Secure Identity Management System (SGI)

Julieta Lizeth Carrillo Hernandez
Cybersecurity Engineering (3rd Trimester)
Polytechnic University of Yucatan (UPY)

Course: Identity and Access Control – Professor: Ernesto López

Mérida, Mexico – 2409040@upy.edu.mx

September 10, 2025

Collaborators
Jorge Enrique Vargas Pech
Data and AI Engineering (3rd Trimester)
Polytechnic University Of Yucatan (UPY)
Mérida, México - 2409244@upy.edu.mx:

Resumen

A secure identity management system is implemented using Python, designed to ensure the integrity, confidentiality, and auditability of user credentials and personal data. The architecture combines strong password hashing (bcrypt), encrypted storage (Fernet/AES), time-based one-time password (TOTP) multifactor authentication, and RSA digital signatures to detect and prevent tampering. All sensitive actions are recorded in encrypted audit logs. The command line user interface supports English and Spanish and operates over SSH for remote or cloud deployments without a graphical environment. Testing demonstrates reliable detection of data modification, strict enforcement of access controls, and robust logging of both successful and failed events. The results show that advanced security and compliance measures can be practically integrated,even in a CLI-only, open source solution.

I. Introduction

Identity is the new perimeter. As organizations continue to embrace digital transformation, the traditional network centric security model is becoming obsolete. Instead, the focus has shifted to protecting identity components like passwords, personal data, and credentials. A robust and secure identity management framework is no longer a luxury; it's a critical foundation for modern cybersecurity.

The threat landscape is more complex and dynamic than ever. Attackers are relentlessly targeting identities because they represent the most direct path to an organization's most valuable assets. Compromised identities can lead to a cascade of devastating consequences, including:

- **Data Breaches:** Unauthorized access to sensitive customer, employee, or corporate data.
- **Financial Loss:** Direct theft of funds, fraudulent transactions, or costs associated with a security incident response.
- **Reputational Damage:** Loss of customer trust and brand credibility, which can take years to rebuild.
- **Operational Disruption:** Business processes can be halted or manipulated, leading to significant productivity loss.

The sheer volume and sophistication of modern attacks, from phishing and credential stuffing to advanced persistent threats, make protecting the integrity of every identity component a top priority.

The primary objective of most cyberattacks is to exploit human weaknesses and technical vulnerabilities to gain unauthorized access. Common attack vectors include:

- **Phishing and Social Engineering** Tricking employees into revealing credentials or sensitive information.
- **Credential Stuffing:** Using lists of stolen usernames and passwords from a data breach to gain access to other accounts.
- **Malware and Keyloggers:** Software designed to record keystrokes and capture login information.
- **Insider Threats:** Malicious or accidental actions by current or former employees who have legitimate access to systems.

Given this reality, organizations must move beyond simple password policies. The integrity of every identity component, from the moment an account is created to its eventual deactivation, it must be protected through a multi layered security approach.

II. Architecture and Design

II-A. Overview

The *Secure Identity Management System (SGI)* is designed to securely manage digital identities within an enterprise environment. Digital identities include user credentials, personal identifying information (PII), and access rights. The key goal is to ensure integrity, confidentiality, and auditability across all components involved in user management and authentication.

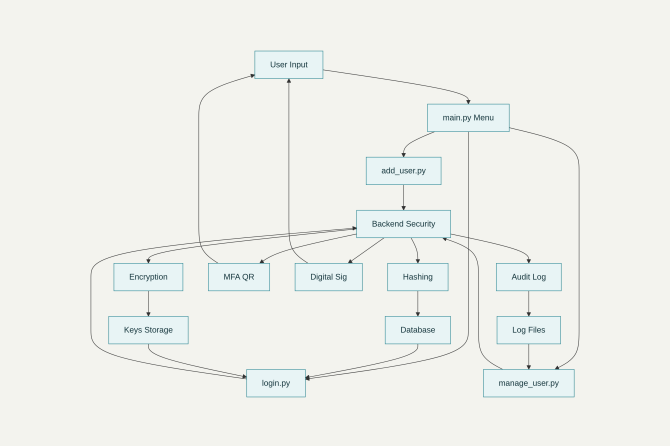
II-B. Core components

1. **Identity Creation and Lifecycle Management**: Users in the system are represented by unique digital identities associated with a set of attributes: username, hashed password, personal data, and role based access privileges. The system supports the full lifecycle of digital identities: creation, modification, verification, and eventual deactivation or deletion, adhering to principles commonly outlined in Identity and Access Management (IAM) frameworks.
2. **Authentication Mechanism:** Authentication is implemented through multiple layers:
   - **Password hashing:** Passwords are never stored in plain text. Instead, the system uses bcrypt, a cryptographically strong hash function that applies salt and iterative rounds to resist brute force and rainbow table attacks.
   - **Multi-factor authentication (MFA):** The system requires time-based one-time passwords (TOTP), compatible with widely used authenticator apps like Google Authenticator. MFA significantly strengthens assurance of user identity.
3. **Data Integrity and Encryption:** All sensitive user data is encrypted at rest using symmetric encryption (Fernet, an implementation of AES-128 with authenticated encryption). Each user record stores:
   - A SHA-256 cryptographic hash of the user's personal data.
   - A digital signature of the hash, generated with the system's RSA private key.
4. **Digital Signature Scheme**: The system features digital signing of user data hashes to assert authenticity and non-repudiation. Generated RSA key pairs (2048 bits or higher) are stored securely within the system. When a data audit or verification stage occurs, the signature is

checked against the stored public key, verifying that data has not been altered maliciously.

