*, le chiavi cifrate e i

metadati. Per la lettura, invece, l'utente scansiona il QR Code con la fotocamera, l'app estrae il *payload* cifrato e l'*hash* identificativo, quindi richiede al *backend* le chiavi associate a quell'*hash*; il *backend* verifica l'autenticazione dell'utente, i permessi di accesso, la scadenza e i contatori di utilizzo e, se autorizzato, recupera le chiavi cifrate dal database, le decifra con la Master Key e le restituisce all'app su canale HTTPS, la quale infine decifra il *payload* del QR con la chiave ricevuta e mostra il messaggio decifrato all'utente. A tal proposito, è utile chiarire in modo esplicito quali informazioni siano trasportate dal QR Code e quali siano invece conservate lato *backend* e database, nonché le differenze tra la gestione dei messaggi testuali e quella delle immagini. Per i messaggi testuali, il QR Code è autosufficiente: contiene l'intero *payload* cifrato *(ciphertext)* insieme all'*hash* identificativo, secondo il formato descritto nel capitolo 3, ma non contiene mai la chiave AES né l'IV utilizzati per la cifratura. Per le immagini, invece, il *ciphertext* risultante dalla cifratura è tipicamente troppo voluminoso per essere codificato in un QR Code; in questo caso il QR Code trasporta esclusivamente l'*hash* identificativo, mentre il *payload* cifrato dell'immagine viene caricato sul *backend* tramite un *endpoint* dedicato e recuperato in lettura tramite l'*hash*. Per quanto riguarda il contenuto conservato nel *backend* e nel database, esso include l'*hash* identificativo del QR Code, la chiave AES e l'IV cifrati con la Master Key (doppia cifratura), i metadati di gestione (proprietario, visibilità pubblica o privata, tipo e valore del limite di utilizzo, numero di utilizzi effettuati, data di scadenza e data di creazione) e il log di *audit* degli accessi; per le immagini, il *backend* conserva inoltre il *payload* cifrato stesso, il tipo MIME e la dimensione originale del file. In caso di QR Code smarrito o sottratto, se il codice è classificato come privato, chi lo trova non può ottenere le chiavi di decifratura senza autenticarsi con le credenziali del proprietario, pertanto il solo possesso fisico del QR Code non è sufficiente a comprometterne il contenuto; se invece il QR Code è pubblico, chiunque sia autenticato sulla piattaforma può leggerlo, e la riservatezza dipende in questo caso dalla scelta di visibilità operata in fase di generazione, non dal possesso del codice. Se il QR Code è scaduto, superata la data di scadenza configurata, il *backend* rifiuta la richiesta di recupero delle chiavi restituendo un errore HTTP 410 Gone, rendendo il *payload* cifrato permanentemente indecifrabile tramite il normale flusso applicativo; analogamente, se il QR è configurato con un numero massimo di letture, il superamento della soglia produce lo stesso esito del QR scaduto (HTTP 410 Gone), indipendentemente dalla validità temporale residua. Infine, se il record nel *backend* viene eliminato o risulta corrotto, poiché le chiavi di decifratura non sono mai incluse nel QR Code ma risiedono esclusivamente, in forma cifrata, nel database del *backend*, il *payload* del QR Code diventa permanentemente indecifrabile, anche qualora il QR Code fisico o digitale sia ancora disponibile; questa conseguenza discende direttamente dalla scelta architetturale di separare dati cifrati (nel QR Code) e materiale crittografico (nel *backend*), e va tenuta in considerazione nella pianificazione di backup e retention del database. Da un punto di vista tecnologico, il *frontend* mobile è stato sviluppato con Delphi 13 RAD Studio, avvalendosi di librerie native per la generazione e scansione dei QR Code e di componenti crittografici per AES-256. Il *backend*, invece, è realizzato in Python 3 [Harrison M., 2017] con il *framework web* asincrono ad alte prestazioni FastAPI, e utilizza JWT (JSON Web Tokens) [Peyrott S. E., 2017] per l'autenticazione stateless, SQLAlchemy ORM per l'accesso al database, la libreria cryptography per AES-256-CBC e Passlib con bcrypt per l'*hashing* delle password. Per la persistenza, infine, è stato scelto PostgreSQL 13+, che garantisce affidabilità e conformità ACID, integrità referenziale e transazionale, elevate prestazioni con indici ottimizzati e robusti meccanismi di sicurezza e *auditing*.

3. Flusso operativo dell'applicazione

Questo capitolo illustra, attraverso una sequenza di schermate reali dell'applicazione, il flusso

operativo completo per la cifratura e la lettura di messaggi e immagini tramite QR Code.
All'avvio (Figura 1), l'applicazione presenta uno splash screen con il logo "QR Secure" che verifica automaticamente la connessione con il *backend* e lo stato del sistema.

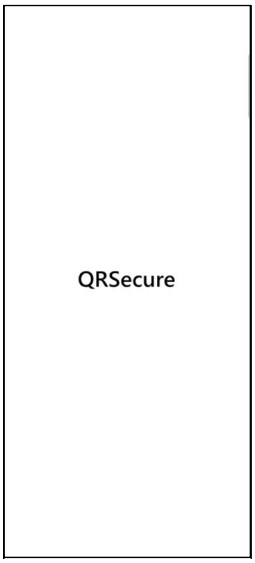


Figura 1 Splash screen iniziale dell'applicazione.
Figure 1 *Application splash screen.*

Durante questa fase vengono eseguite le seguenti operazioni:

- Verifica della raggiungibilità del server *backend* via HTTPS.
- Controllo della validità del certificato SSL/TLS.
- Inizializzazione dei componenti crittografici AES-256.
- Validazione dell'eventuale sessione utente esistente tramite JWT.

