Universidade de Aveiro

Master's in Cybersecurity

Identification, Authentication and Authorization

Year 2024/25

Privacy Preserving Authentication with Verifiable Credentials

Made by:

Carlos Ferreira 108822

Tomás Bogalho 124224

Index

1. Introduction

This project is part of the Identification, Authentication, and Authorization course, focusing on a secure authentication system for military personnel implementing some of the concepts of eIDAS 2.0 and Verifiable Credentials (VCs).

eIDAS 2.0 (Electronic Identification, Authentication, and Trust Services) is an updated European regulation enhancing digital identity frameworks across EU member states. A key component is the European Digital Identity Wallet, which enables secure, cross-border authentication for citizens, businesses, and governments. This regulation strengthens trust services by ensuring interoperability and security for electronic signatures, seals, and certificates. Importantly, it empowers users with control over their personal data while fostering a unified digital identity ecosystem.

Verifiable Credentials (VCs) are digitally signed, tamper-proof credentials that verify identity, qualifications, or attributes in a decentralized and privacy-preserving manner. They eliminate the need for central authorities during verification while ensuring authenticity and integrity through cryptographic signatures.

The project will develop an authentication system that enables military personnel to securely access specific sections within European military bases, ensuring compliance with security clearance levels.

This system will be implemented considering that personnel has already entered the base, as the initial authentication process requires proving their identity, making the application of selective disclosure impractical during that stage.

2. Use Case

The project's authentication system is designed for a high-security facility access control system, specifically within military environments. The primary objective is to manage and secure access for individuals who have already entered the outer perimeter of a base, allowing them to authenticate themselves for entry into various internal areas.

The system leverages Verifiable Credentials (VCs) to facilitate this access, incorporating a tiered approach with three distinct Levels of Assurance (LoA). These LoAs are dynamically applied by the Interface component based on the sensitivity of the area being accessed, providing a flexible yet robust security framework. The overarching goal is to implement a privacy-preserving and highly secure authentication method that significantly minimizes the exposure of personal data, thereby upholding individual privacy while strictly enforcing access control policies.

The strategic decision to implement this system for internal access control within a military base, rather than for initial entry, is founded on several key considerations related to the project's core objectives, the nature of identity verification, and the practical application

Since at the perimeter of a military installation, the paramount security requirement is full and unambiguous identity verification. This stage typically involves comprehensive checks, including biometrics, physical document inspection (e.g., military ID, passport), and often cross-referencing against national security databases. The primary goal is to establish who is attempting to enter and whether they are authorized to be on the premises at all. In this context, revealing a complete identity is not just practical but often legally mandated and crucial for national security, accountability, and immediate threat assessment. Selective disclosure, where a user only proves "I am authorized for entry" without revealing their specific identity, is less applicable or even counterproductive at this initial, high-stakes identity vetting stage.

2.1. Pratical Example

To illustrate the system's practical application, consider "Eagle Base," a highly secure military installation.

2.1.1. Initial Credential Issuance:

Upon a new recruit, "Cadet Alex," joining Eagle Base, they undergo a rigorous background check. Once cleared, the Issuer (simulating the Military Authority/Identity Provider) issues a Verifiable Credential to Cadet Alex. This VC, containing attributes like security_clearance_level: "Confidential", rank: "Cadet", and original_national_base: "PT-1234", is securely loaded onto Alex's smart card (the Holder's Wallet Unit). A unique PIN for higher LoA access is also provided to Alex upon creation.

2.1.2. LoA 1 Access - The Dining Hall:

It's lunchtime, and Cadet Alex needs to enter the dining hall. At the dining hall's access point, Alex taps their smart card on the reader (the Interface). The Interface, configured for LoA 1 (Low), requests only the necessary proof of general base authorization from Alex's VC. Alex's smart card, without requiring a PIN, uses its low-level private key to cryptographically sign a challenge presented by the Verifier. The Verifier quickly confirms the signature and the credential's validity (checking against the Issuer and Blockchain for revocation status and public key resolution). Since only minimal information is needed and revealed, Alex's privacy is preserved. The "Door Open!" light flashes, and Alex enters the dining hall.

2.1.3. LoA 2 Access - The Server Room:

Later that day, Alex, now a qualified IT Specialist, needs to access the Base's main server room to perform maintenance. This room is classified as a "Substantial Access" zone (LoA 2). Alex approaches the server room's access terminal (Interface) and taps their smart card. The Interface prompts Alex to enter their PIN. Alex types in their confidential PIN. This PIN decrypts the high-level private key stored on the smart card, allowing it to sign a more complex cryptographic challenge from the Verifier. The Verifier, after resolving the necessary public key from the Blockchain, validates the signature and confirms Alex's security_clearance_level: "Confidential" via selective disclosure. The access system acknowledges the successful high-assurance authentication, and the server room door unlocks.

2.1.4. LoA 3 Access - The Command Center:

In a critical situation, the Joint Command Center LoA 3 zone (High) needs to be accessed. This highly sensitive area requires the presence and authentication of at least three specific high-ranking officers from a designated group of five. Captain Maria, Major Ben, and Colonel David, all part of the authorized group, arrive at the Command Center access point. Each individually taps their smart card on the Interface and enters their respective PINs. The Interface, designed for LoA 3, collects the successful authentications from all three officers, verifying that they belong to the correct original_national_base group and collectively meet the "three-person rule." Once the Interface confirms three valid LoA 2 authentications from the designated group, it signals the Verifier for final approval. The Verifier, having individually confirmed each officer's LoA 2 credentials, grants access to the Command Center. This multi-party authentication ensures an additional layer of security for the base's most critical asset, preventing any single point of failure or compromise.

2.2. Scenario

A military or high-security facility requires strict access control after an individual has entered the base. Each person is issued a VCs, which is stored securely in a smart card (Wallet Unit). The facility enforces three Levels of Assurance (LoA):

- **LoA 1 (Low Access)** – Grants entry to general areas such as offices, dining halls, or recreational spaces.
- **LoA 2 (Substancial Access)** – Required for entry into restricted zones such as data centers, armories, or command rooms.
- **LoA 3 (High-Security Access)** – Requires at least 3 people present in person out of a group of 5 to enter high security zones. This LoA applies only to the specific zones and not to the VCs themselves. This means that the VCs only have 2 different LoA while LoA 2 (from the VC) is also used for LoA 3 of the zone.

Instead of using traditional ID cards or passwords, the system ensures minimal disclosure of personal data while still verifying access privileges.

2.3. Level of Assurance

The system will support three LoA levels:

- Low: Only requires the verification of low level private key.
- Substantial: Requires the use of the high level private key stored inside the card.
    - This key requires a the Holder to enter a PIN that will be decrypt the key stored in the card.
- High: Requires the use of the high level private key stored inside the card of at least 3 out of 5 people authorized to access the room. If any of them fails the authentication process, access will be denied. (This key also requires the Holder to enter a PIN).

2.4. Applying the Code to the Use Case

To clarify the relationship between the use case and the developed code, the following table maps each role in the use case to its corresponding system component. It specifies the associated Python script, outlines its key functionalities, and describes the communication flows between components.

Use Case Role	System Component	Code Module	Responsibilities	Communication
Military Personnel (Holder)	Holder	`holder.py`	- Stores VC in the smartcard - Signs challenge - Handles PIN for Substantial and High access	- Sends authentication request to Interface with VC - Receives challenge - Sends signed challenge
Access Point Terminal	Interface	`interface.py`	- Acts as the entry gate/door - Defines the LoA - Forwards VC and LoA to Verifier - Redirects the challenge to Holder - Coordinates High LoA group check	- Receives VC from Holder - Sends VC to Verifier - Forwards challenge to Holder - Receives signed challenge and redirects it to Verifier
Service Provider	Verifier	`verifier.py`	- Generates challenge - Checks signature - Calls Issuer to validate VC	- Interface (VC, challenge) - Issuer (VC validity) - Blockchain (resolve DID)
Military Authority/ Identity Provider	Issuer	`issuer.py`	- Issues VCs - Signs credentials - Stores CRLs - Revokes VCs on trigger	- Verifier (VC validation, revocation) - Wallet (store VC) - Blockchain (register DID)
Smart Card VC Wallet	Wallet	`wallet.py`	- Stores issued VCs - Holds holder's metadata	- Receives VC from Issuer and stores it
Decentralized Registry	Blockchain	`blockchain.py`	- Registers and resolves DIDs - Stores public keys	- Registers keys from Issuer - Verifier queries for keys by DID

3. Components Mapping to eIDAS

The system will be composed by four components and their mapping to the eIDAS entities

- **Holder** - European Digital Identity Wallet (EUDI Wallet)
  - Implemented as a smart card, being the major alignements: storing Vcs, implementing selective disclosure, LoA and a PIN for high level credentials.
- **Interface** - Wallet Interaction Point
  - Allows the holder's wallet to interact with the remaining components.
- **Verifier** - Relying Party (Service Provider)
  - The service that verifies the holders identification, using the following alignements: implementing challenge-response authentication, supporting different LoAs and checking revocation lists.
- **Issuer** - Qualified Trust Service Provider (QTSP)
  - Since this is the component that isses the eletronic identification means, with the alignements: issuing cryptographically-sealed credentials, maintaining and publishing revocation lists.

3.1.1. Holder

The Holder corresponds to the military personnel who is issued and maintains possession of their Verifiable Credentials (VCs). These credentials are securely stored within a digital wallet, implemented in our system as a smart card interface. The VCs are encoded in a JSON format that complies with the W3C Verifiable Credentials Data Model specification, enabling structured, interoperable, and cryptographically verifiable identity assertions.

3.1.2. Interface

The Interface component acts as a card reader when applied to the use case, facilitating secure interactions between the Holder's smart card and the Verifier. It determines the required Level of Assurance (LoA) based on the access context and selects the corresponding private key from the smart card. For higher assurance levels, such as substantial or high, the Interface initiates a challenge-response protocol using the Holder's public key retrieved from the blockchain to generate a cryptographic challenge. The Holder must then respond by signing the challenge with the appropriate private key stored on the card, which, in the case of substantial LoA, requires PIN-based decryption.

3.1.3. Verifier

The Verifier is responsible for validating the authenticity and integrity of the Verifiable Credential (VC) presented by the Holder and redirected by the Interface. It operates as the system's relying party and performs cryptographic challenge-response validation to confirm the Holder's identity. Upon receiving a VC and the requested Level of Assurance (LoA), the Verifier resolves the Holder's public key through the Decentralized Identifier (DID) using the simulated blockchain. It then generates a challenge based on the required LoA and verifies the returned signature using the resolved public key. In addition, the Verifier consults the Issuer to confirm the credential's validity period and revocation status.

3.1.4. Issuer

The Issuer serves as the authoritative entity responsible for generating, signing, and issuing Verifiable Credentials (VCs) to the Holder. In alignment with eIDAS 2.0, it operates as a simulated Qualified Trust Service Provider (QTSP), ensuring the authenticity, integrity, and trustworthiness of the issued credentials. The Issuer generates cryptographic key pairs for the Holder, signs the VC using its own private key, and registers the associated public keys on the blockchain to enable future verification. In addition to issuance, the Issuer maintains a credential revocation mechanism publishing revocation information to a public list that the Verifier checks during the authentication process. Credentials may be revoked under specific conditions, such as repeated PIN failures, invalid cryptographic signatures, discharge of the Holder, or the detection of a security incident.

3.1.5. Blockchain

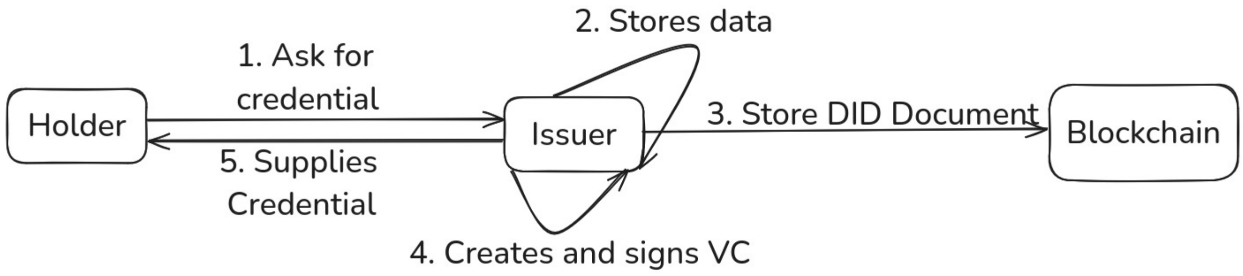
The Blockchain component emulates a decentralized identity registry, responsible for managing and resolving Decentralized Identifiers (DIDs) and their associated public keys. Although not implemented on an actual distributed ledger, this component simulates the behavior of a blockchain by maintaining a secure database of DID-to-key mappings. When a VC is issued, the Issuer registers the Holder’s public keys, corresponding to different Levels of Assurance (LoA), under a newly generated DID. During authentication, the Verifier queries the Blockchain to resolve a DID and retrieve the appropriate public key for signature verification. The Blockchain also supports credential revocation by deleting or invalidating DIDs, thereby disabling the use of associated credentials across the system. This simulated ledger ensures decentralized trust and verifiability of identities without relying on a central key repository.