5. **Audit Logging:** Every sensitive action, such as user creation, login attempts, QR code accesses, and data modifications—is logged into an encrypted audit trail, preserving confidentiality and enabling retrospective security analysis.
6. **User Interface and Access:** The entire system is controlled via a robust command line interface (CLI), designed to run seamlessly in minimal environments including SSH terminals, Linux command prompts, and cloud servers without graphical environments. For secondary setup tasks like TOTP enrollment, the system generates QR codes viewable as ASCII art in terminals or through an embedded temporary HTTP server that displays QR images, ensuring compatibility with wide ranging deployment scenarios.

II-C. Architectural Diagram



Technology Stack

- **Technology Stack:** Python 3.11+
- **Core libraries:**
  - `bcrypt` for password hashing
  - `cryptography` (Fernet) for symmetric encryption of user data
  - `pyotp` for TOTP multi-factor authentication
  - `qrcode` and `Pillow` for QR code generation
  - `pycryptodome` or `cryptography` RSA module for digital signature
  - `csv`,`json` for data serialization and storage
- **Containerization:** Docker (for environment isolation and easy deployment)
- **Transport Security:** SSH for remote secure interaction; optional local AES encryption for data storage

Design Justification

- The use of bcrypt offers resistance against offline password attacks.
- Encrypting the whole user database ensures confidentiality even if disk is compromised.
- Digital signing of hashes guarantees data authenticity and integrity.
- MFA with TOTP prevents unauthorized access even with compromised passwords.
- Audit logs provide accountability and traceability necessary for compliance.
- CLI with ASCII QR codes ensures usability in headless or remote environments, typical in cloud or security conscious deployments.

Security Considerations

- The system enforces strict validation of user input.
- Secret keys and private keys are generated locally and never exported.
- Key rotation and backup strategies are recommended to strengthen operational security.
- The architecture limits attack surface by not exposing unencrypted data or services externally.

III. Implementation Details

The implementation of the Secure Identity Management System (SGI) integrates multiple security mechanisms and Python technologies to ensure robust identity and access management. Below is a detailed description of the key components and workflows involved:

III-A. User Creation and Credential Management

Users are uniquely identified by usernames, and credential security is paramount. When creating a new user:

- **Password Generation:** A cryptographically strong random password (32 characters, including letters, digits, and symbols) is generated using Python's `secrets` library.
- **Password Protection:** The password is hashed with `bcrypt`, a modern adaptive hashing function that incorporates salt and multiple hashing rounds, making brute force attacks computationally infeasible. The hash is stored, never the plaintext password.
- **Role Assignment:** Users are assigned predefined roles (Finance, HR, IT, etc.), which help control permissions related to system administration and access control.
- **Storage:** User data, including hashed password, personal information (name, age, etc.), and role is securely serialized and encrypted before persistence.

III-B. Secure Data Storage and Encryption

The system's user database is a CSV file stored encrypted on disk:

- **Symmetric Encryption using Fernet:** The entire CSV database is encrypted with `Fernet` (from Python's cryptography package), which uses AES-128 in CBC mode with HMAC for authentication. The symmetric encryption key is generated once on first run and stored as a key file (`keyfile`).
- **Data Serialization:** User details are serialized to CSV format in memory after validation, and then encrypted to prevent exposure if storage media is compromised.
- **Decryption Process:** When the system needs to read user data, it decrypts the CSV file to rebuild the users list in memory for application use.

III-C. *Integrity Verification: Hashing and Digital Signatures*

- **Data Hashing:** Each user's personal data is converted to JSON and hashed with SHA-256 ensuring data integrity.The computed hash is signed with RSA private key (`private_key.pem`), providing non-repudiation and protecting against tampering.
- **Digital Signature:** The computed hash is signed with RSA private key (`private_key.pem`), providing non-repudiation and protecting against tampering.
- **Verification:** On data load or audit, the system verifies both the hash and the corresponding signature with the RSA public key. Mismatch indicates data corruption or attacks.

III-D. *Authentication and Authorization:*

- **Login Process:** Password verification is done by comparing bcrypt hashes. No password is revealed or stored in plaintext.
- **Multi-Factor Authentication (MFA) with TOTP:** Time-based One-Time Passwords are generated for users during enrollment using pyotp. Users scan a QR code rendered in ASCII (or optional HTTP server for graphical display), then enter OTP codes for validation.
- **Role Based Access Control:** User roles dictate access to administration functions, limiting critical actions only to authorized personnel (admins/root).