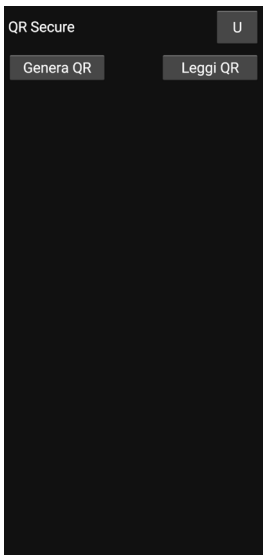


Figura 2 Schermata di verifica connessione completata.
Figure 2 *Connection verification screen.*

Una volta completate le verifiche con esito positivo, viene visualizzato il messaggio "Connessione verificata...", come mostrato in Figura 2, e l'applicazione procede alla schermata principale.
La Figura 3 mostra la schermata principale che presenta un'interfaccia essenziale e intuitiva con due pulsanti principali:

- Genera QR: per cifrare e generare nuovi QR Code.
- Leggi QR: per scansionare e decifrare QR Code esistenti.

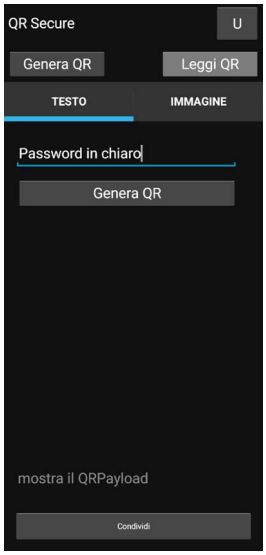
Figura 3 Schermata principale dell'applicazione.
Figure 3 *Application main screen.*



Nell'angolo superiore destro è presente l'icona utente (U) che consente l'accesso alle impostazioni del profilo, alla gestione account e alla funzione di logout.
La funzione di generazione permette di cifrare sia messaggi testuali che immagini. Selezionando "Genera QR" si accede a una schermata con due tab (Figura 4):

- TESTO: per cifrare messaggi testuali.
- IMMAGINE: per cifrare immagini dalla fotocamera o galleria.

Figura 4 Interfaccia per la generazione di QR Code con testo cifrato.
Figure 4 *Interface for generating QR Codes with encrypted text.*



a. Procedura per cifrare un messaggio testuale:
   - Selezionare il tab "TESTO".
   - Inserire il messaggio da cifrare nel campo di testo.
   - Premere il pulsante "Genera QR".

L'applicazione esegue automaticamente:

- Generazione chiave AES-256 (32 byte casuali).
- Generazione IV (16 byte casuali).
- Cifratura del messaggio con AES-256-CBC.
- Generazione *hash* SHA-256 identificativo.

- Invio chiavi al *backend* per doppia cifratura con Master Key.
- Creazione e visualizzazione del QR Code.



Figura 5 Messaggio di QR Code generato con successo.
Figure 5 *Successfully generated QR Code message.*

Il sistema mostra una finestra di dialogo con il messaggio "QR generato con successo" come conferma dell'operazione eseguita (Figura 5). In basso viene visualizzato l'*hash* identificativo univoco del QR Code generato. Il QR Code può essere condiviso, per esempio inviato alla propria e-mail per essere stampato.
La funzionalità di cifratura immagini consente di proteggere crittograficamente fotografie e immagini, trasformandole in QR Code che possono essere condivisi in modo sicuro. La Figura 6 mostra la sezione dell'app dedicata alla creazione di immagini cifrate.

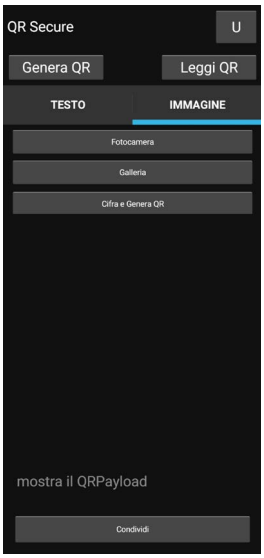


Figura 6 Finestra IMMAGINE con varie opzioni di selezione.
Figure 6 *IMAGE window with various selection options.*

b. Procedura per cifrare un'immagine:

- Selezionare il pulsante "IMMAGINE".

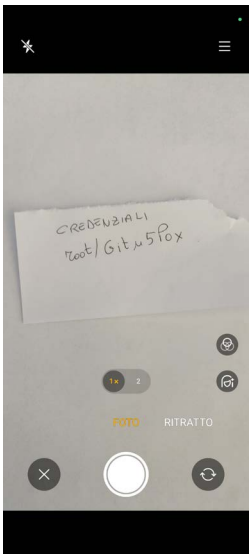
Scegliere la sorgente dell'immagine:

- Fotocamera: scatta una nuova foto.

- Galleria: seleziona un'immagine esistente.
- Cifra e Genera QR: cifra l'immagine selezionata.

Scegliendo di voler cifrare una immagine tramite la camera del dispositivo mobile si aprirà la camera e sarà possibile inquadrare l'immagine da cifrare (Figura 7).

Figura 7 Immagine catturata tramite la camera del cellulare.
Figure 7 *Image captured by the mobile phone camera.*



Al termine della fase di cattura immagine verrà mostrata una preview della stessa come mostra la Figura 8.

Figura 8 Preview dell'immagine.
Figure 8 *Image preview.*



Se l'immagine risulta quella desiderata si continua con il passo di generazione QR cifrato:

- Premere il pulsante "Cifra e Genera QR" per confermare la selezione.

L'applicazione esegue:

- Generazione chiave AES-256 e IV dedicati.
- Cifratura dell'immagine con AES-256-CBC.
- Upload immagine cifrata al *backend*.

- Doppia cifratura delle chiavi con Master Key server.
- Generazione QR Code con *hash* identificativo.