3.2. Backend Logic

The Backend Logic forms the core of the authentication system, handling key operations such as supplying, verifying, and revoking credentials. It ensures that the authentication process remains secure, efficient, and compliant with the requirements set forth by the system architecture. The backend is responsible for maintaining the integrity of the credentials, ensuring their validity, and enforcing policies such as credential revocation in response to various conditions.

In this section, we will explore the essential functionalities of the backend logic:

3.2.1. Supplying Credentials



The image represents a simplified credential issuance process in a VC system. It illustrates five mains steps.

1) **Holder Requests Credential**: The entity labeled "Holder" initiates a request to the "Issuer" for a credential.
2) **Issuer Stores Data**: The Issuer records the necessary data before issuing the credential.

3) **DID is stores in blockchain**: The issuer sends the DID Document and the blockchain stores it.

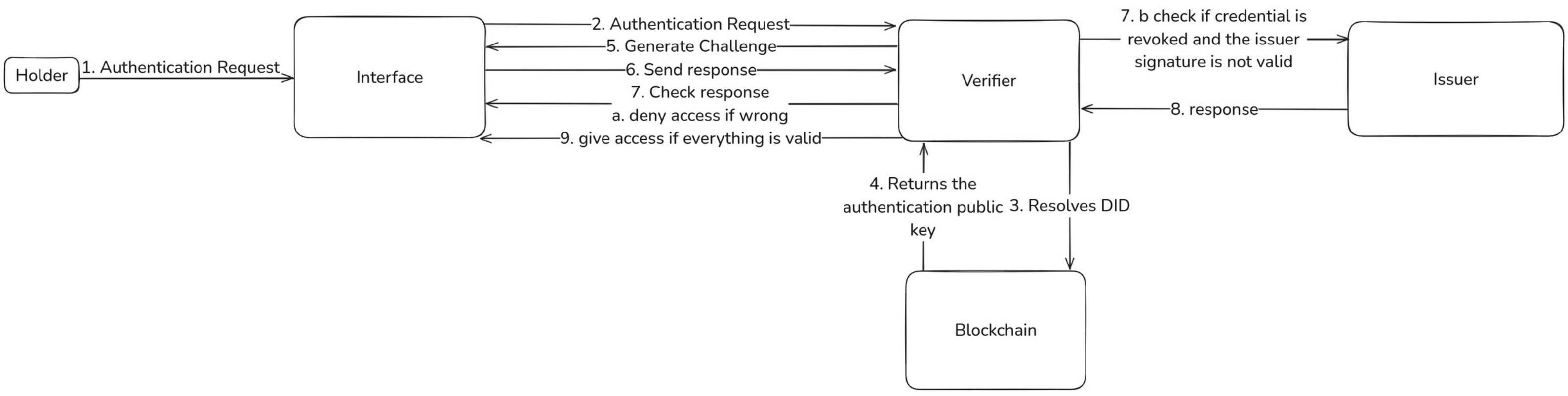
4) **Issuer creates and signs the VC**

5) **Issuer Supplies Credential**: The Issuer sends the credential back to the Holder.

    a) The holder stores the VC inside its wallet unit in our case, the smart card.

This diagram outlines the fundamental interaction between the Holder and Issuer, forming the basis for a privacy-preserving authentication system using Verifiable Credentials.

3.2.2. Verifying Credentials (Low LoA)


Holder
1. Authentication Request
Interface
2. Authentication Request
5. Generate Challenge
6. Send response
7. Check response
a. deny access if wrong
9. give access if everything is valid
Verifier
7. b check if credential is revoked and the issuer signature is not valid
8. response
Issuer
4. Returns the authentication public key
3. Resolves DID
Blockchain

This image represents the verification process of a VC with selective disclosure and challenge-response authentication. The flow includes these steps:

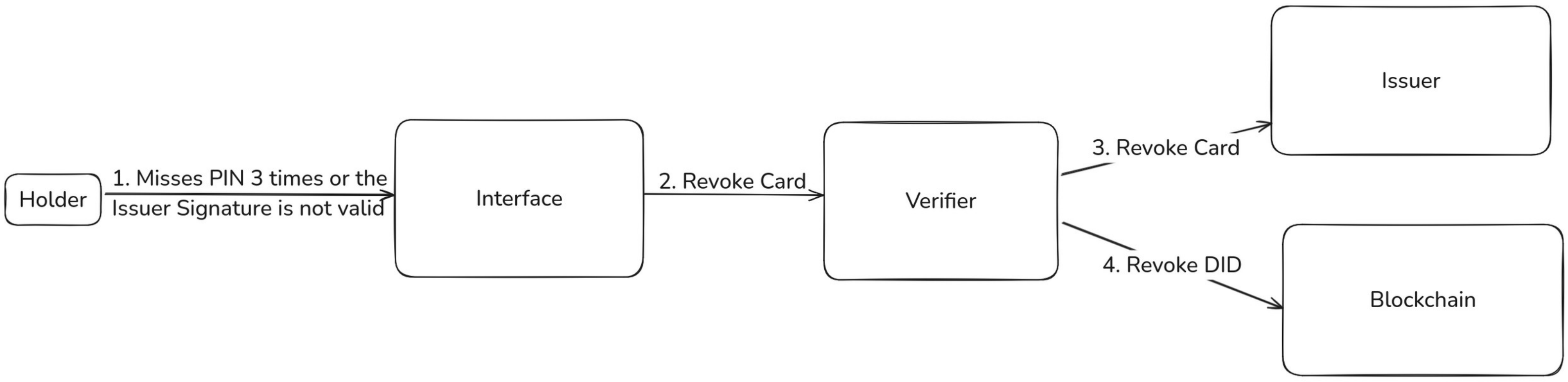
1. **User wants to authenticate:** Initiates a login request with the interface.
2. **Interface forwards the request**
3. **Verifier resolves the DID**: Verifier sends the DID to the simulated Blockchain to obtain the DID Document.
4. **Returns the public key**: Retrieves the public key from the DID Document and returns it according to the section clearance level.
5. **Verifier generates a challenge-response**
6. **User responds to the challenge**: The user signs the challenge and returns the response. If the authentication is high level, the user will be required to provide the corresponding PIN.
7. **Check challenge response**:
    1. If the response is wrong, access is denied
    2. Otherwise, the verifier will check with the issuer if the signature is valid and the smart card is not revoked
8. **Issuer responds**
9. **Verifier gives access if everything is valid**

3.2.3. Verifying Credentials (Substantial/High LoA)

These are a variation of the Low LoA credential verification however, they have one more step where the Interface asks the Holder for the PIN to the high level private key that will be used to respond to the challenge response.

In the case of a High LoA, the authentication process is also dependent on the other 2 people trying to enter the room.

3.2.4. Revoking Credentials



This diagram illustrates the revocation process of a credential when authentication failures occur. The flow begins when a Holder (user) repeatedly fails authentication or presents an invalid credential signature. Step-by-Step Description:

1. **Authentication Failure:**
    1. The Holder fails to enter the correct PIN three times or presents a credential with an invalid Issuer Signature.
    2. This triggers the revocation process at the Interface level.
2. **Revocation Request to Verifier:**
    1. The Interface sends a Revoke Card request to the Verifier.
    2. The Verifier is responsible for validating and processing the revocation request.
3. **Revocation Request to Issuer:**
    1. The Verifier forwards the Revoke Card request to the Issuer.
    2. The Issuer updates the credential status to revoked in the system.
4. **Revocation Request to Blockchain**

Once the credential is revoked, it can no longer be used for authentication. The user will need to request a new credential or go through a recovery process to regain access.

3.2.5. Credential Revocation Scenarios and Recovery Protocols

The integrity and security of any authentication system depend critically on robust credential revocation mechanisms and clearly defined recovery protocols. Each distinct scenario leading to credential revocation necessitates a tailored response to balance security, operational efficiency, and user experience. The following outlines the specific circumstances under which a Verifiable Credential (VC) may be revoked and the corresponding mandatory recovery actions.

1. Holder Misses PIN 3 Times (Temporary Suspension)

Scenario: If the Holder fails to enter the correct PIN for their high-level private key three consecutive times during a "Substantial" or "High" Level of Assurance (LoA) authentication attempt, the associated smart card (Wallet Unit) will be automatically locked. This immediate locking mechanism serves as a crucial security measure to prevent brute-force attacks and unauthorized access attempts using compromised PINs.

Recovery Action: To regain access, the Holder must personally appear at a designated security office or administrative center within the base. They will be required to submit a formal request for card unlocking, which is subject to rigorous identity verification processes (e.g., biometric verification, cross-referencing official records). This in-person requirement ensures that only the legitimate Holder can initiate the recovery, mitigating risks associated with remote compromise and reinforcing the security posture for higher LoA credentials.

2. Card with Invalid Signature (Permanent Revocation & Replacement)

Scenario: This situation arises when a Verifiable Credential presented by the Holder is determined to possess an invalid Issuer Signature. This could be indicative of various critical security issues, including:

- Tampering: The VC has been altered or corrupted since its original issuance.
- Forgery Attempt: The VC is not legitimate and was never genuinely issued by the trusted Issuer.
- Compromised Issuer Key: Although less likely in a well-secured system, it could hypothetically indicate a compromise of the Issuer's private key.

Recovery Action: In such cases, the compromised or invalid card's credentials are permanently revoked. The Holder is mandated to immediately request the issuance of a brand-new smart card containing fresh, cryptographically secure VCs. The old, invalid card must be physically handed over to security personnel for secure destruction, ensuring it cannot be re-used or analyzed for malicious purposes. This stringent protocol is essential to maintain the foundational trust in the system's issued credentials.

3. Discharge of the Holder (Permanent Revocation - No Recovery)

Scenario: When a Holder is formally discharged from their duty within the military or high-security organization, their authorization to access sensitive areas and information ceases entirely. This represents a definitive termination of their operational relationship with the facility.

Recovery Action: Recovery or reinstatement of credentials is categorically not permitted. All associated VCs are immediately and permanently revoked, and all access privileges are irrevocably terminated across the entire system. Furthermore, as part of the formal discharge process, the individual is required to return all identification material, and smart cards to ensure a complete and secure separation from the organization. This ensures that former personnel cannot retain or exploit any form of digital access.

4. Security Incident (Temporary Suspension & Investigation)

Scenario: If credential revocation occurs due to a suspected or confirmed security incident (e.g., detection of unusual activity associated with the Holder's account, a system-wide breach affecting credential integrity, or indications of compromise affecting the Holder's device), a comprehensive response is triggered.

Recovery Action: The Holder's smart card and associated credentials are temporarily suspended (by putting them in a revocation list temporarily) while a full investigation is initiated.

- If the Holder is found responsible for the security incident or is implicated in malicious activity, they will face appropriate disciplinary action, which may include permanent discharge from duty. In such instances, their credentials will be permanently revoked as per the "Discharge of the Holder" protocol.
- If no fault is found on the part of the Holder, and the incident is determined to be external or unrelated to their actions, their identity and associated credentials may be reinstated.

3.3. Data

3.3.1. Data Storage

According to eIDAS 2.0, secure storage is essential to guarantee that sensitive data is protected against unauthorized access, tampering, and data breaches.

The system must ensure that personal data and digital credentials are stored in a way that complies with data protection regulations such as GDPR and follows the best practices outlined by eIDAS 2.0 for trust services. Specifically, data storage for the VCs and related authentication information should meet the following requirements:

- **Encryption at Rest:** VCs stored in the holder’s digital wallet or in the issuers database are encrypted.
- **Access Control and Role-based Permissions:** Access to stored data must be strictly controlled. Only authorized personnel or systems should have access to sensitive data, and this access should be logged for auditability, for example following the principle of RBAC.
- **Data Minimization and Storage Limitation:** In compliance with GDPR and eIDAS 2.0, data should only be stored for as long as necessary for its intended purpose.
- **Secure Backup and Disaster Recovery:** Data stored in the system must be regularly backed up and protected against loss or corruption. Backup copies must also be encrypted and stored securely. Furthermore, a disaster recovery plan should be in place to ensure that data can be recovered in the event of a hardware failure, cyberattack, or other disruptive incident.