III-E. *Cryptographic Key Management*

- **Key Generation:** At first setup, the system generates:
  - A Fernet symmetric key for overall database encryption.
  - RSA private/public key pairs for signing and verification.
- **Key Storage:** Keys are securely stored as files within the user directory, never shared or transmitted.
- **Key Security:** Given their sensitivity, private keys are used solely in local cryptographic operations and never exposed.

III-F. *Auditing and Logging*

- **Encrypted Audit Trails:** All significant user actions (registrations, logins, failed attempts, manipulations) are logged into encrypted files using the same Fernet encryption.
- **Immutability and Traceability:** Logs include timestamps, user identifiers, action types, and cryptographic hashes to prevent tampering.
- **Access Control:** Only root users can retrieve and decrypt logs, ensuring confidentiality.

III-G. *User Interface and Workflow*

- **Command-Line Interface:** The entire user experience is controlled by simple CLI menus and prompts, ensuring functionality in headless environments (servers, SSH sessions, containerized deployments).
- **QR Code Handling:** QR codes for MFA setup are dynamically generated and rendered directly in the terminal as ASCII art, compatible with diverse clients and remote sessions.
- **Language Support:** The UI supports English and Spanish, improving accessibility and user friendliness.

III-H. *Deployment Environment:* ***Docker***

- The system is packaged within a Docker container for consistency and portability.
- Exposes SSH on customized ports with persistent storage mounted for user data and logs.
- Supports remote CLI interaction over secure tunnels.

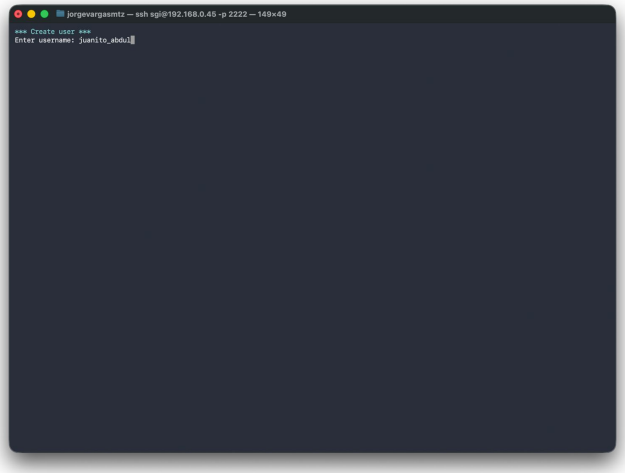
IV. Testing and Proof Evidence

This section presents the main functional and security tests carried out to validate the Secure Identity Management System (SGI). Each test demonstrates that the system meets the requirements for security, integrity, and traceability established in the project objectives.

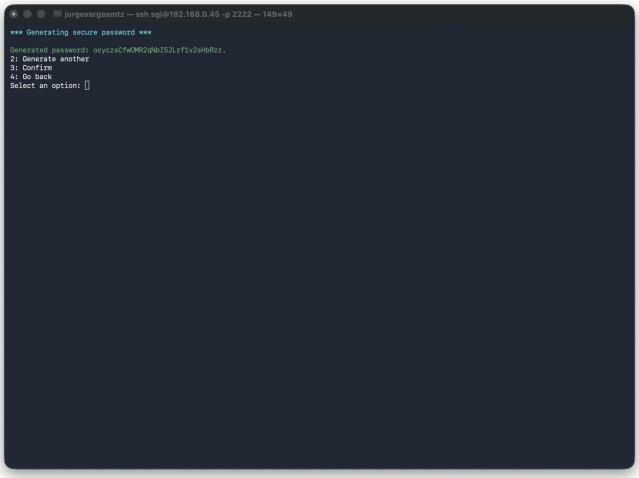
IV-A. *User Lifecycle Tests*

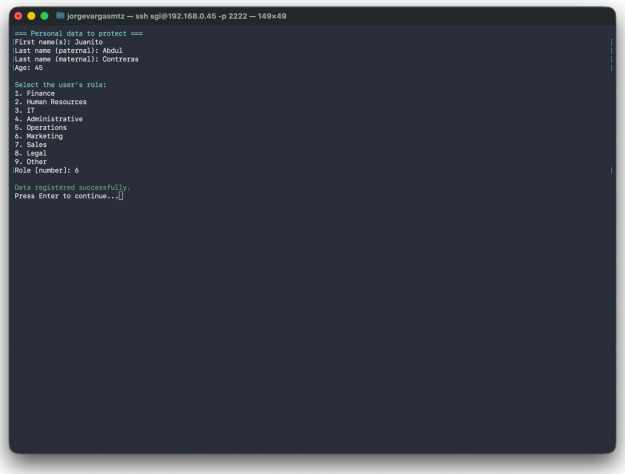
Test 1: Secure User Creation

- A new user is created using the CLI.