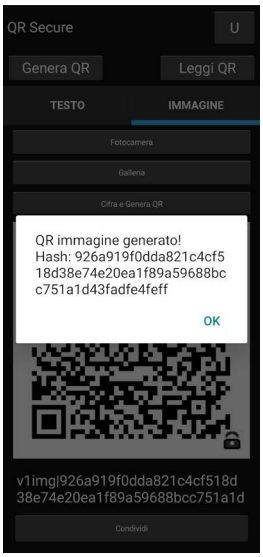


Figura 9 QR Code generato per l'immagine cifrata con hash identificativo.
Figure 9 *QR Code generated for the image encrypted with an identifier hash.*

Il QR Code risultante (Figura 9) contiene esclusivamente l'*hash* identificativo e il *payload* cifrato. L'immagine originale rimane protetta sul server, cifrata due volte (cifratura client + cifratura Master Key server).
La funzione di lettura consente di scansionare QR Code generati dall'applicazione e, previa autenticazione e verifica dei permessi, di decifrarne il contenuto. Il *payload* codificato in un QR Code testuale segue il formato v1|*hash*|*ciphertext*, dove i tre campi, separati dal carattere "|", hanno il seguente significato:

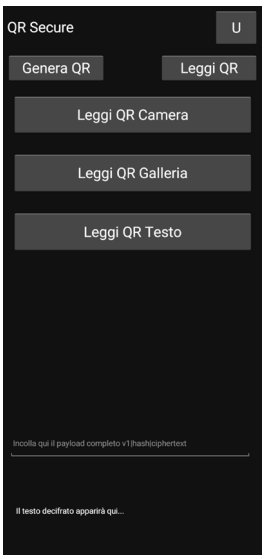
- v1 - identifica la versione del formato di *payload*. Il prefisso di versione consente future evoluzioni del protocollo (ad esempio un cambio di algoritmo o di struttura del *payload*) mantenendo la compatibilità con QR Code generati da versioni precedenti dell'applicazione, che possono essere riconosciuti e gestiti in modo differenziato in fase di lettura;
- *hash* - è l'identificativo univoco del QR Code, calcolato lato client al momento della generazione (SHA-256) a partire dal *ciphertext*. Questo *hash* è il valore con cui l'app interroga il *backend* (endpoint /qr/get-key) per recuperare la chiave e l'IV necessari alla decifratura; è anche la chiave primaria con cui il record è indicizzato nel database (tabella qr_keys);
- *ciphertext* - è il risultato della cifratura AES-256-CBC del messaggio originale, codificato in un formato testuale (es. Base64) compatibile con la capacità di un QR Code. Il *ciphertext* non è mai decifrabile senza la chiave e l'IV corrispondenti, che l'app ottiene dal *backend* solo dopo autenticazione e verifica dei permessi.

Differenza tra QR Code testuali e QR Code associati a immagini: il formato v1|*hash*|*ciphertext* descritto sopra si applica ai QR Code generati a partire da messaggi di testo, nei quali l'intero *ciphertext* è incluso nel codice. Per le immagini, per le ragioni dimensionali illustrate nel capitolo 2, il QR Code contiene tipicamente solo la coppia versione/*hash* (il campo *ciphertext* è assente o è sostituito da un riferimento), mentre il *payload* cifrato dell'immagine viene scambiato con il *backend* tramite gli endpoint dedicati di upload e download, identificato dallo stesso *hash*.

c. Opzioni di lettura disponibili come visibili in Figura 10:

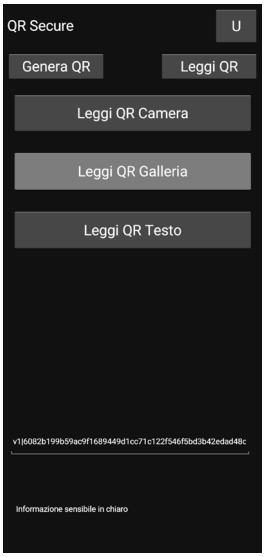
- Leggi QR Camera: scansione diretta tramite fotocamera del dispositivo.
- Leggi QR Galleria: selezione di un'immagine QR salvata nella galleria.
- Leggi QR Testo: inserimento manuale del *payload* complete: v1|*hash*|*ciphertext*.

Figura 10 Menu di lettura QR Code con tre opzioni disponibili.
Figure 10 *QR Code reader menu with three available options.*



d. Procedura di lettura QR Code da testo (Figura 11):
- Selezionare "Leggi QR Testo".
- Incollare o digitare il *payload* completo nel formato: v1|*hash*|*ciphertext*.

Figura 11 Campo per inserimento manuale del payload QR.
Figure 11 *Field for manually entering the QR Code payload.*



L'applicazione esegue automaticamente:
- Parsing del *payload* (estrazione versione, *hash*, dati cifrati).
- Richiesta HTTPS autenticata al *backend* con *hash*.
- Verifica permessi utente (pubblico/privato).
- Verifica scadenza temporale.
- Recupero chiave e IV decifrati dal *backend*.
- Decifratura locale del *payload* con AES-256-CBC.
- Visualizzazione del contenuto decifrato cliccando su "Leggi QR Testo".

Il sistema mostra una finestra di dialogo con il titolo “Testo decifrato” e con il testo originale (Figura 12). Questo conferma che l’intero processo di cifratura/decifratura e la gestione sicura delle chiavi tramite *backend* ha funzionato correttamente.

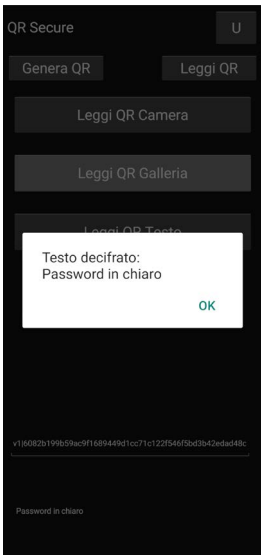


Figura 12 Messaggio decifrato visualizzato correttamente.
Figure 12 *Decrypted message displayed correctly.*

e. Procedura di lettura QR Code da Camera:
- Selezionare “Leggi QR Camera”.
- Fotografare il QR Code (Figura 13).