3.3.2. Data Transmission

Since sensitive information, such as Vcs and personal identification data, is exchanged between the Holder, Verifier, and Issuer, it is essential that this data is transmitted securely to protect against unauthorized access, tampering, and interception.

This section discusses the mechanisms and protocols used to safeguard data during transmission between the different components of the system. These include the use of encryption, secure communication protocols, and strategies for ensuring that the data integrity and authenticity of the credentials are preserved throughout the transmission process.

Key considerations for secure data transmission in the authentication system include:

1. **Encryption**: Ensuring that all sensitive data, such as VCs and the holder's personal information, is encrypted during transmission to prevent unauthorized access.
2. **Authentication of Communication Channels**: Verifying that data is transmitted only to trusted parties, preventing Man-in-the-Middle (MITM) attacks and ensuring the legitimacy of the recipient.
3. **Integrity Checks**: Ensuring that the data has not been altered during transmission through the use of hashing, signatures, or other integrity mechanisms.
4. **Secure Communication Protocols**: Using secure protocols such as TLS/SSL, HTTPS, or DTLS to safeguard data in transit and ensure that the data remains confidential and protected from tampering.

4. Technologies and Libraries for Implementation

4.1. Programming Language

The programming language used is Python since it is an easy-to-work-with language. In past projects, we have implemented functionalities similar to those required in this project, such as digital signatures and certificates. Additionally, Python offers extensive support for cryptographic and identity management operations, making it a suitable choice for our implementation.

For the implementation, the following libraries were used:

- **cryptography**: A robust library that offers cryptographic recipes and primitives, including asymmetric encryption (RSA, ECC), hashing, and certificate management.
- **Flask**: A lightweight web framework used to build the API endpoints for the application, enabling communication between client and server components.
- **requests:** A simple and user-friendly HTTP library for sending and receiving data via RESTful APIs, used primarily for client-side communication with external services or between internal modules.

These libraries together facilitated the secure exchange of information and provided a stable foundation for the project's authentication and encryption requirements.

5. Data Model for the VCs

The authentication system employs Verifiable Credentials (VCs) to represent digital military identification cards. The design of these VCs adheres to the W3C Verifiable Credentials Data Model specification, ensuring interoperability, cryptographic verifiability, and semantic richness through JSON-LD. This adherence is crucial for aligning with eIDAS 2.0 principles, which advocate for standardized and trust-enhancing digital identity solutions.

A Verifiable Credential is essentially a tamper-proof digital assertion about a subject (the Holder) issued by an Issuer. It contains claims (attributes) about the subject and a cryptographic proof generated by the Issuer, certifying the authenticity and integrity of the claims.

Data Field	Description	Type and Format	Purpose within Use Case
Original national base	The country and unique identifier of the military base where the Holder was initially assigned or is primarily stationed.	String Countries initial and ZIP code Example: PT-2560	Critical for managing personnel across different military installations and for group-based access (e.g., in LoA 3, ensuring all authenticated individuals originate from the same designated group/base).
Rank	The Holder's military rank.	String Example: Officer, Sargent, etc	Defines hierarchical access permissions and can be used for role-based access control within certain areas of the base, allowing or restricting entry based on rank protocols.
Division	The specific military division or branch the Holder belongs to.	String Example: Air Force, Marine, etc.	Similar to rank, division allows for departmental or branch-specific access restrictions, ensuring personnel only access areas relevant to their operational domain.
Security Clearance Level	The highest security clearance level granted to the Holder.	String	Crucial for determining Levels of Assurance (LoA). This attribute is the primary determinant for granting access to classified zones, enabling granular control over highly sensitive areas based on verified clearance.
Health Code	A specific internal identifier or code related to the Holder's health status (e.g., for access to medical facilities or specific operational roles).	String	Enables highly specific access permissions based on medical requirements without revealing detailed health records. For example, proving fitness for a particular duty station or access to a specialized medical bay.
Nationality	The Holder's nationality.	String. Example: PT, UK, UA	Necessary for compliance with international regulations or specific base protocols concerning multi-national personnel. Can be selectively disclosed if required by the access point's policy.

Data Field	Description	Type and Format	Purpose within Use Case
Full Name	The full name of the holder of the VcsThe complete legal name of the credential Holder.	String	Basic identification; can be selectively disclosed for verification procedures requiring personal identification, but can be omitted when privacy is paramount (e.g., proving only security clearance).
Initial Date	The date from when the VCs are valid	String dd-mm-yyyy	The initial date from when the VC becomes valid
Final Date	The date from when the VCs are became invalid	String dd-mm-yyyy	The final date from when the VC becomes invalid
Issuer	The entity that issued the VCsidentifies the entity (e.g., the Military Authority) that cryptographically signed and issued the VC.	String	Its presence is essential for Verifiers to know who vouched for the claims in the credential and to retrieve the Issuer's public key for signature verification.
High Level Private Key	The key used in the sections where the LoA is substantial, that is encrypted with a PIN that the holder must insert in order to use it. The issuer knows the correspondent public key.	String	Encrypted with a PIN, which the Holder must provide to unlock and use it for "Substantial" or "High" LoA authentication.
Low Level Private Key	The key used in the sections where the LoA is low, but unlike the previous one, this one doesn't need a PIN since it isn't encrypted. The issuer knows the correspondent public key.	String	Not encrypted with a PIN, a simplified version when the Holder only needs to present the smart card to the interface.
DID Identifier	The Decentralized Identifier (DID) of the Holder. This uniquely identifies the subject of the VC.	String (DID URI)	Essential for linking the VC to the specific individual and resolving their associated public keys for cryptographic challenge-response authentication. This allows the Verifier to retrieve the Holder's current public keys from the simulated Blockchain.

These VCs are third-party attested, meaning they are issued and cryptographically signed by a trusted authority (the Issuer, simulating a Qualified Trust Service Provider). This digital signature, contained within the proof field of the VC, ensures the authenticity and integrity of the credential. Any Verifier can cryptographically validate this proof to confirm that the VC has not been tampered with and genuinely originates from the declared Issuer, thereby preventing forgery and guaranteeing trustworthiness.

5.1. VC Schema

A fundamental aspect of designing a robust and interoperable Verifiable Credentials (VC) system is the precise definition of its schema. The VC Schema outlines the standardized structure, types, and expected attributes for each digital military ID card issued within our authentication system