- The password is generated automatically and appears only once on screen (never stored bare).

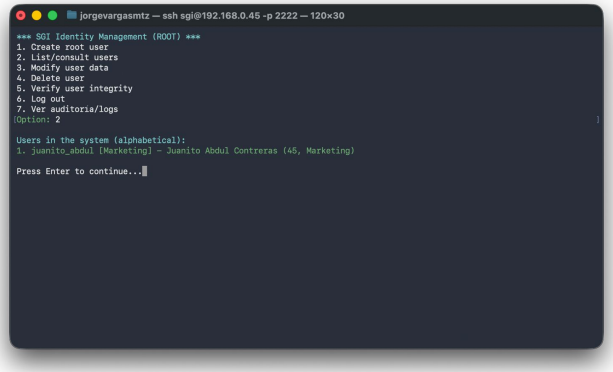




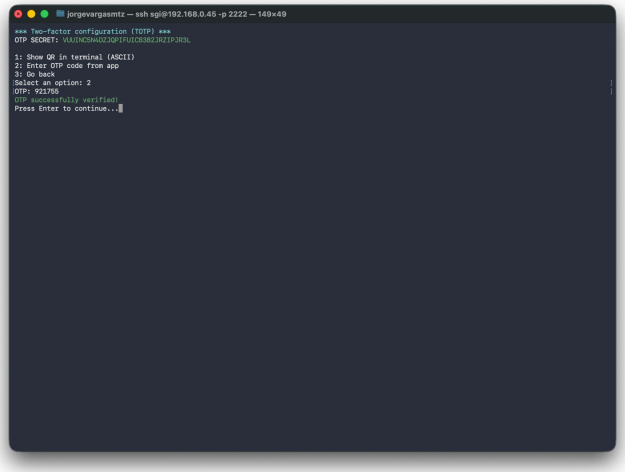
- The user's record in the encrypted database is verified to contain only a bcrypt-hash—not the original password.

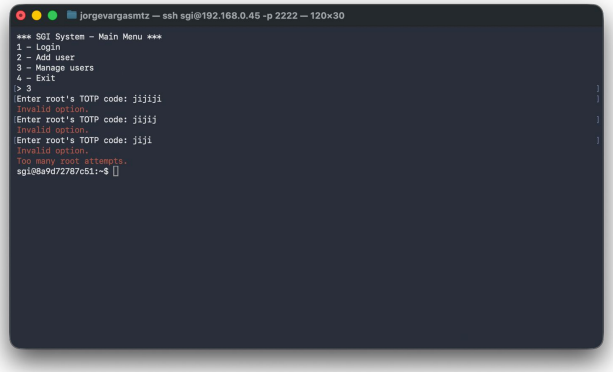
Test 2: Role Assignment and User Management

- Users are assigned appropriate roles.



- Attempts to perform admin-only actions as non root are denied, validating access control.





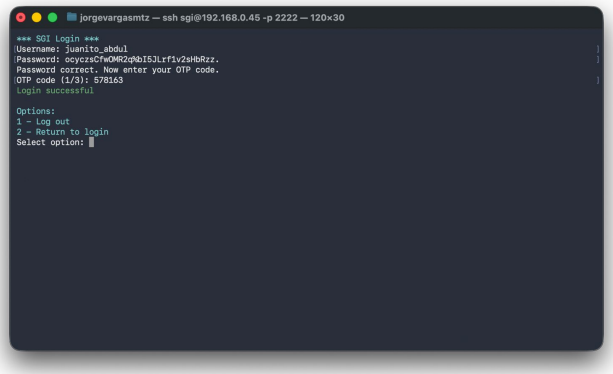
- The system queries for personal data (name, age, role) and associates a role.

IV-B. Authentication and MFA

Test 3: Login with Correct Credentials

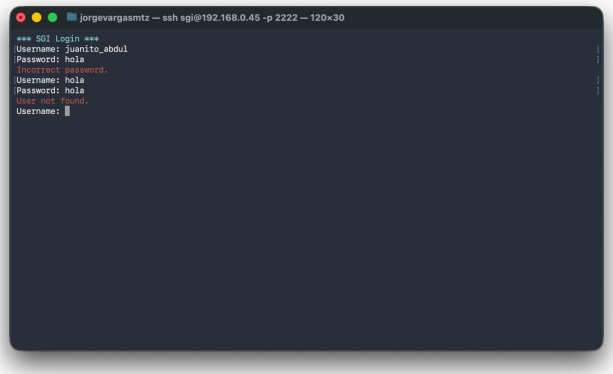
- The registered user authenticates using their password and TOTP (generated in an authenticator app after scanning the QR code).

- Login only succeeds if both password and time-based code are valid.