L’applicazione esegue automaticamente:
- Parsing del *payload* (estrazione versione, *hash*, dati cifrati).
- Richiesta HTTPS autenticata al *backend* con *hash*.
- Verifica permessi utente (pubblico/privato).
- Verifica scadenza temporale.
- Recupero chiave e IV decifrati dal backend.
- Decifratura locale del *payload* con AES-256-CBC (Figura 14).
- Visualizzazione del contenuto decifrato (Figura 15).



Figura 13 Momento in cui la camera sta inquadrando il QR Code cifrato.
Figure 13 *Moment when the camera is framing the encrypted QR Code.*



Figura 14 Processo di decodifica in corso.
Figure 14 *Decoding process in progress.*

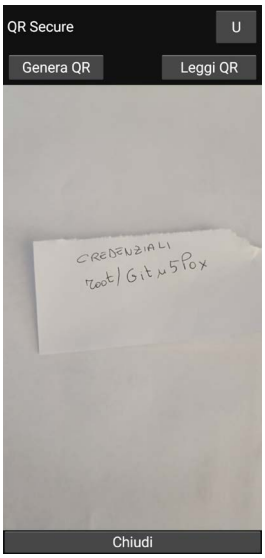


Figura 15 Immagine decodificata e mostrata a video.
Figure 15 *Image decoded and displayed on screen.*

Durante il processo di lettura, il sistema può generare diversi messaggi di errore in base alla situazione:

- QR Code non trovato: *hash* non presente nel database.
- Accesso negato: utente non autorizzato per QR privati.
- QR Code scaduto: validità temporale terminata.
- Limite utilizzi raggiunto: numero massimo di letture esaurite.
- Errore di decifratura: *payload* corrotto o chiavi non valide.
- Tutti i tentativi di accesso, sia autorizzati che negati, vengono registrati nel log di *audit* del *backend* con timestamp UTC, ID utente, *hash* QR, tipo operazione ed esito.

4. Funzionalità dell'applicazione

a. Splash Screen e Verifica Connettività

Come illustrato nel capitolo 3, all'avvio l'applicazione presenta una schermata iniziale che verifica la connessione con il *backend* e lo stato del sistema. Questa fase consente di:

- verificare la raggiungibilità del server *backend*.

- controllare la validità del certificato SSL/TLS.
- inizializzare i componenti crittografici.
- validare la sessione utente esistente.

b. Generazione QR Code Cifrato
L'utente può inserire un messaggio o selezionare un'immagine che verrà automaticamente:

- Cifrato localmente con AES-256-CBC.
- Trasformato in QR Code visualizzabile e salvabile.
- Associato ad un identificativo univoco (*hash* SHA-256).
- Registrato sul *backend* con doppia cifratura delle chiavi.

c. Lettura QR Code Cifrato
Durante la scansione, il sistema esegue automaticamente:

- Acquisizione del QR tramite fotocamera, galleria o input manuale.
- Estrazione del *payload* cifrato e dell'identificativo.
- Richiesta autenticata al *backend* per le chiavi di decifratura.
- Verifica autorizzazioni e validità temporale.
- Decifratura locale del contenuto.
- Visualizzazione del messaggio o immagine originale.

d. Gestione Multi-Utente
La piattaforma supporta la registrazione e la gestione di un numero arbitrario di utenti, consentendo l'utilizzo del sistema in contesti aziendali, organizzativi o multiutente.
Ogni utente dispone di credenziali personali per l'accesso all'applicazione e ai servizi *backend*. Per garantire elevati standard di sicurezza, al primo accesso ogni nuovo utente è obbligato a modificare la password inizialmente assegnata dall'amministratore.
Tale procedura consente di:

- Eliminare l'utilizzo di password temporanee o predefinite.
- Assicurare la riservatezza delle credenziali.
- Ridurre il rischio di accessi non autorizzati.
- Le password vengono memorizzate esclusivamente in forma *hashata* utilizzando bcrypt, uno degli algoritmi crittografici più sicuri per lo *hashing* di password.

e. Classificazione e Controllo Accessi
Da un punto di vista operativo, la distinzione non riguarda la visibilità del QR Code in sé (che, come codice a barre bidimensionale, è sempre leggibile da chiunque possieda una fotocamera), bensì l'autorizzazione a ottenere dal *backend* la chiave e l'IV necessari a decifrarne il contenuto. "Pubblico" e "privato" sono quindi attributi del record associato all'*hash* nel database, verificati dal *backend* ad ogni richiesta di lettura (endpoint /qr/get-key), e non proprietà crittografiche del QR Code stesso.
Un QR Code pubblico può essere decifrato da qualunque utente autenticato e registrato sulla piattaforma, indipendentemente dal fatto che ne sia o meno il proprietario: la protezione riguarda l'accesso non autenticato (chi non ha un account non può comunque ottenere le chiavi), ma non la riservatezza tra utenti registrati. Un QR Code privato, invece, è decifrabile esclusivamente dal proprietario: il *backend* verifica che l'identificativo dell'utente autenticato coincida con il proprietario del QR Code prima di restituire chiave e IV; nessun altro utente, anche se registrato sulla piattaforma, può ottenerne la decifratura.
Il sistema distingue due differenti modalità di accesso ai QR Code cifrati.
I QR Code classificati come pubblici possono essere letti da qualsiasi utente autenticato e regolarmente registrato sulla piattaforma. Questa modalità risulta particolarmente utile per:

- Documentazione condivisa aziendale.
- Informazioni comuni a gruppi di lavoro.
- Procedure operative standard.
- Comunicazioni destinate a gruppi estesi di utenti.