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Verifiable Credential Schema",
  "type": "object",
  "properties": {
    "holder_pin": {
      "type": "string"
    },
    "signature": {
      "type": "string"
    },
    "vc_json": {
      "type": "object",
      "properties": {
        "did_identifier": {
          "type": "string"
        },
        "division": {
          "type": "string"
        },
        "final_date": {
          "type": "string",
          "pattern": "^\\d{4}/\\d{2}/\\d{2}$"
        },
        "full_name": {
          "type": "string"
        },
        "health_code": {
          "type": "string"
        },
        "initial_date": {
          "type": "string",
          "pattern": "^\\d{4}/\\d{2}/\\d{2}$"
        },
        "issuer_id": {
          "type": "string"
        },
        "nationality": {
          "type": "string"
        },
        "original_national_base": {
          "type": "string"
        },
        "private_pem_low_loa": {
          "type": "string"
        },
        "private_pem_substantial_loa": {
          "type": "string"
        },
        "public_pem_low_loa": {
          "type": "string"
        },
        "public_pem_substantial_loa": {
          "type": "string"
        },
        "rank": {
          "type": "string"
        },
        "security_clearance_level": {
          "type": "string"
        }
      },
      "required": [
          "did_identifier",
          "division",
          "final_date",
          "full_name",
          "health_code",
          "initial_date",
          "issuer_id",
          "nationality",
          "original_national_base",
          "private_pem_low_loa",
          "private_pem_substantial_loa",
          "public_pem_low_loa",
          "public_pem_substantial_loa",
          "rank",
          "security_clearance_level"
      ]
    }
  },
  "required": [
    "holder_pin",
```

6. Security Requirements

- **Multi-Factor Authentication (MFA):** Require multiple factors for authentication, like the need to insert the PIN when accessing sections with a substantial LoA.
- **Public Key Infrastructure:** Implement cryptography in the authentication process using digital signatures, certificates, and asymmetric key pairs. The methods used should be based on industry-standard cryptographic algorithms such as RSA (2048-bit or higher), ECC (Elliptic Curve Cryptography), and signing. Certificates should be issued by a trusted Certificate Authority (CA) or a simulated Qualified Trust Service Provider (QTSP) to ensure authenticity and integrity.
- **Secure VC Issuance**: Ensure that the VCs are issued by a trusted entity, such as a Qualified Trust Service Provider (QTSP) or a certified issuer, and are digitally signed using cryptographic keys.
- **Revocation Mechanism**: Implement a robust credential revocation system using techniques such as Revocation Registries, Status Lists, or OCSP (Online Certificate Status Protocol) to allow real-time verification of credential validity.
- **Selective Disclosure**: Implement cryptographic techniques such as BBS+ signatures, JSON selective disclosure signatures to enable users to share only the necessary attributes of their Verifiable Credentials (VCs)
- **Zero-Knowledge Proofs**
- **Compliance with GDPR**
- **Strong Encryption Standards:** Use AES-256 for data encryption and TLS 1.3 for secure communication.
- **Elliptic Curve Cryptography (ECC):** Prefer ECC-based cryptographic schemes for signing and authentication.
- **Key Management:** Implement secure key storage, rotation for the asymmetric key pairs.
- **End-to-End Encryption (E2EE):** Ensure secure transmission of credentials and authentication data.
- **Prevent Replay Attacks:** Use unique cryptographic challenges and timestamps.

7. Privacy Preserving Mechanisms

In this chapter, we explore the privacy-preserving mechanisms integrated into the authentication system, ensuring that military personnel can securely authenticate themselves while minimizing the exposure of personal data. Given the highly sensitive nature of the data involved (e.g., military ranks, security clearance, personal identifiers), maintaining privacy is crucial.

7.1. Selective Disclosure

One of the core principles of the system is selective disclosure, which allows military personnel to share only the necessary attributes from their Verifiable Credentials (VCs) during authentication. For example, if a user needs to prove their security clearance level for accessing a specific section of the base, they can present only that attribute, without revealing other personal details such as rank, division, or health information.

While the principle of selective disclosure is a core objective, it is important to clarify the practical implementation in the current prototype. Ideally, technologies like Zero-Knowledge Proofs (ZKPs) or advanced signature schemes such as BBS+ signatures allow for cryptographic proofs of a subset of attributes without revealing the entire credential. This would mean that only the specific, requested attributes (or a proof of their existence) are transmitted and validated, while the rest of the VC remains entirely private to the Holder.

However, in this implementation, due to the project's scope and time constraints, a full, cryptographically enforced selective disclosure at the protocol level (e.g., proving only a subset of attributes without sending the full VC) was not fully realized. Instead, the system primarily focuses on application-level selective disclosure:

The Verifier's logic is designed to extract and process only the specific attributes required for a given Level of Assurance (LoA) access request. For example, for an LoA 1 access, the Verifier might only computationally verify the presence of a "base access" attribute and the overall validity of the VC's signature, without logically processing or storing other personal data contained within the VC.

The primary challenge encountered in implementing true cryptographic selective disclosure was the inherent requirement for the Verifier to access the complete Verifiable Credential in order to verify the Issuer's cryptographic signature. The integrity of the entire VC document, as attested by the Issuer, is validated as a single unit. Without a more advanced cryptographic primitive (like a zero-knowledge proof of knowledge for specific attributes within a signed credential), the Verifier must receive and computationally process the full VC to ensure its authenticity and that it hasn't been tampered with since issuance.

Therefore, while the system conceptually operates on the principle of minimizing revealed data (by Verifiers only using what's necessary), the underlying technical implementation for signature verification still requires the transmission of the full VC.

7.2. Zero-Knowledge Proofs

Zero-Knowledge Proofs (ZKPs) play a key role in preserving privacy while ensuring that the Holder's data is verified without revealing the underlying information. This cryptographic technique allows one party (the Holder) to prove to another party (the Verifier) that they possess certain credentials or attributes, such as security clearance, without revealing the actual values or other sensitive information

In the context of Verifiable Credentials, ZKPs would ideally enable a Holder to demonstrate specific attributes or properties derived from their VCs (e.g., "I am over 18," "I have Top Secret clearance," "I am part of Unit X") without disclosing the actual birthdate, the specific clearance level string, or the full unit identifier. This directly addresses the privacy objective by allowing verification without unnecessary data exposure, truly embodying the "privacy-preserving" aspect of the system.

While the implementation of full, generic Zero-Knowledge Proof schemes (such as zk-SNARKs or zk-STARKs) for arbitrary attribute proofs within a Verifiable Credential was outside the scope and time constraints of this prototype, the system incorporates a form of Zero-Knowledge Proof within its challenge-response authentication mechanism.

Specifically, for both Low and Substantial Levels of Assurance (LoA), the authentication flow involves the following:

- Challenge Generation: The Verifier generates a random cryptographic challenge.
- Proof of Knowledge (via Signature): The Holder, using their securely stored (and, for Substantial LoA, PIN-protected) private key, cryptographically signs this challenge.
- Verification: The Holder sends this signature back to the Verifier. The Verifier, using the Holder's publicly registered key (retrieved via DID resolution from the Blockchain), verifies the signature.

In this process, the Verifier confirms that the Holder possesses the correct private key (and, implicitly, the associated credential claims that the key is authorized to prove) without the private key ever leaving the Holder's smart card or being revealed to the Verifier. The Verifier learns that the Holder knows the secret (the private key) and can use it to sign the challenge, but not what the secret is. This satisfies the fundamental definition of a Zero-Knowledge Proof: knowledge of a secret is proven without revealing the secret itself.

8. Implementation

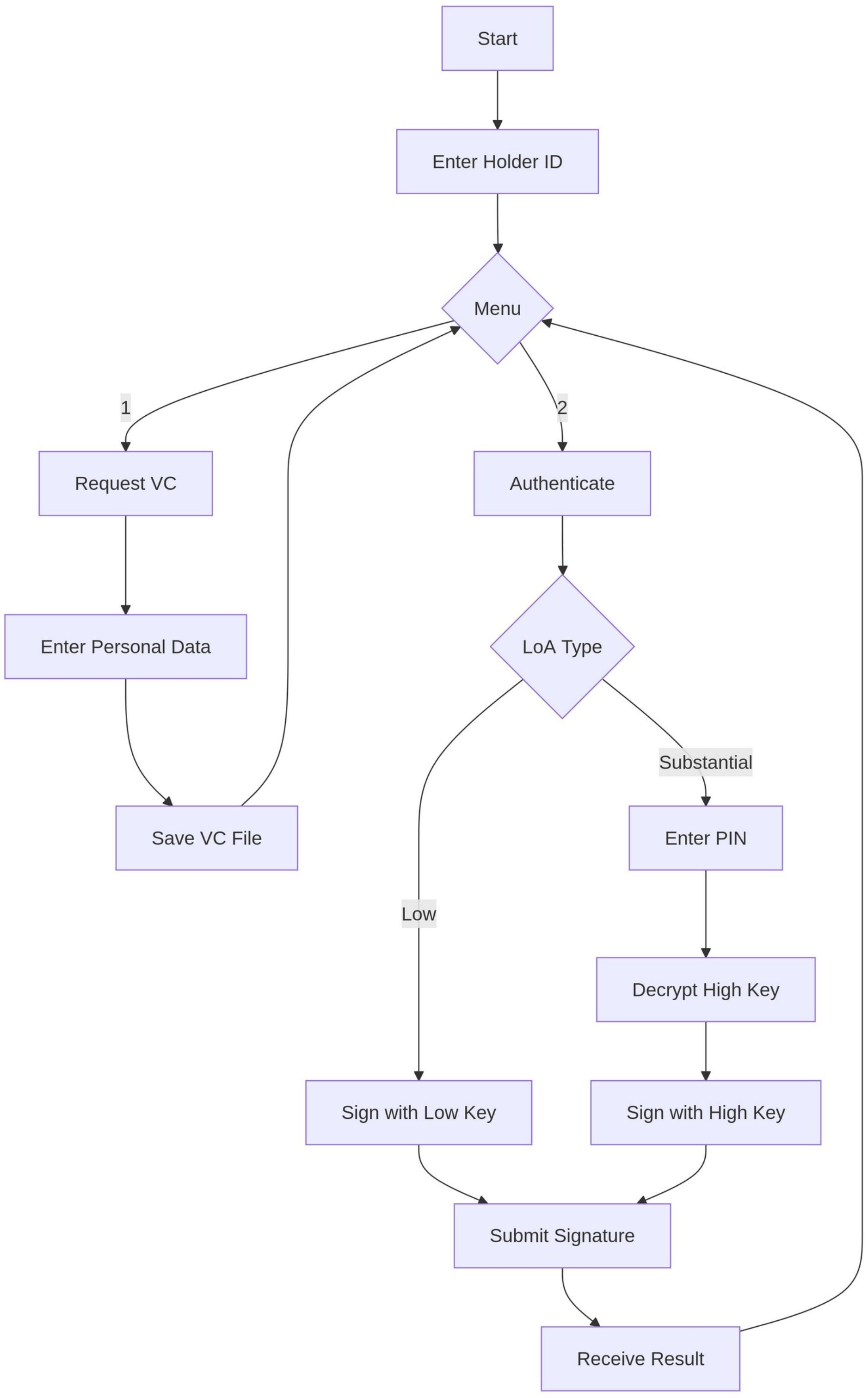
In order to implement the project, we split the logic into multiple programs: holder, verifier, interface, wallet, issuer and blockchain. The requests library was used to interact between all of these.

8.1. Holder

The holder.py script implements the Holder component of our authentication system, representing military personnel who possess and use Verifiable Credentials (VCs) for secure access. This Python-based component provides two core functionalities: VC acquisition and authentication handling. When a user selects "Get VC" in the command-line interface, the script collects personal data (name, nationality, health code) and sends it to the Issuer via a POST request to /register_holder. The Issuer responds with a digitally signed Verifiable Credential containing both low-assurance and high-assurance cryptographic keys, along with a PIN for accessing the high-assurance key. The script securely stores this VC in a JSON file while removing the PIN from the stored data for security.

For authentication, when the user selects "Authenticate," the script initiates a challenge-response process with the Interface component. It first requests an authentication challenge from /auth_req, receiving both a cryptographic challenge and the required Level of Assurance (LoA). For substantial LoA access, the script prompts the user to enter their PIN, which is used to decrypt the high-assurance private key stored in the VC. For low LoA access, it directly uses the unencrypted low-assurance key. The script then hashes the challenge using SHA-256 and signs it using RSA-PSS padding with the appropriate private key. This signed challenge is sent to the Interface via /send_challenge_to_verifier for verification.

The implementation incorporates several critical security features. The high-assurance private key remains encrypted in storage and only becomes accessible through PIN-based decryption, protecting against credential theft. Cryptographic operations use industry-standard SHA-256 hashing and RSA-PSS signing with 2048-bit keys.

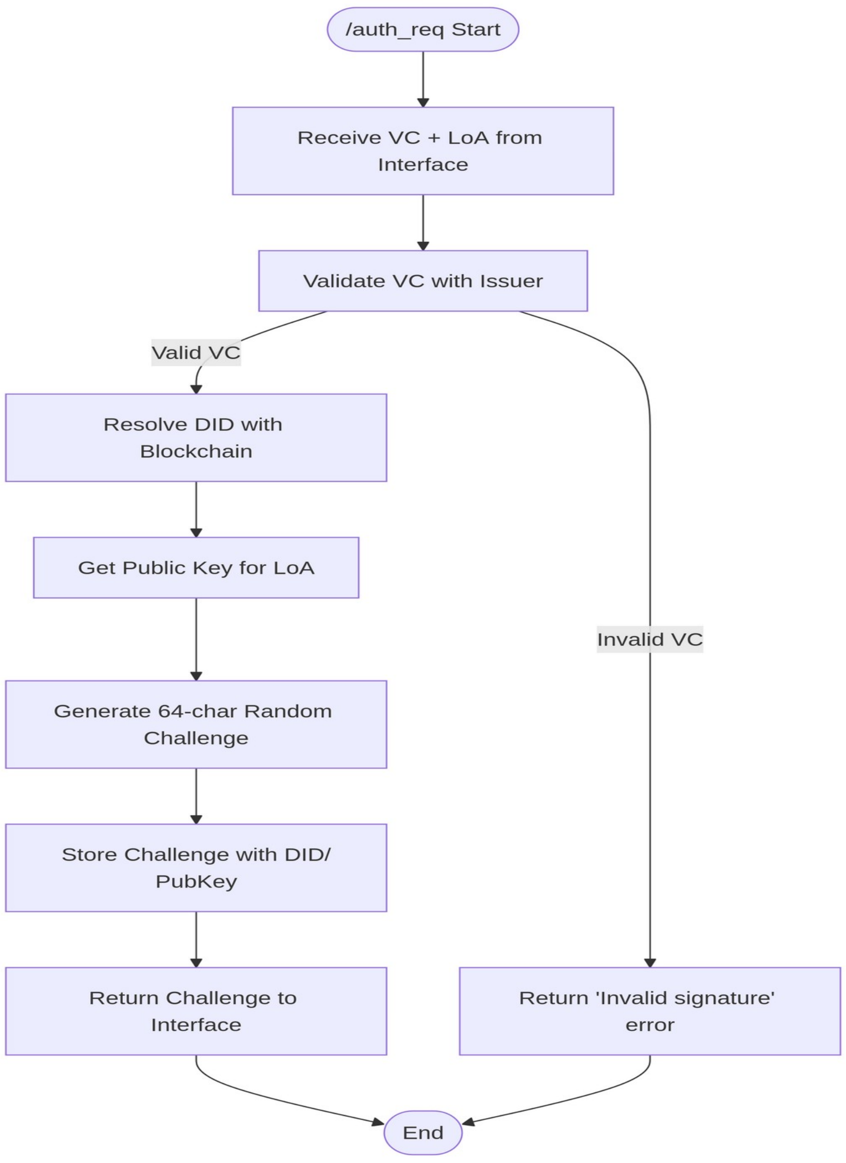


8.2. Verifier

The verifier.py script implements the Verifier component of our authentication system, serving as the cryptographic validation service that authenticates holders' credentials through a secure challenge-response protocol. This Flask-based application provides two critical endpoints that form the core of the verification process:

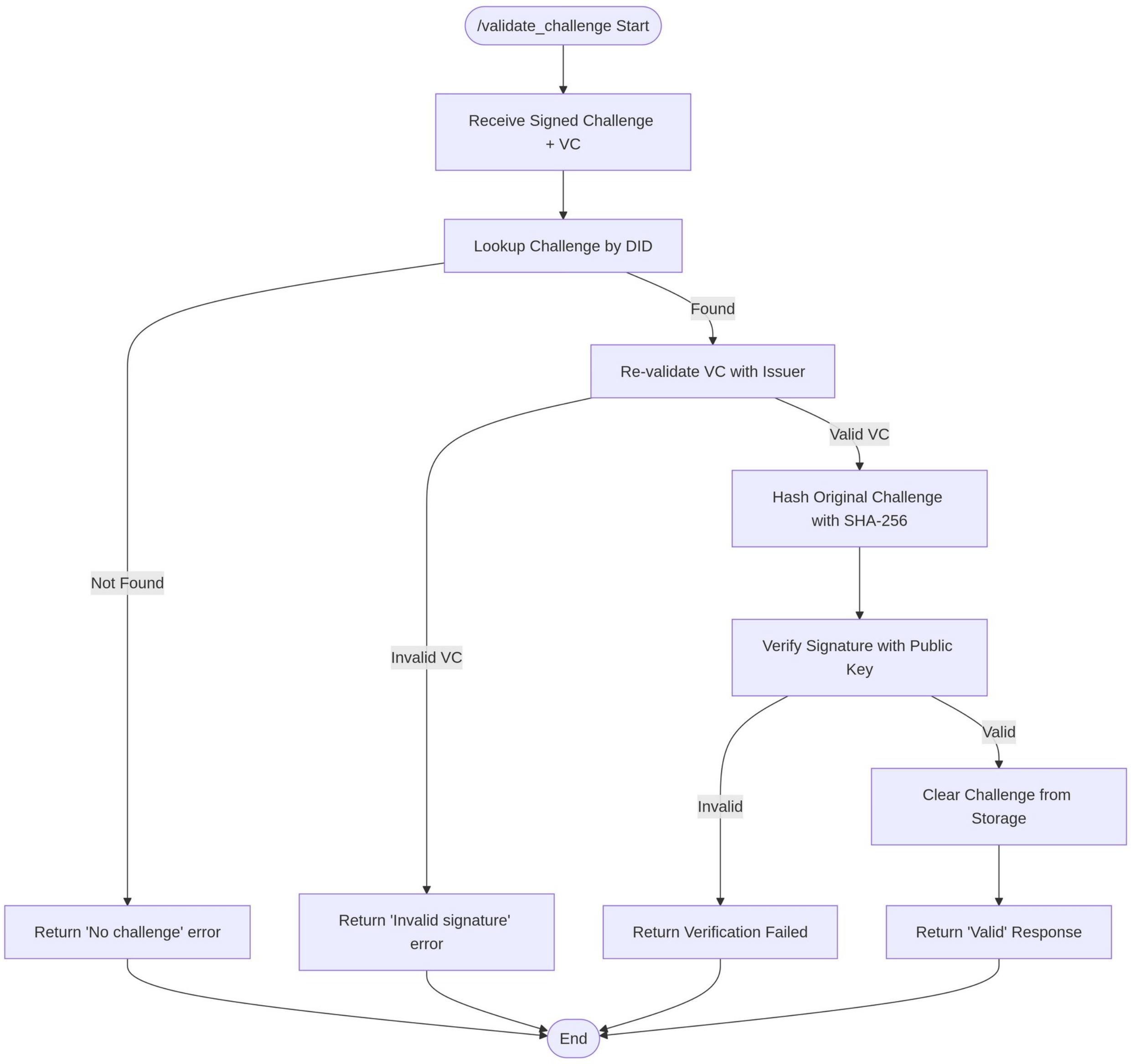
- **Challenge Generation Endpoint (/auth_req):** This endpoint serves as the entry point for authentication requests, initiating the cryptographic challenge-response process. When the Interface component forwards a holder's Verifiable Credential (VC) and requested Level of Assurance (LoA), the Verifier first validates the credential's integrity by sending it to the Issuer's validation service. This critical step ensures the VC hasn't been revoked, remains within its validity period, and bears a legitimate signature from the trusted Issuer. Upon successful validation, the Verifier resolves the holder's public key by querying the Blockchain registry with the Decentralized Identifier (DID) contained in the VC, specifically requesting the key corresponding to the required LoA level (either low or substantial assurance).

  The core security mechanism then activates as the Verifier generates a unique 64-character cryptographic challenge using cryptographically secure randomization. This challenge is temporarily stored in memory along with the holder's DID and public key, creating a verifiable session context. The newly generated challenge is returned to the Interface, which forwards it to the Holder for signing. This entire process establishes a secure foundation for authentication by ensuring only valid credentials can initiate challenges while binding each challenge to a specific identity and assurance level through the DID-public key association.


/auth_req Start
Receive VC + LoA from Interface
Validate VC with Issuer
Valid VC
Resolve DID with Blockchain
Get Public Key for LoA
Generate 64-char Random Challenge
Store Challenge with DID/ PubKey
Return Challenge to Interface
Invalid VC
Return 'Invalid signature' error
End

- **Signature Verification Endpoint (/validate_challenge):** This endpoint completes the authentication loop by validating the Holder's cryptographic proof of identity. When the Interface submits a signed challenge along with the original VC, the Verifier first retrieves the challenge context by looking up the holder's DID in its temporary storage. If no matching challenge is found (indicating a potential replay attack or session timeout), it immediately returns an error. For valid sessions, the Verifier performs a crucial second validation of the VC with the Issuer - a security measure that detects potential credential revocation occurring between challenge issuance and verification.

  The verification process then enters its cryptographic phase: the original challenge is hashed using SHA-256 to recreate the exact digest that should have been signed. The Verifier loads the PEM-formatted public key from the stored session context and verifies the signature using RSA-PSS padding with MGF1 and SHA-256 parameters. Successful verification confirms two critical facts: that the Holder possesses the private key corresponding to their public credential, and that the authentication response was specifically generated for this session's challenge. The endpoint finally returns a validation result to the Interface, which enforces access control decisions based on this cryptographic proof of identity.

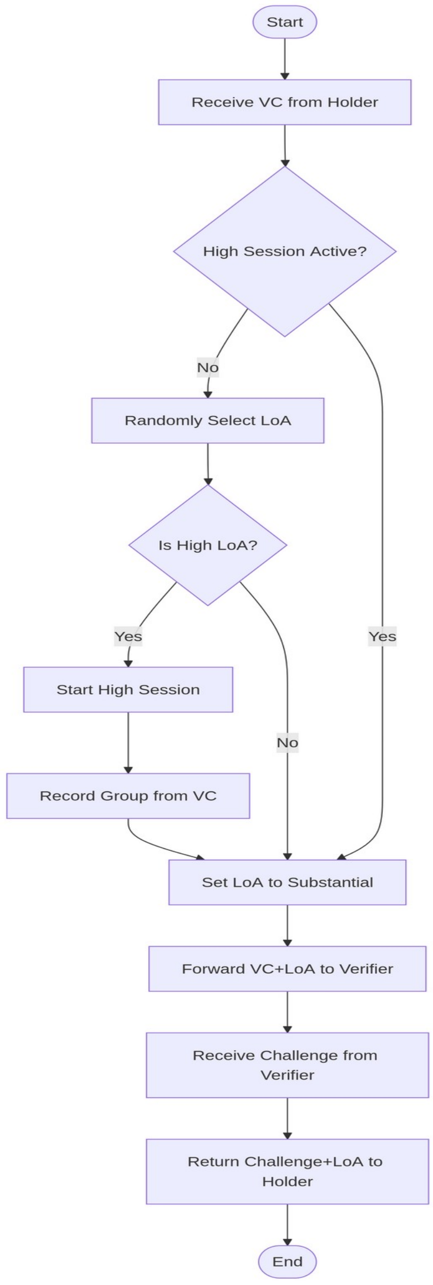

/validate_challenge Start
Receive Signed Challenge + VC
Lookup Challenge by DID
Found
Not Found
Re-validate VC with Issuer
Valid VC
Invalid VC
Hash Original Challenge with SHA-256
Verify Signature with Public Key
Valid
Invalid
Clear Challenge from Storage
Return 'No challenge' error
Return 'Invalid signature' error
Return Verification Failed
Return 'Valid' Response
End

8.3. Interface

The interface.py script implements the Interface component that acts as the central orchestrator between Holders and Verifiers while managing complex authentication scenarios. This Flask-based application handles two critical endpoints with specialized logic for high-security group authentication:

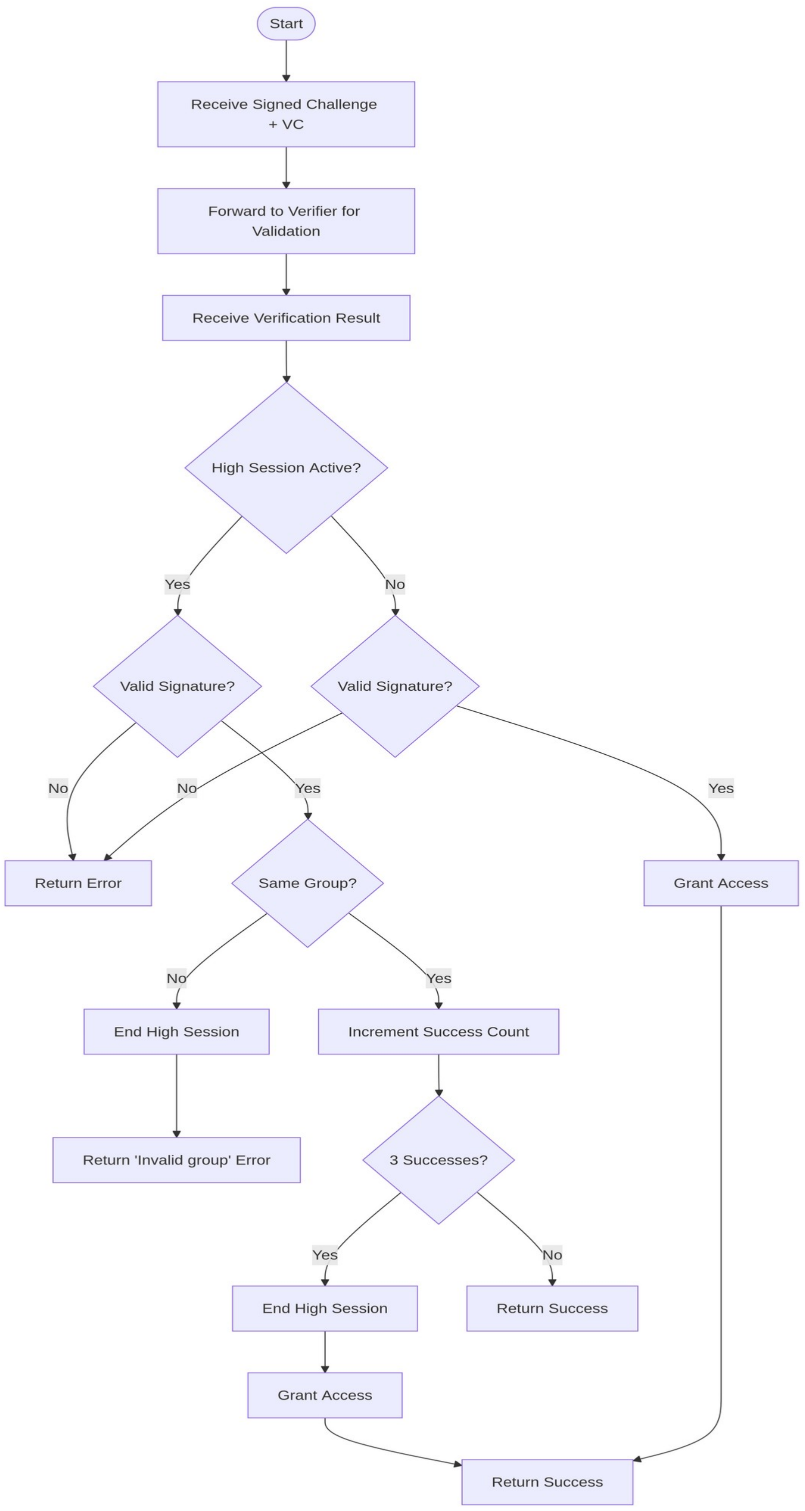
- **Authentication Request Endpoint (/auth_req):** This endpoint serves as the entry point for all authentication requests, dynamically determining the required Level of Assurance (LoA) based on the current security context. When a Holder presents their Verifiable Credential (VC) by tapping their smart card, the interface first checks if a high-security group authentication session is already in progress. If no session is active, it randomly selects an LoA (low, substantial, or high) to simulate different access scenarios within the military base environment.

  For high-security requests (LoA 3), the interface initiates special session management: it records the first Holder's original_national_base attribute as the required group identifier, sets the LoA to substantial (since high-security access requires multiple substantial authentications), and initializes a success counter. The VC and determined LoA are then forwarded to the Verifier's /auth_req endpoint, which returns a cryptographic challenge. This challenge, along with the LoA information, is finally returned to the Holder. The interface essentially acts as a policy enforcement point, translating physical access requests into appropriate authentication workflows while maintaining state for group-based access control.



- **Challenge Response Endpoint (/send_challenge_to_verifier):** This endpoint processes the signed cryptographic challenges from Holders and enforces the military's group authentication policies. Upon receiving a signed challenge and VC, the interface first delegates cryptographic verification to the Verifier's /validate_challenge endpoint.

  The core complexity emerges during high-security sessions (LoA 3), where the interface performs critical additional checks: it verifies that the Holder belongs to the same original_national_base group that initiated the session. Valid same-group authentications increment a success counter, and when three successful authentications occur, access is granted and the session terminated. Group mismatches immediately abort the session with an "Invalid group" error.



8.4. Wallet

The wallet.py script implements the Wallet as a secure credential registry. This Flask service provides a single /register_vc endpoint that receives and stores Verifiable Credential metadata from the Issuer. When credentials are issued, the Wallet persists critical military ID attributes including base assignment, rank, clearance level, and personal identifiers.

The service initializes its database at startup using a predefined schema (wallet.sql), ensuring proper structure for credential storage. By maintaining an immutable record of all issued credentials without storing private keys, the Wallet serves as the system's authoritative audit trail while preserving privacy through strict data minimization.

8.5. Issuer

The issuer.py script implements the Issuer component, serving as the cryptographic trust anchor and credential authority within our military authentication system. This Flask-based application fulfills two critical roles through dedicated endpoints, functioning as a simulated Qualified Trust Service Provider (QTSP) aligned with eIDAS 2.0 principles.

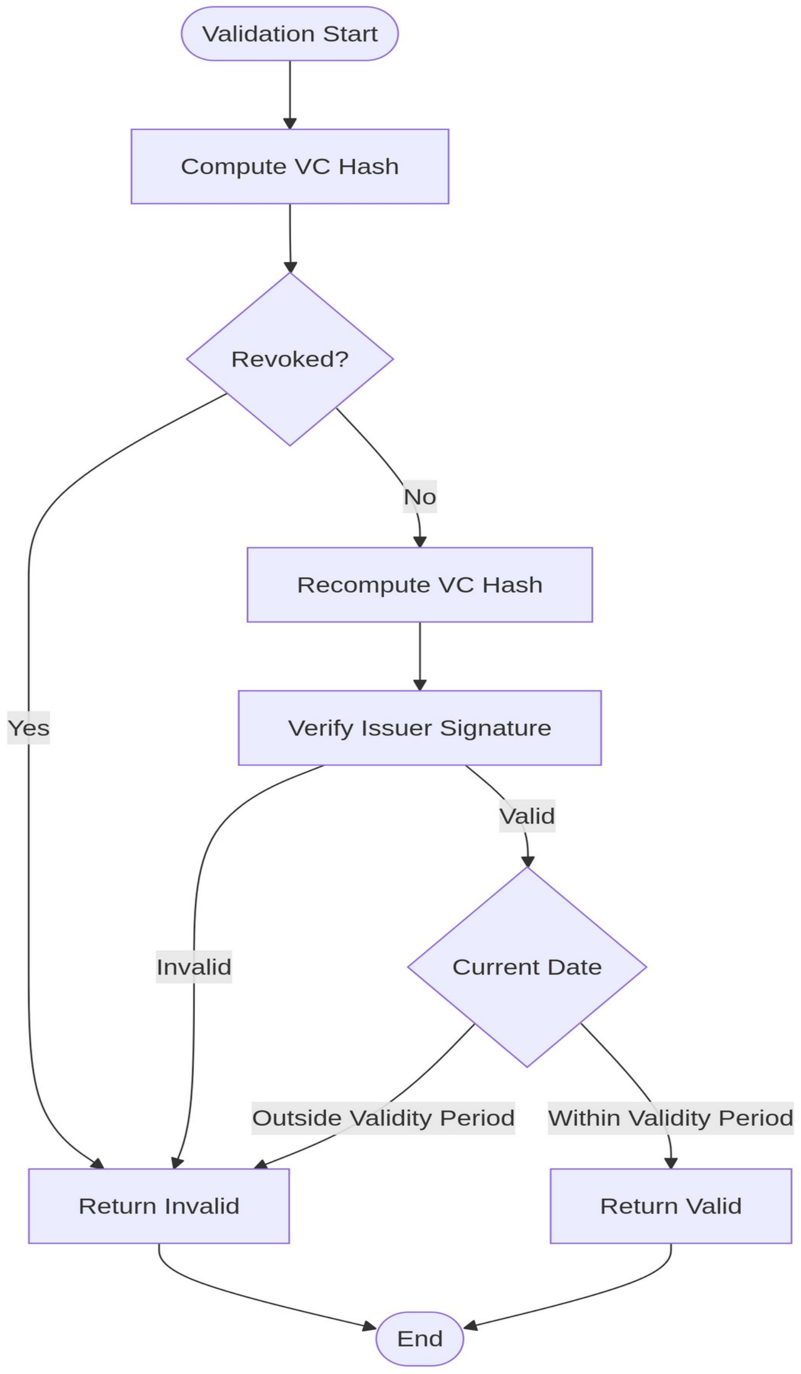
- **Credential Issuance Endpoint (/register_holder):** The /register_holder endpoint orchestrates the complete lifecycle of Verifiable Credential creation for military personnel. When a new holder (such as Cadet Alex in the Eagle Base scenario) submits their personal data, the endpoint initiates a secure cryptographic sequence. First, it generates two distinct RSA-2048 key pairs: one unencrypted key for low-assurance access (dining halls, offices) and another PIN-protected key for substantial/high-security areas (server rooms, command centers). The substantial private key undergoes AES encryption using a randomly generated 6-digit PIN, establishing the two-factor authentication requirement mandated for sensitive zones.

  These public keys are immediately dispatched to the Blockchain component, which returns a unique Decentralized Identifier (DID) to cryptographically bind the holder's identity to their keys. The endpoint then constructs a comprehensive military credential by combining system-assigned attributes (base assignment, rank, division, and security clearance from predefined lists) with the holder's personal data (name, nationality, health code).

  Crucially, the endpoint then creates a tamper-proof Verifiable Credential by hashing the complete payload with SHA-256 and signing it with the Issuer's private key using RSA-PSS padding. This cryptographic seal ensures any future tampering will be detectable during verification. The final output containing the signed credential, encrypted keys, and access PIN - is securely delivered to the holder, completing a trusted issuance process that transforms raw identity data into a privacy-preserving military credential aligned with eIDAS QTSP standards.

- **Credential Validation Endpoint (/check_vc_validity):** The /check_vc_validity endpoint serves as the cryptographic gatekeeper during authentication operations, performing three critical verification checks on every presented credential. When a Verifier submits a credential (like when Cadet Alex accesses the server room), the endpoint first computes the credential's SHA-256 hash and checks it against the Certificate Revocation List (CRL) database. This immediate revocation check prevents compromised credentials from being used, such as those revoked due to security incidents or personnel discharge.

  For non-revoked credentials, the endpoint proceeds to cryptographic authentication: it rehashes the core credential data and verifies the Issuer's digital signature using RSA-PSS padding with identical parameters to those used during signing. Finally, the endpoint performs temporal validation by comparing the current date against the credential's activation and expiration dates, blocking both expired credentials and those not yet active.



8.6. Blockchain

The blockchain.py script implements the Blockchain component as a simulated decentralized identity registry within our authentication system. This Flask-based application provides two essential endpoints that manage the lifecycle of Decentralized Identifiers (DIDs) and their cryptographic bindings:

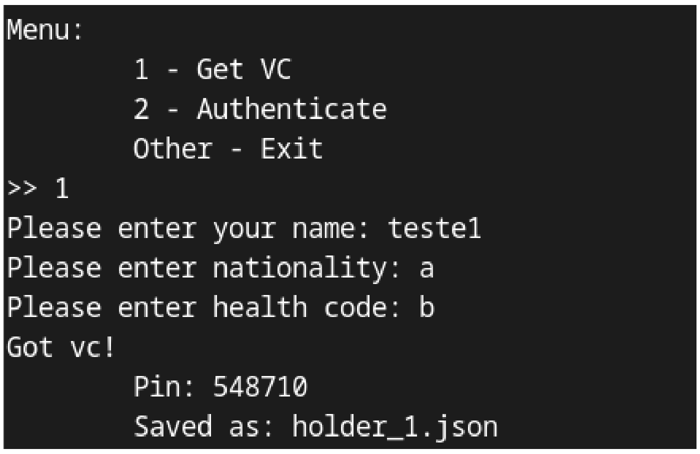
- **DID Registration Endpoint (/register_did):** When the Issuer generates new credentials, it invokes this endpoint to establish a trusted cryptographic identity. The endpoint receives the holder's two public keys (for low and substantial LoA) and generates a unique 64-character DID using cryptographically secure randomization. The newly created DID is returned to the Issuer for embedding in the Verifiable Credential, creating an immutable binding between the holder's identity and their cryptographic keys. This process simulates blockchain-based identity registration by providing decentralized, tamper-resistant key management without actual distributed ledger complexity.
- **DID Resolution Endpoint (/resolve_did):** During authentication operations, verifiers call this endpoint to retrieve the cryptographic material needed for signature validation. The endpoint accepts a DID and Level of Assurance (LoA) specification, then queries the database. Based on the requested LoA, it returns either the low-assurance or substantial-assurance public key associated with the DID. This resolution service is critical for enabling challenge-response verification without centralized key storage, as it allows verifiers to dynamically obtain the correct public key while maintaining the decentralized trust model fundamental to our privacy-preserving authentication approach.

9. Usage

To use the system, a startup script has been made. This script start all the necessary services and a terminal to use the holder application manually.

9.1. Get VC

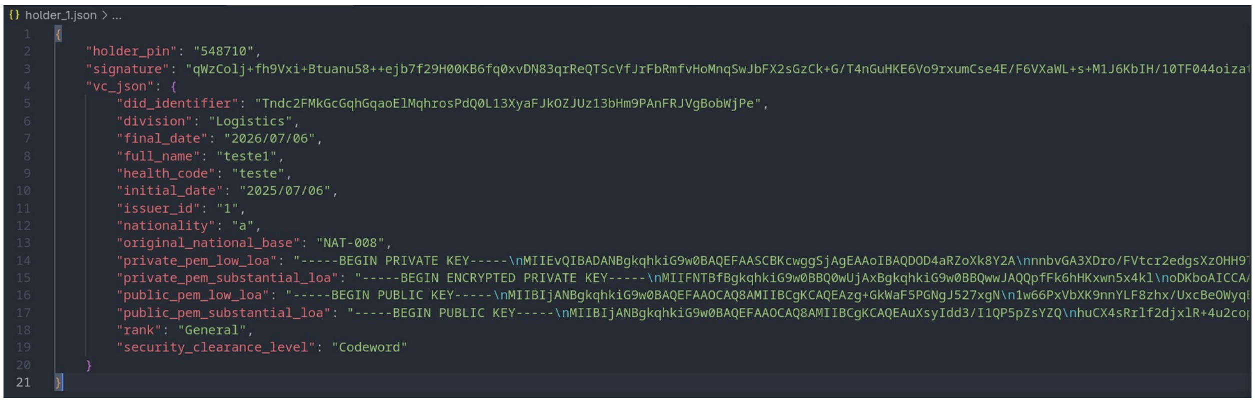
This topic presents the situation where a user requests their VC.