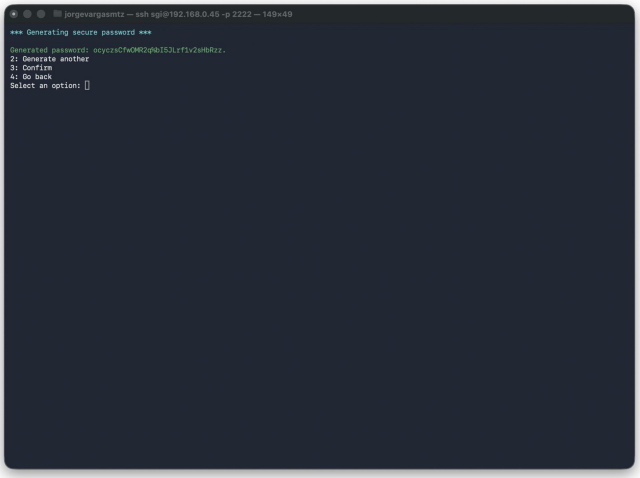


Test 4: Login with Incorrect or Weak Passwords

- Attempting to login with a bad password fails.

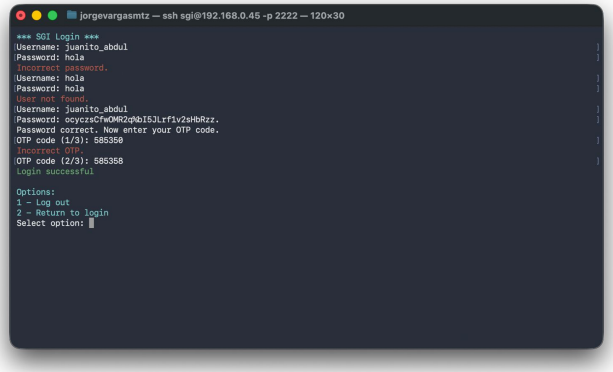


- The system refuses to create users with weak or non-compliant passwords (by design, only strong passwords are generated).



Test 5: TOTP MFA Enforcement

- Login rejects incorrectly entered TOTP codes.



- New users cannot proceed until MFA registration and successful OTP validation.

Test 6: Detection of Tampered Data

- The encrypted database file (`user_db.csv`) is manually altered (e.g., change of age/role/fields).
- On the next load, the system detects hash/signature mismatch and flags the data as invalid, blocking access.

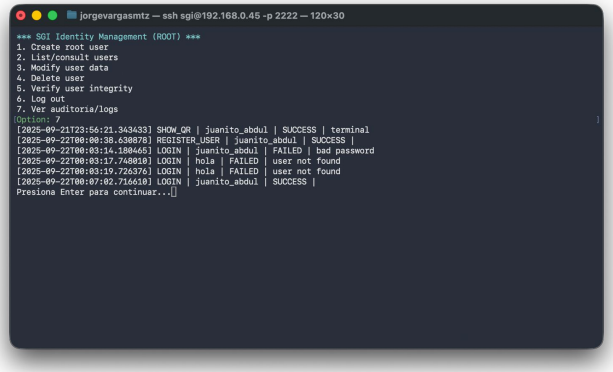
Test 7: Digital Signature Validation

- During an integrity check, each record's SHA-256 hash and RSA signature are validated—proving no data tampering occurred if all are correct.

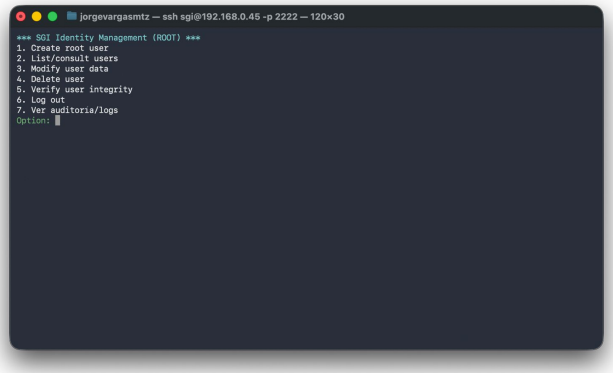
IV-C. Auditability and Logging

Test 8: Encrypted Logging of All Events

- The system records every critical action (user creations, logins, modifications) into an encrypted log file (`/users/logs/log_audit.enc`).



- Only admin/root can decrypt and inspect the logs, confirming their confidentiality.



Test 9: Log Immutability

- An attempt to alter or remove log entries fails. Upon admin review, signature/hashing inconsistencies in the logs are detected and flagged.