I QR Code classificati come privati possono essere letti esclusivamente dall'utente che li ha generati. Il *backend* verifica l'identità del richiedente prima di autorizzare il recupero delle chiavi crittografiche necessarie alla decifratura.
Questa modalità garantisce la massima riservatezza per:

- Dati personali sensibili.
- Credenziali di accesso.
- Informazioni riservate e confidenziali.
- Documentazione soggetta a vincoli di riservatezza.

f. Gestione della Validità Temporale
Ogni QR Code può essere associato a un intervallo di validità configurabile al momento della generazione. La durata selezionabile varia da 24 ore fino a 10 anni.
Alla scadenza del periodo impostato:

- Il *backend* nega automaticamente il recupero delle chiavi di decifratura.
- Il contenuto non risulta più accessibile.
- Il QR Code viene considerato automaticamente scaduto.
- Viene registrato un log di accesso negato per *audit* trail.

Questa funzionalità consente di implementare scenari quali:

- Credenziali temporanee per accessi limitati.
- Autorizzazioni a termine per collaboratori esterni.
- Accessi limitati nel tempo a informazioni sensibili.

5. Sicurezza implementata

Le affermazioni sul livello di sicurezza offerto da QR Secure, richiamate in questo capitolo, sono da intendersi nel contesto del seguente modello di minaccia, che esplicita gli scenari di attacco considerati, i componenti ritenuti fidati e le conseguenze della compromissione di ciascun componente.

a. Componenti e fiducia

- Dispositivo mobile (client Delphi): considerato parzialmente fidato. Genera chiave e IV e cifra i dati prima della trasmissione, ma un dispositivo compromesso (malware, root/jailbreak, debug attivo) può esporre in chiaro i dati durante l'elaborazione, prima della cifratura o dopo la decifratura.
- Canale di trasmissione: protetto tramite HTTPS/TLS; si assume l'integrità del certificato del server, verificata all'avvio.
- *Backend* (Python FastAPI): considerato fidato per l'esecuzione della logica di autorizzazione e per la detenzione della Master Key, ma non per la protezione dei dati residenti nel database in caso di compromissione dell'host stesso.
- Database PostgreSQL: considerato potenzialmente esposto (ad esempio in caso di dump non autorizzato, backup non protetti o accesso diretto da parte di un amministratore malevolo); per questo motivo chiave e IV non vi sono mai memorizzati in chiaro.
- Master Key: componente critico, la cui compromissione vanifica la protezione

crittografica delle chiavi memorizzate nel database.

b. Scenari di attacco considerati

- Sottrazione o smarrimento del solo QR Code (fisico o digitale): il *ciphertext* (o, per le immagini, il solo *hash*) non è sufficiente a ricavare il contenuto in chiaro senza le chiavi custodite dal *backend*.
- Accesso non autorizzato al database (es. dump, backup rubato): le chiavi ivi memorizzate sono cifrate con la Master Key; senza quest'ultima, non sono utilizzabili.
- Compromissione del server applicativo senza esfiltrazione del database: un attaccante con accesso al processo *backend* in esecuzione potrebbe, in linea di principio, intercettare la Master Key in memoria o le chiavi in transito verso il client; questo scenario non è completamente mitigato dall'architettura attuale e rappresenta un limite del modello.
- Compromissione del dispositivo client: un dispositivo con malware attivo durante l'uso dell'app può accedere ai dati in chiaro nel momento in cui vengono cifrati o decifrati localmente; questo scenario esula dal perimetro di protezione offerto dal *backend*.

c. Conseguenze della compromissione della Master Key

La Master Key è il singolo punto di fallimento più critico del sistema: la sua compromissione consente di decifrare tutte le chiavi e gli IV memorizzati nel database e, di conseguenza, tutti i *payload* dei QR Code per i quali un attaccante disponga anche del *ciphertext* (contenuto nel QR Code stesso o, per le immagini, caricato sul *backend*). Per questo motivo la Master Key non è mai conservata nel database né nel codice sorgente, ma risiede esclusivamente in variabili d'ambiente del server; resta comunque affidata alla configurazione operativa del sistema (permessi del file .env, accesso amministrativo al server) e non a un meccanismo di custodia hardware dedicato, aspetto che potrebbe rientrare nelle evoluzioni future.

d. Principi Architetturali di Sicurezza

Il sistema QR Secure adotta i seguenti principi fondamentali:

- Cifratura end-to-end: i dati sensibili sono cifrati sul dispositivo prima della trasmissione.
- Doppia cifratura delle chiavi: le chiavi di cifratura utente vengono a loro volta cifrate con Master Key sul server.
- Separazione tra dati cifrati e materiale crittografico: il QR Code non contiene mai le chiavi.
- Autenticazione forte: JWT con refresh token e scadenza automatica.
- *Audit* trail completo: ogni accesso viene registrato con timestamp, utente, esito.

e. Cifratura Multi-Livello

Il sistema implementa una doppia cifratura per garantire la massima sicurezza:

- I dati utente vengono cifrati localmente sul dispositivo mobile utilizzando.
- Algoritmo: AES-256-CBC (Advanced Encryption Standard, 256 bit).
- Chiave: 32 byte (256 bit) generati tramite CSPRNG.
  - IV: 16 byte (128 bit) generati casualmente per ogni cifratura.
  - Padding: PKCS#7 standard.

Le chiavi di cifratura utente e gli IV vengono a loro volta cifrati prima della memorizzazione nel database utilizzando:

- Algoritmo: AES-256-CBC.
- Master Key: chiave di 256 bit residente esclusivamente sul server.
- IV univoco per ogni operazione di cifratura.