```
Menu:
        1 - Get VC
        2 - Authenticate
        Other - Exit
>> 1
Please enter your name: teste1
Please enter nationality: a
Please enter health code: b
Got vc!
        Pin: 548710
        Saved as: holder_1.json
```

Figure 1: Get VC

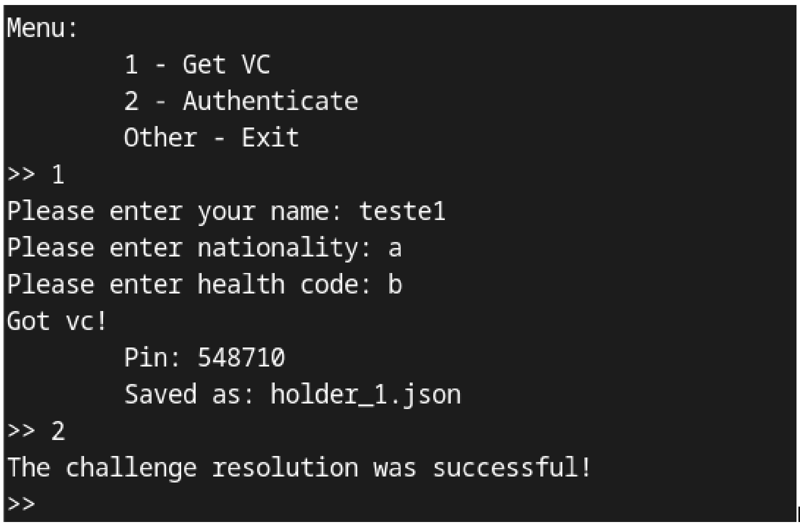
The resulting JSON file looks something like this:



```
{} holder_1.json > ...
{
    "holder_pin": "548710",
    "signature": "qWzColj+fh9Vxi+Btuanu58++ejb7f29H00KB6fq0xvDN83qrReQTScVfJrFbRmfvHoMnqSwJbFX2sGzCk+G/T4nGuHKE6Vo9rxumCse4E/F6VXaWL+s+M1J6KbIH/10TF044oiza
    "vc_json": {
        "did_identifier": "Tndc2FMkGcGqhGqaoElMqhrosPdQ0L13XyaFJkOZJUz13bHm9PAnFRJVgBobWjPe",
        "division": "Logistics",
        "final_date": "2026/07/06",
        "full_name": "teste1",
        "health_code": "teste",
        "initial_date": "2025/07/06",
        "issuer_id": "1",
        "nationality": "a",
        "original_national_base": "NAT-008",
        "private_pem_low_loa": "-----BEGIN PRIVATE KEY-----\nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDOD4aRZoXk8Y2A\nnnbvGA3XDro/FVtcr2edgsXzOHH9
        "private_pem_substantial_loa": "-----BEGIN ENCRYPTED PRIVATE KEY-----\nMIIFNTBfBgkqhkiG9w0BBQ0wUjAxBgkqhkiG9w0BBQwwJAQQpfFk6hHKxwn5x4kl\noDKboAICCA
        "public_pem_low_loa": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzg+GkWaF5PGNgJ527xgN\n1w66PxVbXK9nnYLF8zhx/UxcBeOWyq
        "public_pem_substantial_loa": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuXsyIdd3/I1QP5pZsYZQ\nhuCX4sRrlf2djxlR+4u2co
        "rank": "General",
        "security_clearance_level": "Codeword"
    }
}
```

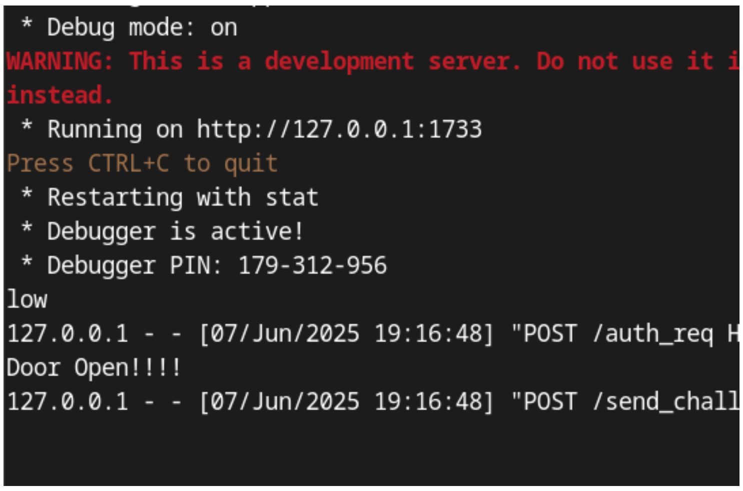
9.2. Low LoA

In this topic, the low LoA situation is presented by the interface to the holder.