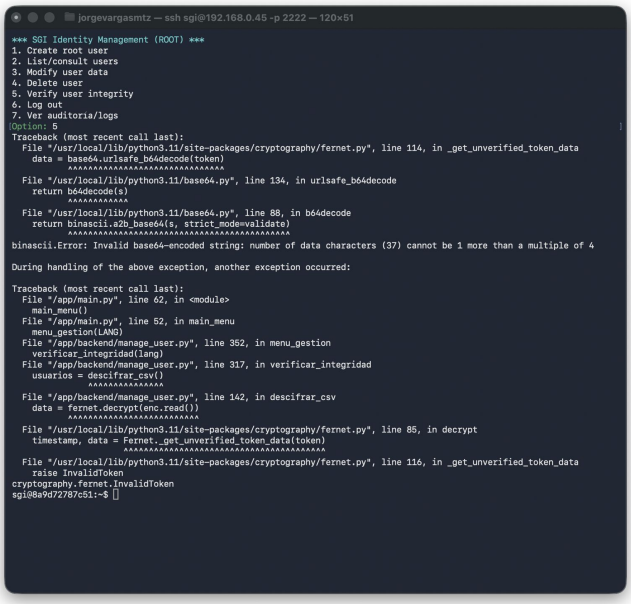


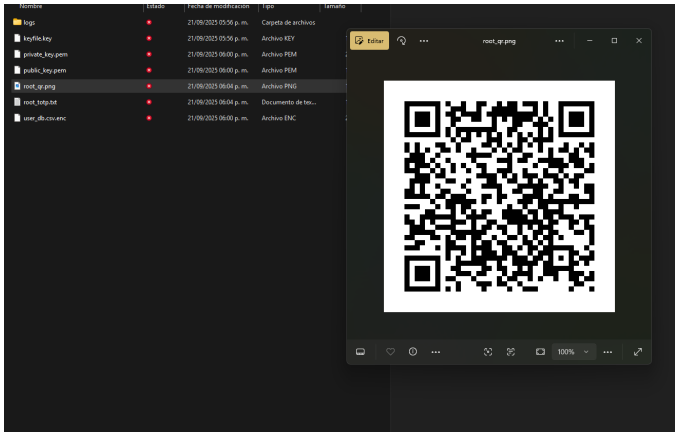
Figura 1. Deleting the database can stop the program execution.

IV-D. QR/Remote and SSH Compatibility

Figura 2. Stats when using via SSH

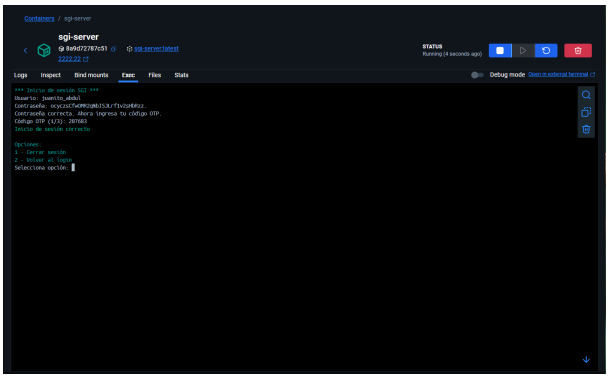
Test 10: QR Code Enrollment via CLI & Remote Access

- QR enrollment is tested in both a local Linux console and over an SSH session from another machine.



- ASCII QR is rendered correctly; external authenticator apps scan and generate valid OTPs.

- System functions equally in Windows, Linux, and containerized (Docker) deployments



IV-E. Screenshots and Evidence

For each critical operation, screenshots are taken of:

- The CLI during user registration (showing password generation and QR code rendering).
- Login attempts with valid and invalid credentials.
- Admin actions (logs review, audit trail).
- Evidence of error/warning when tampered data or incorrect TOTP is provided.

V. Audit Logging

Audit logging is a critical component of the Secure Identity Management System (SGI), providing traceability, accountability, and forensic capability. The audit log records all significant events performed within the system—such as user creation, authentication attempts, administrative changes, and data access.

V-A. Objectives and Role

The primary objectives of audit logging in SGI are:

- To provide an immutable record of critical user and system actions.
- To ensure compliance with security and data integrity policies.
- To support investigation and response in the event of a security breach or suspicious activity.

V-B. What is Logged?

The system logs the following actions:

- User registration (username, role, timestamp)
- Successful and failed login attempts
- MFA enrollment and validation events
- Administrative actions (modification, deletion, data review, integrity verification)
- QR code accesses for TOTP setup
- Any unauthorized or rejected actions

Every log entry includes:

- Timestamp of the event
- Actor (username or “root/admin” if privileged action)
- Action type
- Outcome/status (success, fail, rejected, etc.)
- Hash or cryptographic evidence (where applicable)

V-C. Log Security and Integrity

All audit log entries are encrypted on disk using the same Fernet symmetric key as the user database:

- The log file (`log_audit.enc`) is not readable or modifiable without proper authentication and the encryption key.
- Only system administrators (root) can decrypt, review, or export the audit log, protecting sensitive operational details.
- Any attempt to alter audit log entries breaks the cryptographic hash/signature chain, resulting in a detection of log tampering.

V-D. Operational Flow

Audit logging is tightly integrated into every critical system workflow:

1. Before or after every significant operation, an audit event is generated and appended to the encrypted log.
2. The logging mechanism is protected against concurrent or unauthorized access.
3. Log retrieval (for audit or compliance review) is performed via privileged CLI commands, enforcing separation of duties.