- Storage: la Master Key non è mai memorizzata nel database, solo in variabili d'ambiente del server.

f. Vantaggi dell'approccio a doppia cifratura:
- La compromissione del QR Code non è sufficiente: senza le chiavi dal server, il contenuto resta cifrato.
- La compromissione del database non è sufficiente: le chiavi sono cifrate con la Master Key.
- La compromissione del solo server richiede anche il possesso del QR Code cifrato.
- Protezione multi-livello: un attaccante deve compromettere simultaneamente più componenti.

g. Modello di Controllo degli Accessi.
Il *backend* applica rigorose politiche di autorizzazione durante la fase di lettura. Prima di restituire le chiavi di decifratura, il sistema verifica:
- Autenticazione dell'utente tramite JWT valido e non scaduto.
- Stato dell'account utente (attivo, verificato).
- Tipologia del QR Code (pubblico o privato).
- Eventuale scadenza temporale del QR Code.
- Contatori di utilizzo e limiti configurati.

Questo approccio permette di combinare protezione dei dati, flessibilità operativa e sicurezza, tramite la distinzione tra visibilità pubblica e privata, la gestione delle scadenze e dei limiti di utilizzo. La distinzione tra QR Code pubblici e privati rende la piattaforma adatta sia all'uso personale sia agli ambienti collaborativi aziendali.

h. *Auditing* e Tracciabilità
Il sistema mantiene un *audit* log completo di tutte le operazioni critiche:
- Ogni tentativo di accesso (autorizzato o negato) viene registrato con timestamp UTC.
- Vengono memorizzati: ID utente, *hash* QR Code, tipo operazione, esito, motivo del rifiuto.

6. Casi d'uso

L'applicazione può trovare impiego in numerosi scenari in cui è necessario proteggere informazioni sensibili o controllare l'accesso a risorse fisiche e digitali. La combinazione tra QR Code, crittografia e validazione tramite *backend* consente di realizzare soluzioni sicure, pratiche e facilmente utilizzabili. Di seguito vengono illustrati alcuni casi d'uso particolarmente significativi.
Uno degli utilizzi più semplici ed immediati consiste nella memorizzazione di password complesse. Invece di annotare la password in chiaro su un foglio o salvarla in un file non protetto, l'utente può generare un QR Code cifrato contenente la credenziale. Il codice può essere stampato e posizionato in prossimità della propria postazione di lavoro.
Funzionamento (Figura 16):
- L'utente inserisce la password nell'applicazione.
- L'app genera il QR Code cifrato.
- Il QR Code viene stampato o visualizzato su un supporto fisico.
- Quando necessario, l'utente utilizza l'app mobile per decifrare il contenuto.

Vantaggi:
- La password non è leggibile ad occhio nudo.
- Solo gli utenti autorizzati possono decifrarla.

- Riduzione del rischio di accessi non autorizzati.
- Maggiore praticità rispetto alla gestione manuale delle credenziali.

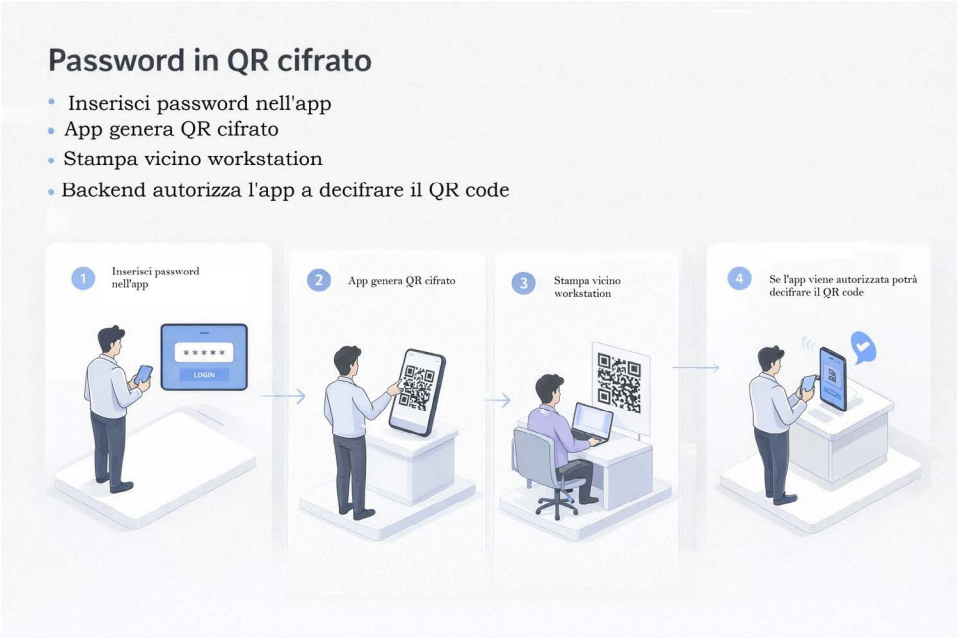


Figura 16 Come conservare una password con QR Secure (immagine generata con IA).
Figure 16 *How to store a password with QR Secure (AI generated image).*

È utile confrontare questo caso d›uso con le soluzioni già ampiamente adottate per la gestione delle password, in modo da chiarirne vantaggi e limiti specifici. I password manager (locali o basati su cloud, es. con cifratura end-to-end e sblocco tramite master password o biometria) offrono generazione automatica, compilazione integrata nei form di login e sincronizzazione multi-dispositivo: funzionalità che QR Secure non replica, poiché il suo QR Code cifrato è pensato per la conservazione e condivisione occasionale di una singola credenziale, non per la gestione quotidiana di un intero portafoglio di password.
Rispetto all'annotazione in chiaro su carta o alla conservazione in una cassaforte fisica, il vantaggio di QR Secure è la protezione crittografica del contenuto anche in caso di smarrimento del supporto stampato: chi trova il QR Code non può leggerne il contenuto senza accedere anche alle chiavi custodite dal *backend*. Rispetto a un password manager con cifratura end-to-end, QR Secure introduce invece una dipendenza aggiuntiva dalla disponibilità del *backend* e dalla rete al momento della decifratura: se il servizio non è raggiungibile, la password stampata non può essere letta, mentre un password manager offline resta accessibile localmente.
Il caso d'uso è quindi più appropriato per scenari specifici, ad esempio la trasmissione una tantum di una credenziale a un collaboratore, o la sua conservazione fisica come backup di emergenza, piuttosto che come sostituto di un password manager per l'uso quotidiano.
Un'altra applicazione particolarmente interessante riguarda il controllo degli accessi ad aree riservate, sale server, laboratori o locali tecnici. In questo scenario, un display installato accanto alla porta visualizza un QR Code temporaneo generato dal *backend*. Il codice contiene una password o un token cifrato valido solo per un intervallo di tempo limitato.
Sequenza Operativa (Figura 16):