```
Menu:
        1 - Get VC
        2 - Authenticate
        Other - Exit
>> 1
Please enter your name: teste1
Please enter nationality: a
Please enter health code: b
Got vc!
        Pin: 548710
        Saved as: holder_1.json
>> 2
The challenge resolution was successful!
>>
```

Fig
ure 2: Low LoA



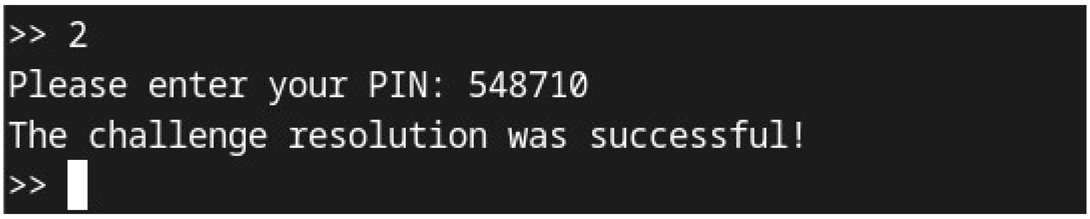
```
 * Debug mode: on
WARNING: This is a development server. Do not use it i
instead.
 * Running on http://127.0.0.1:1733
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 179-312-956
low
127.0.0.1 - - [07/Jun/2025 19:16:48] "POST /auth_req H
Door Open!!!!
127.0.0.1 - - [07/Jun/2025 19:16:48] "POST /send_chall
```

Figure 3: Low LoA interface output

From this pictures, it is possible to see that the chosen LoA was “low”, the PIN was not required and that the door was opened.

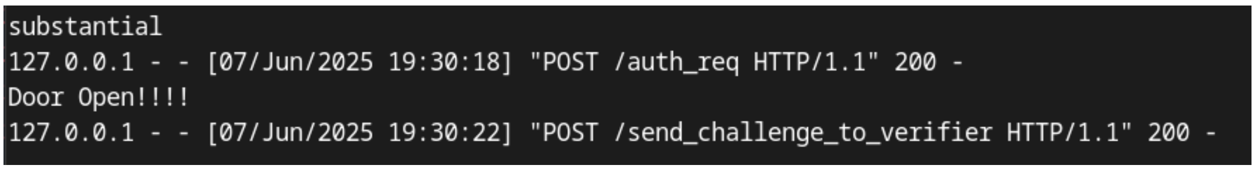
9.3. Substancial LoA

The substancial LoA is presented next.



```
>> 2
Please enter your PIN: 548710
The challenge resolution was successful!
>> 
```

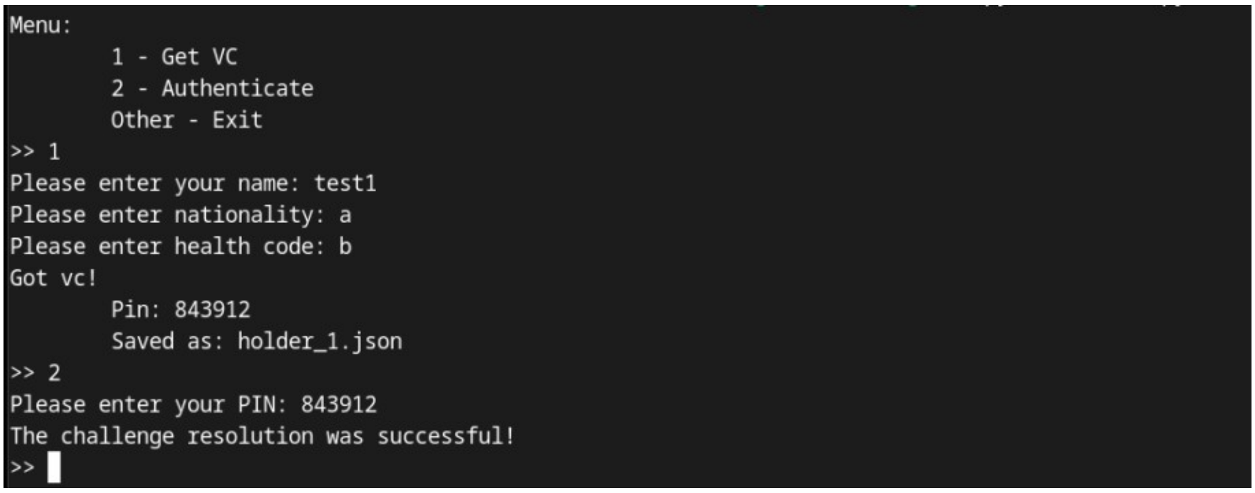
And the output from the interface.



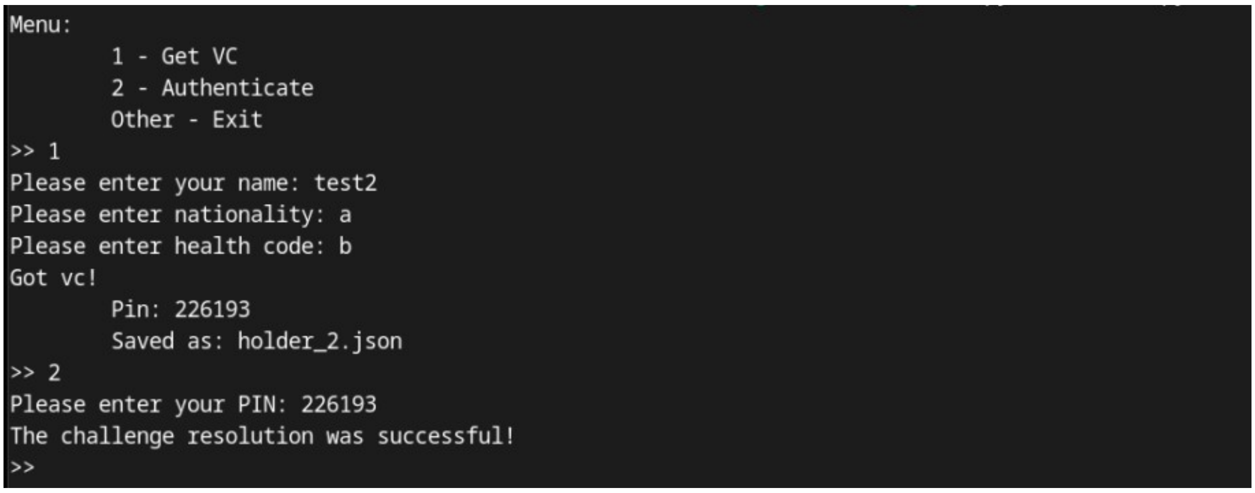
```
substantial
127.0.0.1 - - [07/Jun/2025 19:30:18] "POST /auth_req HTTP/1.1" 200 -
Door Open!!!!
127.0.0.1 - - [07/Jun/2025 19:30:22] "POST /send_challenge_to_verifier HTTP/1.1" 200 -
```

9.4. High LoA

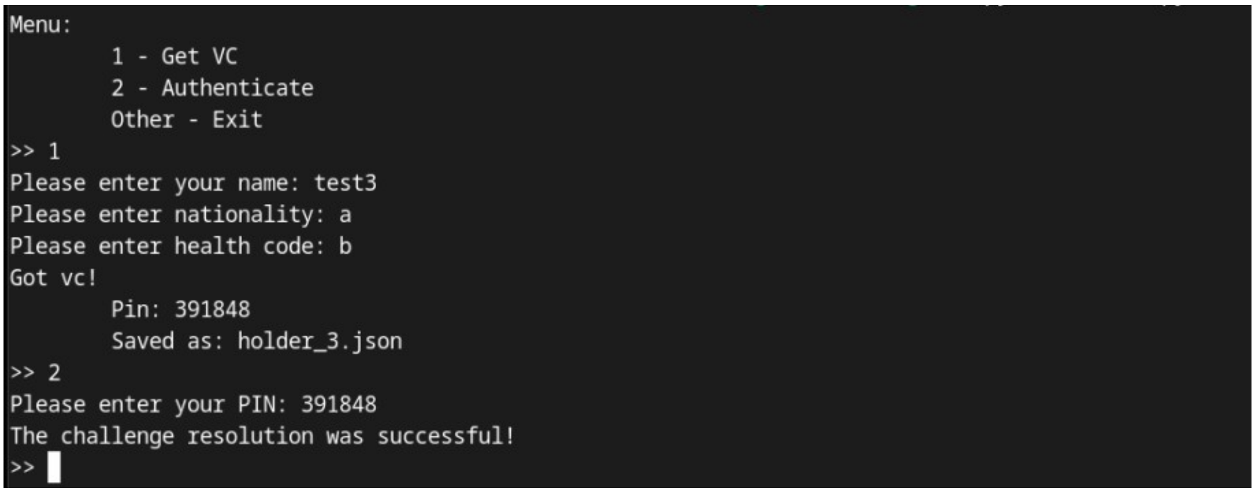
Now, the High LoA is presented. There will be three users (test1, test2 and test3)



```
Menu:
        1 - Get VC
        2 - Authenticate
        Other - Exit
>> 1
Please enter your name: test1
Please enter nationality: a
Please enter health code: b
Got vc!
        Pin: 843912
        Saved as: holder_1.json
>> 2
Please enter your PIN: 843912
The challenge resolution was successful!
>> 
```

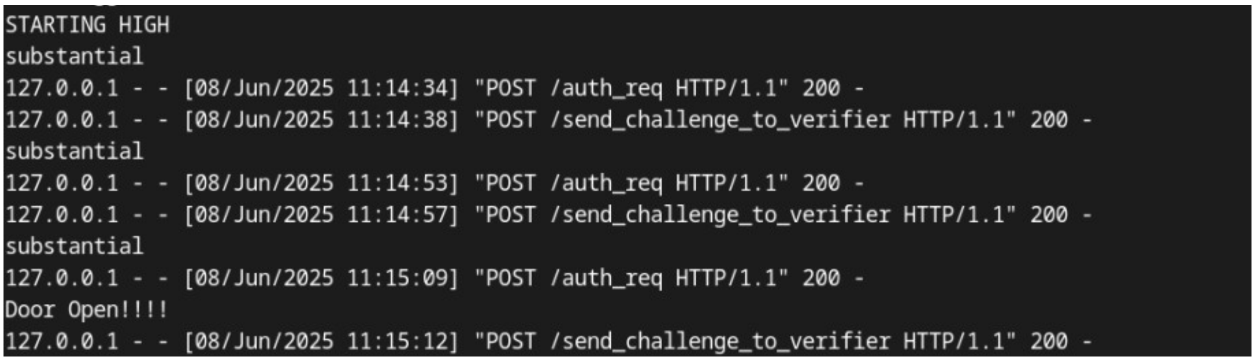