V-E. Evidence and Example

Below is an (anonymized) sample of decrypted log entries for illustration:

```
[2025-09-20 16:23:10] USER:alice ACTION:register STATUS:success
[2025-09-20 16:25:41] USER:alice ACTION:mfa_enroll STATUS:success
[2025-09-20 16:26:01] USER:alice ACTION:login STATUS:fail (invalid otp)
[2025-09-20 16:27:48] USER:admin ACTION:log_review STATUS:success
```

V-F. Compliance and Retention

Best practice recommendations:

- Backup encrypted logs as part of the overall disaster recovery plan.
- Periodically rotate and archive logs to ensure performance and facilitate long term forensic investigations.
- Store logs on secure, access controlled media.

VI. Integrity and Data Compliance

Ensuring the integrity and compliance of personal and identity data is at the core of the Secure Identity Management System (SGI). This section describes the technical and procedural mechanisms employed to protect against unauthorized data modification, guarantee traceability, and support compliance with data security and privacy regulations.

VI-A. Data Integrity Mechanisms

- **Cryptographic Hashing:** Every sensitive user data record is hashed using the SHA-256 algorithm. This hash represents a fingerprint of the data; any minimal change to the data yields a completely different hash value.
- **Digital Signatures:** The SHA-256 hash of each user record is digitally signed using the system's RSA private key. The resulting signature, along with the hash itself, is stored alongside each record. This provides non-repudiation and ensures data authenticity.
- **Signature Verification:** On every access, modification, or audit operation, the system recomputes and verifies the hash and signature using the RSA public key. Any mismatch is promptly detected and signaled as an integrity violation.
- **Tamper-Proof Storage:** All records are stored in a symmetrically encrypted database file (Fernet/AES), preventing unauthorized direct access or modification at the storage level.

VI-B. Compliance Procedures

- **Change Management:** Any change to personal data (modification, deletion, or recovery) is always processed through the secure application interface, which automatically triggers re hashing and re-signing of data.
- **Auditability:** All modifications, access attempts, and integrity checks are logged in the encrypted audit trail, supporting traceability, forensic investigations, and regulatory inquiries as needed.
- **Access Control:** Only authorized administrators can access, audit, or export sensitive identity records, ensuring compliance with the principle of least privilege and segregation of duties.
- **Fail Secure Behavior:** If an integrity violation is detected (hash/signature fails to validate), the affected records are locked down, access is denied, and an alert is recorded in the logs—preventing use of potentially tampered data.

VI-C. Data Retention and Compliance

- **Data Retention:** User and audit records are retained according to the organization's data policy. Encrypted backups are recommended as part of best practices.
- **Privacy Considerations:** Plaintext access to personal data is only available in-memory to privileged processes; at rest, data is always encrypted and integrity checked.
- **Legal and Regulatory Alignment:** The use of strong encryption, digital signatures, and auditable action records supports compliance with major security standards (such as GDPR, ISO/IEC 27001, and industry best practices for identity protection).

VII. Improvements and Additional Recommendations

- **Key Rotation and Management:** Implement automatic rotation and periodic renewal for both symmetric (Fernet) and asymmetric (RSA) keys to minimize risk if a key is exposed. Integrate with a dedicated secrets management tool (e.g., HashiCorp Vault) for enterprise scenarios.
- **End-to-End Encrypted Backup and Disaster Recovery:** Automate regular encrypted backups of the user database and logs to secure, off server locations. Test and document full restore procedures.
- **Stronger MFA and Passwordless Options:** Expand support for hardware based authentication (e.g., Yubikey, FIDO2) or mobile push notifications for true passwordless security.
- **Session Management and JWT Hardening:** Move towards using signed JSON Web Tokens (JWT) for session persistence, coupled with short expiration periods and periodic re verification with TOTP.
- **Granular Role-Based Access Control (RBAC):** Implement more flexible and detailed permission models, including delegation of specific administrative rights, rather than a simple root/non root model.
- **Automated Vulnerability and Integrity Monitoring:** Integrate periodic automated system scans, monitoring file integrity (e.g., using tools like Tripwire), and real time alerting for attempted tampering or unauthorized access.
- **Compliance Automation:** Develop exportable compliance reporting (e.g., for GDPR, ISO/IEC 27001) with audit log summaries and checklist compliance status.
- **REST API and Web Dashboard:** For environments that require graphical dashboards, create a secure web interface with proper CSRF, XSS mitigations, and strict authentication, without exposing core services to the internet.

VII-A. Best Practice Recommendations

- Regularly review and update all cryptographic libraries to defend against newly discovered vulnerabilities.
- Enforce strict file system permissions, especially for private keys and encrypted user data.
- Conduct regular security audits and penetration testing of the entire system.
- Employ least privilege principles for all users and administrators.