- Il *backend* genera dinamicamente un QR Code temporaneo.
- Il display installato accanto alla porta lo visualizza.
- L'utente autorizzato utilizza l'app mobile per inquadrare il codice.

- L'app decifra il contenuto.
- Il contenuto viene nuovamente cifrato e inviato al *backend* per la verifica.
- Il *backend* valida l'identità e autorizza l'accesso.
- Il sistema di controllo disattiva temporaneamente la serratura elettronica.
- Il display mostra un indicatore verde di accesso consentito.

Vantaggi:

- Eliminazione di badge fisici o chiavi tradizionali.
- Utilizzo di credenziali temporanee e dinamiche.
- Elevata sicurezza grazie alla doppia cifratura.
- Tracciabilità completa di ogni accesso.
- Possibilità di impostare scadenza e numero massimo di utilizzi per ciascuna credenziale.



Figura 17 Come controllare un accesso tramite QR Secure (immagine generata con IA).
Figure 17 *How to control access with QR Secure (AI generated image).*

La realizzazione di questo scenario richiede alcune estensioni rispetto alla configurazione base della piattaforma. Gli interventi necessari sono limitati e di facile implementazione, in quanto di natura modulare e senza impatto sull'architettura esistente.
Lato *backend* è necessario predisporre la logica di generazione del QR Code temporaneo, con token cifrato e scadenza configurabile, nonché un endpoint dedicato alla verifica del token e all'autorizzazione dell'accesso. Il *backend* dovrà inoltre gestire l'interfacciamento con il sistema di controllo della serratura elettronica.
Riguardo l'app mobile abbiamo che questa è già predisposta per cifrare e inviare, ma richiede l'aggiunta di una sezione per gestire questo flusso specifico.

7. Conclusioni ed evoluzioni future

QR Secure dimostra come sia possibile coniugare semplicità d'uso, portabilità e robustezza crittografica in un'unica piattaforma destinata alla protezione e alla condivisione sicura delle informazioni. L'integrazione tra applicazione mobile, *backend* centralizzato e database relazionale ha permesso di realizzare un ecosistema affidabile, scalabile e adatto sia all'utilizzo

individuale sia agli scenari aziendali multiutente. L'approccio adottato, basato sulla doppia cifratura (AES-256-CBC lato client + AES-256-CBC con Master Key lato server) e sulla separazione fisica tra dati cifrati e materiale crittografico, costituisce uno dei principali punti di forza del sistema, garantendo elevati livelli di sicurezza anche in presenza di compromissione parziale dell'infrastruttura. La flessibilità introdotta attraverso la distinzione tra QR pubblici e privati, unita alla gestione della validità temporale e dei controlli di accesso granulari, rende la piattaforma estremamente versatile e adatta a molteplici contesti applicativi: dalla protezione di credenziali personali alla condivisione sicura di documenti aziendali. Come illustrato nel capitolo 4, l'interfaccia utente risulta intuitiva e permette agli utenti di cifrare e decifrare contenuti testuali e multimediali con pochi semplici passaggi, rendendo la crittografia avanzata accessibile anche a utenti non tecnici.

Tuttavia, il panorama della sicurezza informatica è in continua evoluzione. Gli algoritmi oggi considerati sicuri potrebbero, nel prossimo futuro, essere messi in discussione dall'affermazione del calcolo quantistico su larga scala. La ricerca nel campo della crittografia post-quantum è già oggi uno dei settori più dinamici e strategici dell'intera informatica moderna. Quando la crittografia quantistica e gli algoritmi post-quantum diventeranno strumenti di uso comune, sarà probabilmente necessario ripensare radicalmente anche sistemi come QR Secure, progettando nuove soluzioni ancora più robuste, efficienti e resilienti contro minacce computazionali oggi inimmaginabili. Forse, in quel momento, questa applicazione avrà concluso il proprio compito, lasciando spazio a una nuova generazione di strumenti capaci di affrontare le sfide della sicurezza del futuro.

In attesa di tali sviluppi, l'architettura di QR Secure è stata comunque progettata secondo criteri di modularità, scalabilità ed estendibilità, consentendo l'introduzione di nuove funzionalità senza modificare radicalmente l'infrastruttura esistente. Tra le principali evoluzioni previste si segnalano l'integrazione con sistemi di autenticazione multifattore (MFA/2FA), il supporto a firme digitali dei contenuti (PKI), la gestione di gruppi e ruoli avanzati con gerarchie, la revoca immediata dei QR Code anche già distribuiti, le notifiche push per eventi critici di sicurezza, la sincronizzazione offline con aggiornamento differito, l'*auditing* avanzato con dashboard di analisi e l'integrazione con sistemi aziendali tramite API REST documentate.

Bibliografia

Buchanan, W. (1998). *Mastering pascal and delphi programming*. Macmillan Education UK. https://doi.org/10.1007/978-1-349-14770-0

Harrison, M. *Illustrated Guide to Python 3: A Complete Walkthrough of Beginning Python with Unique Illustrations Showing How Python Really Works*. hairysun.com, 2. Treading on Python Series.

