


# ZERO KNOWLEDGE PASSWORDLESS LOGIN SYSTEM



autenticazione senza password  
usando Zero-Knowledge Proof

## Obiettivo:

Mostrare il funzionamento  
attraverso una **demo** di Zero  
Knowledge Passwordless tra  
**Attori del sistema** e con  
**possibili attacchi**

Realizzato da :  
Andrea Mattioli

---

# COS'È UNA ZKP? (ZERO-KNOWLEDGE PROOF)

Una ZKP permette al **Prover** di convincere il **Verifier** che conosce un segreto **senza rivelarlo né inviare la password**.

Nel nostro progetto usiamo lo **schema di identificazione di Schnorr** (gruppi modulari).

## Setup (una tantum)

- Scegliamo parametri globali: primo sicuro  $p$ , ordine  $q$  con  $q \mid (p-1)$ , generatore  $g$  di ordine  $q$ .
- L'utente ha una chiave segreta  $x$  (derivata dalla password, es.  $x = H(\text{password} \parallel \text{user\_id}) \bmod q$ ).
- Chiave pubblica:  $y = g^x \bmod p$ .

## Protocollo (login)

1. **Impegno:** Prover sceglie  $r \leftarrow \{1, \dots, q-1\}$  e invia  $t = g^r \bmod p$ .
  2. **Sfida:** Verifier invia una sfida casuale  $c \leftarrow \{1, \dots, q-1\}$ .
  3. **Risposta:** Prover calcola  $s = r + c \cdot x \pmod{q}$  e invia  $s$ .
  4. **Verifica:** Verifier accetta se  $g^s \equiv t \cdot y^c \pmod{p}$ .
-

---

# COS'È UNA ZKP? (ZERO-KNOWLEDGE PROOF)

Perché è “zero-knowledge” ?

- Victor vede solo (**t = commitment** , **c = challenge**, **s = response**) e verifica la relazione; **non** può ricavare **x** (presuppone la difficoltà del logaritmo discreto).

**Nota importante (niente cifratura)**

- **ZKP ≠ cifratura**: non “crittiamo/decrittiamo” la password. La password serve solo a **derivare x**; la prova usa sole esponenziazioni modulari.

**Variante non interattiva (Fiat–Shamir)**

Opzionale: si può sostituire la sfida con **c = H(t || user\_id || nonce)** per eliminare il round-trip, mantenendo la stessa verifica (usato in blockchain e sistemi distribuiti, dove non c'è un verificatore dedicato).

---

---

# COS'È UNA ZKP? (ZERO-KNOWLEDGE PROOF)

## Mini-esempio

$$p = 23, q = 11, g = 2, x = 4 \Rightarrow y = g^x = 16 \bmod 23$$

$$\text{Peggy: } r = 7 \Rightarrow t = g^r = 13 \bmod 23$$

$$\text{Victor: } c = 5$$

$$\text{Peggy: } s = r + c \cdot x \bmod q = 7 + 5 \cdot 4 = 27 \equiv 5 \bmod 11$$

$$\text{Verifica: } g^s = 2^5 = 32 \equiv 9, t \cdot y^c = 13 \cdot 16^5 \equiv 9 \pmod{23} \Rightarrow \text{accetta.}$$

---

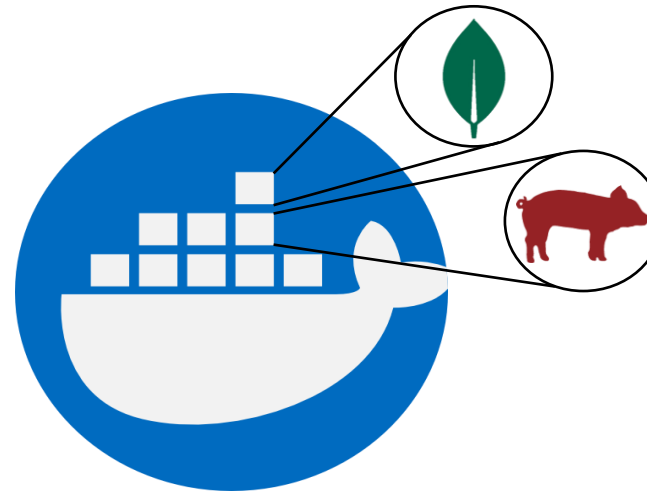
---

# TECNOLOGIE E STRUMENTI

- Python 3.9+
  - Librerie create:
    - zkp\_core.py
    - crypto\_utils.py
  - Librerie usate:
    - Streamlit
    - JWT tokens



- Docker
  - MongoDB
  - MailHog



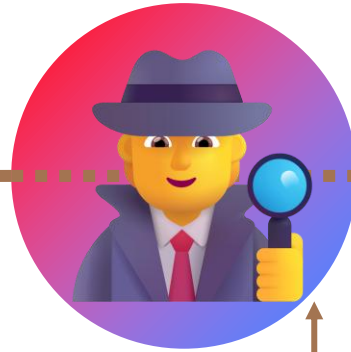
---

# RUOLI DEGLI ATTORI

- Attori Principali



Peggy (Provider)  
Utente



Anthony  
( Proxy /  
Attaccante )



Victor  
(Verifier)

- Attori Secondari



Partner di Anthony  
(Emulatore)



Malia  
(Mail Provider Utente)

---

**CONTINUIAMO CON L'ESECUZIONE DEL CODICE...**

---