```
Menu:
        1 - Get VC
        2 - Authenticate
        Other - Exit
>> 1
Please enter your name: test2
Please enter nationality: a
Please enter health code: b
Got vc!
        Pin: 226193
        Saved as: holder_2.json
>> 2
Please enter your PIN: 226193
The challenge resolution was successful!
>>
```



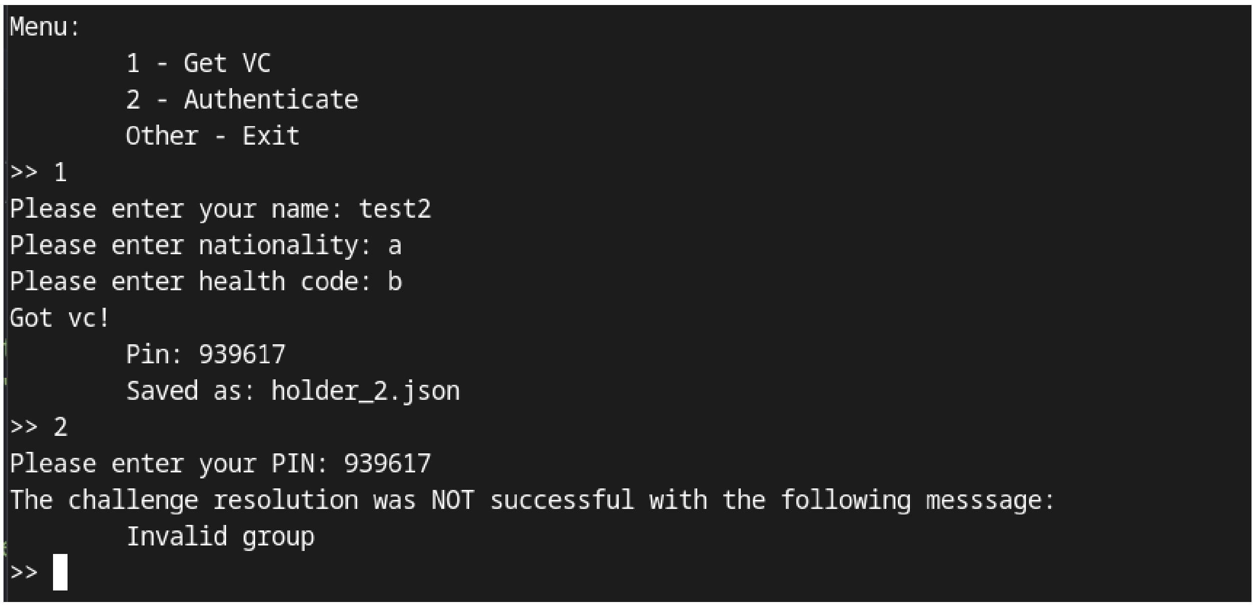
```
Menu:
        1 - Get VC
        2 - Authenticate
        Other - Exit
>> 1
Please enter your name: test3
Please enter nationality: a
Please enter health code: b
Got vc!
        Pin: 391848
        Saved as: holder_3.json
>> 2
Please enter your PIN: 391848
The challenge resolution was successful!
>>
```

And the output from the interface.



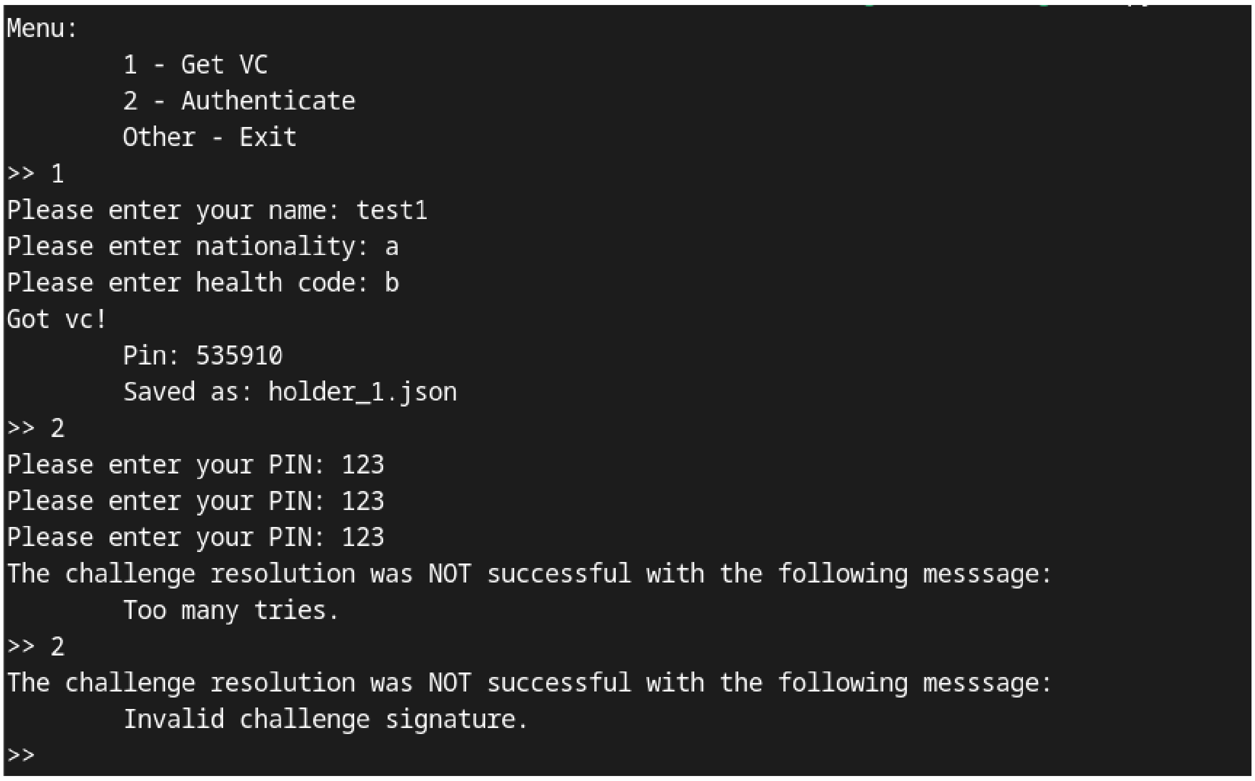
```
STARTING HIGH
substantial
127.0.0.1 - - [08/Jun/2025 11:14:34] "POST /auth_req HTTP/1.1" 200 -
127.0.0.1 - - [08/Jun/2025 11:14:38] "POST /send_challenge_to_verifier HTTP/1.1" 200 -
substantial
127.0.0.1 - - [08/Jun/2025 11:14:53] "POST /auth_req HTTP/1.1" 200 -
127.0.0.1 - - [08/Jun/2025 11:14:57] "POST /send_challenge_to_verifier HTTP/1.1" 200 -
substantial
127.0.0.1 - - [08/Jun/2025 11:15:09] "POST /auth_req HTTP/1.1" 200 -
Door Open!!!!
127.0.0.1 - - [08/Jun/2025 11:15:12] "POST /send_challenge_to_verifier HTTP/1.1" 200 -
```

If the user isn’t in the group necessary the following error will appear.


```
Menu:
        1 - Get VC
        2 - Authenticate
        Other - Exit
>> 1
Please enter your name: test2
Please enter nationality: a
Please enter health code: b
Got vc!
        Pin: 939617
        Saved as: holder_2.json
>> 2
Please enter your PIN: 939617
The challenge resolution was NOT successful with the following messsage:
        Invalid group
>>
```

9.5. Revocation

If a user fails the PIN more then 3 times, the VC gets revoked and the message “Too many tries” appears, and if the user tries to use it again the error “Invalid Challenge Signature” appears due to the signature not being valid since it was revoked


```
Menu:
        1 - Get VC
        2 - Authenticate
        Other - Exit
>> 1
Please enter your name: test1
Please enter nationality: a
Please enter health code: b
Got vc!
        Pin: 535910
        Saved as: holder_1.json
>> 2
Please enter your PIN: 123
Please enter your PIN: 123
Please enter your PIN: 123
The challenge resolution was NOT successful with the following messsage:
        Too many tries.
>> 2
The challenge resolution was NOT successful with the following messsage:
        Invalid challenge signature.
>>
```

10. Critical Analysis of the Use Case

Strengths of the Implemented System

The authentication system successfully delivers a functional prototype that demonstrates core eIDAS 2.0 and Verifiable Credentials principles within a military context. Its modular architecture - clearly separating Holder, Verifier, Issuer, and Blockchain components - enables robust access control with three distinct Levels of Assurance (LoA). The implementation effectively handles complex military requirements including group authentication (LoA 3), where three personnel from the same base must authenticate simultaneously for high-security access. Cryptographic security is maintained through RSA-2048 signatures with PSS padding and SHA-256 hashing, while PIN protection for substantial/high LoA provides two-factor authentication. The system also implements a comprehensive credential lifecycle: issuance, selective disclosure, revocation checking, and expiry validation. Privacy preservation is achieved through minimal data exposure during authentication and application-level selective disclosure, aligning with GDPR principles despite cryptographic limitations.

Limitations and Technical Constraints

Several planned features remain unimplemented due to project constraints. The absence of Zero-Knowledge Proofs (ZKPs) means true cryptographic selective disclosure wasn't achieved, requiring full credential transmission during verification. RSA was prioritized over Elliptic Curve Cryptography (ECC) for familiarity despite ECC's performance advantages. The simulated blockchain uses a simple SQLite database rather than distributed ledger technology, limiting decentralization benefits. Communication between components occurs over unencrypted HTTP instead of TLS/HTTPS, creating potential man-in-the-middle vulnerabilities. No replay attack protection exists beyond session-based challenge tracking. The smart card emulation uses JSON files rather than secure hardware elements. A graphical interface would be essential for real-world deployment.

Security Considerations

While the cryptographic foundation is robust, several security aspects require enhancement:

1. **Key Management**: Private keys are stored in files rather than HSMs or secure enclaves
2. **PIN Protection**: No hardware-enforced retry limits or entropy requirements
3. **Communication Security**: Lack of TLS enables eavesdropping and tampering
4. **Revocation Propagation**: No mechanism to instantly invalidate sessions after revocation
5. **Side-Channel Protection**: No mitigation for timing/power analysis attacks

Practical Deployment Challenges

Military deployment would face operational hurdles:

- **Integration**: Compatibility with existing PKI and physical access systems
- **Fallback Mechanisms**: No redundancy for system failures during critical operations
- **Biometric Integration**: Hardware support for multi-factor authentication
- **Cross-Base Interoperability**: Standardization for NATO-wide deployment
- **Audit Compliance**: Enhanced logging for regulatory requirements

11. Conclusion

This project successfully demonstrated a functional implementation of a privacy-preserving authentication system based on Verifiable Credentials (VCs) and aligned with the principles of eIDAS 2.0. The system is designed to manage access control in military environments, offering varying Levels of Assurance (LoA) to balance security needs with usability and privacy requirements.

The architecture was modular and clearly structured, encompassing credential issuance, selective disclosure, revocation, and verification mechanisms. The implemented components Holder, Verifier, Issuer, Wallet, Interface, and Blockchain collectively enabled secure, end-to-end credential handling. Despite limited time, the team achieved a working system that illustrates the potential of decentralized digital identity for high-security contexts.

While several advanced features (e.g., Zero-Knowledge Proofs, secure communication protocols, replay protection) were not implemented, the system serves as a strong foundation for further development. These omitted features have been identified and documented, presenting clear opportunities for future enhancements.

Overall, the project met its primary goals, demonstrating not only technical viability but also strong alignment with regulatory frameworks. It also highlighted the importance of balancing security, privacy, and practical deployment considerations.

Percentage of Effort Devoted:

- **Carlos Ferreira** – 50%
- **Tomás Bogalho** – 50%