Peyrott, S. E. (2017). *The JWT handbook. Auth0*.

Wong, D. (2021). *Real-world cryptography*. Manning Publications Co.

APPENDICE

I passi seguenti hanno natura puramente procedurale (richiesta e installazione dell'APK, creazione dell'account, primo accesso) e sono riportati in appendice in quanto non necessari alla comprensione dell'architettura, del modello dati o del modello di sicurezza descritti nei capitoli precedenti.
Per poter utilizzare la piattaforma è necessario seguire una semplice procedura di attivazione iniziale che prevede l'installazione dell'applicazione e la creazione delle credenziali di accesso.
L'applicazione, al momento, non è distribuita tramite store ufficiali, pertanto, deve essere richiesta direttamente al referente del sistema.
Inviare una richiesta via e-mail all'autore del manoscritto.
Il referente, convalidata la richiesta, provvederà a fornire il file **APK** necessario per l'installazione.

Installazione dell'applicazione
Una volta ricevuto il file APK:

- Copiare il file sul dispositivo Android.
- Abilitare, se necessario, l'installazione da "Origini sconosciute".
- Avviare l'installazione del file APK.
- Completare la procedura guidata.

L'accesso alla piattaforma è consentito solo agli utenti registrati.
Contestualmente all'invio del file APK verranno inviate anche le credenziali di accesso alla piattaforma.
Il referente fornirà:

- Username.
- Password iniziale (o procedura di attivazione).

Dopo l'installazione:

- Aprire l'applicazione.
- Inserire le credenziali ricevute.
- Effettuare il login.

Al primo accesso è richiesto il cambio della password.
Note finali:

- L'accesso alla piattaforma è riservato agli utenti autorizzati.
- Le credenziali sono personali e non devono essere condivise.
- In caso di problemi di accesso, contattare lo stesso indirizzo e-mail utilizzato per la richiesta iniziale.

QUADERNI di GEOFISICA

ISSN 1590-2595 https://collane-editoriali.ingv.it/index.php/qdg/it/index/

QUADERNI di GEOFISICA è una rivista scientifica che pubblica dati, misurazioni e osservazioni geofisiche (anche preliminari) in italiano e inglese. L'obiettivo è una diffusione rapida e gratuita per la comunità scientifica, garantita da un processo di peer-review e da un Editorial Board multidisciplinare. La rivista è inclusa in "Emerging Sources Citation Index" di Clarivate Analytics e "Open Access Journals" di Scopus, e adotta il modello Diamond Open Access, che non prevede costi né per gli autori né per i lettori.

QUADERNI di GEOFISICA is a scientific journal that publishes geophysical data, measurements and observations, including preliminary results, in Italian and English. Its aim is to ensure rapid and free dissemination to the research community, supported by a peer-review process and by a multidisciplinary Editorial Board. The journal is included in the Emerging Sources Citation Index of Clarivate Analytics and it's included in the Open Access journals of Scopus. It follows the Diamond Open Access model, with no charges for either authors or readers.

RAPPORTI TECNICI INGV

ISSN 2039-7941 https://collane-editoriali.ingv.it/index.php/rt/it/index

RAPPORTI TECNICI INGV è una rivista tecnico-scientifica che mette in mostra le numerose attività tecnologiche e di ricerca condotte dai ricercatori e tecnici INGV, e non solo. Pubblicata in italiano e inglese, RAPPORTI TECNICI INGV è ideata per valorizzare anche quei lavori che offrono un contributo significativo dal punto di vista tecnico, operativo o documentale fondamentale per lo sviluppo dell'Ente. La rivista adotta il modello Diamond Open Access, che non prevede costi né per gli autori né per i lettori.

RAPPORTI TECNICI INGV is a technical–scientific journal that showcases the wide range of technological and research activities carried out by INGV researchers and technical staff, as well as by external contributors. Published in Italian and English, RAPPORTI TECNICI INGV is designed to highlight works that provide a significant contribution from a technical, operational, or documentary perspective, and that are essential to INGV development. The journal follows the Diamond Open Access model, with no charges for either authors or readers.

MISCELLANEA INGV

ISSN 2039-6651 https://collane-editoriali.ingv.it/index.php/mi/it/index

MISCELLANEA INGV rappresenta una tipologia di editoria non tradizionalmente catalogabile nelle ordinarie riviste scientifiche di settore. Pubblicata in lingua italiana e inglese, MISCELLANEA INGV offre uno spazio per la pubblicazione di materiali eterogenei che, pur non rientrando nei formati più strutturati delle altre Collane editoriali INGV, rivestono un interesse scientifico, tecnico o divulgativo. La rivista adotta il modello Diamond Open Access, che non prevede costi né per gli autori né per i lettori.

MISCELLANEA INGV represents a form of publishing that cannot be classified within the standard scientific journals. Published in Italian and English, MISCELLANEA INGV provides a venue for the publication of heterogeneous materials that are of scientific, technical or outreach interest, even though they do not fall within the more structured formats of the other INGV editorial series. The journal follows the Diamond Open Access model, with no charges for either authors or readers.

Progetto grafico
Barbara ANGIONI
Istituto Nazionale di Geofisica e Vulcanologia

Ufficio Editoriale
Francesca DI STEFANO | Coordinatore

Segreteria di redazione
segreteria.collane-editoriali@ingv.it

Editing | Produzione | Grafica redazionale
Barbara ANGIONI
Massimiliano CASCONE
Francesca DI STEFANO
Patrizia PANTANI

© 2026 the Author(s). All rights reserved. Open Access

Creative Commons Attribution 4.0 International


Display mostra
QR temporaneo
App
scansiona
e decifra
Ricifra e invia
al backend
Backend verifica
e sblocca porta