VII-B. Notes on Production Use

Although the current implementation is robust for academic and controlled professional contexts, any deployment in a real enterprise setting should be preceded by an in-depth security review, compliance verification, and operational hardening to address organizational risk and real world threats.

VIII. Conclusion

The development and implementation of the Secure Identity Management System (SGI) have demonstrated the feasibility and effectiveness of building a robust, auditable, and tamper-resistant framework for digital identity protection within an organization. By integrating state-of-the-art security mechanisms—such as cryptographically strong password hashing (bcrypt), AES-encrypted storage, TOTP-based multi-factor authentication, and digital signature schemes—the system successfully addresses the primary goals of integrity, confidentiality, authenticity, and auditability.

The rigorous application of access control, encrypted audit logging, and real-time integrity verification ensures that any attempt at unauthorized access or data tampering is either blocked or detected in a timely manner. Moreover, the system's deployment-agnostic design, including SSH/CLI workflows and QR code handling for MFA, makes it suitable for modern, distributed, and cloud-enabled environments.

While further improvements are possible, the current iteration of SGI provides strong protections aligned with leading industry practices and compliance requirements. The project highlights not only the technical challenges of identity and access management, but also the importance of holistic system-level thinking, operational procedures, and ongoing review in effective cybersecurity.

IX. References

- **Bresz, F., Renshaw, T., Rozek, J., White, T., & Rai, S.** (2007). *Gestión de identidades y accesos (GTAG 9)*. Recuperado de https://auditoresinternos.es/wp-content/uploads/2022/07/12_GTAG9-Gestion-de-identidades-y-accesos.pdf
- **SailPoint Technologies, Inc.** (2025). *¿Qué es la administración de identidades y accesos (IAM)?* Recuperado de https://www.sailpoint.com/es-mx/identity-library/identity-and-access-management
- **DataCamp.** (2025). *Entrada de usuario de Python: Manipulación, validación y buenas prácticas*. Recuperado de https://www.datacamp.com/es/tutorial/python-user-input
- **GeeksforGeeks.** (2022). *Hashing Passwords in Python with BCrypt*. Recuperado de https://www.geeksforgeeks.org/python/hashing-passwords-in-python-with-bcrypt/
- **pyca.** (2025). *bcrypt: Modern(-ish) password hashing for your software and your servers*. Recuperado de https://github.com/pyca/bcrypt
- **Udemy.** (2025). *Python TOTAL - Programador Avanzado en 16 días*. Recuperado de https://www.udemy.com/course/python-total/?couponCode=25BBPMXNVD25CTRL
- **Cryptography Developers.** (2025). *Welcome to pyca/cryptography — Cryptography 47.0.0.dev1 documentation*. Recuperado de https://cryptography.io/en/latest/
- **Nooner, M.** (2016). *Welcome to PyQRCode's documentation! — pyqrcode 1.2 documentation*. Recuperado de https://pythonhosted.org/PyQRCode/
- **PyOTP Contributors.** (2025). *PyOTP documentation*. Recuperado de https://pyauth.github.io/pyotp/
- **Sweigart, A.** (2014). *Welcome to Pyperclip's documentation! — Pyperclip 1.5 documentation*. Recuperado de https://pyperclip.readthedocs.io/en/latest/
- **Reitz, K., & colaboradores.** (2025). *Requests: HTTP for Humans™ — Requests 2.32.5 documentation*. Recuperado de https://requests.readthedocs.io/en/latest/
- **Clark, J. A., Lundh, F., y colaboradores.** (2025). *Pillow (PIL Fork) 11.3.0 documentation*. Recuperado de https://pillow.readthedocs.io/en/stable/
- **Microsoft.** (2025). *Dockerfile y contenedores de Windows*. Recuperado de https://learn.microsoft.com/es-es/virtualization/windowscontainers/manage-docker/manage-windows-dockerfile
- **Skillsoft.** (2025). *Gestión de usuarios*. Recuperado de https://documentation.skillsoft.com/es/percipio/Content/A_Administrator/admn_user_management.htm
- **Fadheli, A., & Abdullahi, M.** (2024). *How to Implement 2FA in Python - The Python Code*. Recuperado de https://thepythoncode.com/article/implement-2fa-in-python

Cuadro I
SECURITY CHECKLIST AND EVALUATION

Item	Evaluation
▪ Adequate design of proposed security solutions. **Fact:** This ensures the system architecture is robust against known threats from the start. ▪ Correct implementation of integrity mechanisms (hashing, signatures, MFA). **Fact:** These mechanisms are crucial for preventing unauthorized data tampering. ▪ Effectiveness in detecting data alteration attempts. **Fact:** A high detection rate indicates a well-configured monitoring and alerting system. (2.5 points)	Done.
▪ Quality and completeness of the tests performed. **Fact:** Thorough testing is the primary way to validate the functionality and security of the implemented solutions. ▪ Clear documentation of implemented security mechanisms. **Fact:** Comprehensive documentation is vital for maintenance, future development, and knowledge transfer. (2.5 points)	Done