Department of Computer Science
Technical University of Cluj-Napoca

# Artificial Intelligence

*Laboratory activity*

Name: Mureșan Sebastian-Ioan
Group: 30434
Email: muresan.sebastian2000@gmail.com

Teaching Assistant: Andrei Dumitraș

# Contents

# Chapter 1

# Introduction

This Password Manager project is a practical implementation of AI principles in secure password management. It integrates **Prover9**, a reasoning system for automated theorem proving, and **Mace4**, a tool for model checking, to validate the logic of encryption and decryption.

Passwords are encrypted using a Caesar cipher, and their correctness is validated through logical proofs before being stored. Prover9 ensures that the encryption-decryption pipeline adheres to the defined logical axioms, while Mace4 checks for potential counterexamples, reinforcing the robustness of the system.

The project is designed for real-time demonstration, with changes to the local `passwords.txt` file reflecting user interactions, such as registration, password retrieval, and updates. This document explores the AI-driven mechanisms, their relevance, and the critical components of the Password Manager.

# Chapter 2

# AI-Driven Logic: Prover9 and Mace4

## 2.1 Prover9 for Logical Validation

Prover9 plays a critical role in this project, ensuring that the encryption and decryption operations are logically sound. It uses a carefully structured input file to validate axioms and prove the correctness of the encryption pipeline. The main axioms are:

- **Axiom 1**: `all x (decrypt(encrypt(x)) = x)`
  This ensures that decryption fully reverses the effect of encryption.
- **Axiom 2**: `all x (encrypt(decrypt(x)) = x)`
  This validates the reversibility of encryption.
- **Axiom 3 (Optional)**: Unique encryption logic (`x != y -> encrypt(x) != encrypt(y)`).

The goal is to prove that for any password, the following holds:

$$decrypt(encrypt(\texttt{password})) = \texttt{password}$$

## 2.2 Mace4 for Counterexample Generation

While Prover9 is used for theorem proving, **Mace4** complements this by searching for counterexamples to the axioms. If a counterexample exists, it suggests a logical flaw in the system. This two-fold approach of theorem proving and model checking ensures that the encryption-decryption logic is both provably correct and free from inconsistencies.

## 2.3 Dynamic Input File Generation

The Prover9 input file is dynamically generated based on the user-provided password, as shown below:

```python
def generate_prover9_input(password):
    with open("encryption_logic.in", "w") as file:
        file.write(f"""
        % Axioms for Encryption/Decryption
        formulas(assumptions).
        all x (decrypt(encrypt(x)) = x).
        all x (encrypt(decrypt(x)) = x).
        end_of_list.
        % Goal: Validate encryption and decryption
        formulas(goals).
        decrypt(encrypt("{password}")) = "{password}".
        end_of_list.
        """)
```

This ensures that the validation process adapts to each password entered by the user.

# Chapter 3

# Key Components

## 3.1 Encryption and Decryption

Encryption and decryption are implemented using a Caesar cipher, a simple yet effective method for this demonstration. The `EncryptionManager` class encapsulates these operations.

```
class EncryptionManager:
    @staticmethod
    def encrypt(password, key=3):
        return ''.join(chr(ord(char) + key) for char in password)

    @staticmethod
    def decrypt(encrypted_password, key=3):
        return ''.join(chr(ord(char) - key) for char in encrypted_password)
```

## 3.2 File Management

The `FileManager` class handles all interactions with the `passwords.txt` file. This file is updated in real-time, reflecting every user operation, including registration, password retrieval, and updates.

```
class FileManager:
    def read_users(self):
        users = {}
        with open("passwords.txt", "r") as file:
            for line in file:
                username, encrypted_password = line.strip().split(":")
                users[username] = encrypted_password
        return users

    def write_users(self, users):
        with open("passwords.txt", "w") as file:
            for username, encrypted_password in users.items():
                file.write(f"{username}:{encrypted_password}\n")
```

## 3.3 User Management

The `UserManager` class integrates encryption, file management, and Prover9 validation. It facilitates user registration, password retrieval, and updates.

```
class UserManager:
    def register(self, username, password):
        if validate_logic_with_prover9(password):
            encrypted_password = EncryptionManager.encrypt(password)
            users[username] = encrypted_password
        else:
            print("Prover9 validation failed. Registration aborted.")
```

# Chapter 4

# Usage and Relevance

## 4.1 Relevance

This Password Manager highlights the importance of AI in secure systems. By integrating Prover9 and Mace4, it not only ensures theoretical correctness but also provides practical security. The use of a local file (`passwords.txt`) for real-time updates makes it a tangible demonstration of AI in action.

## 4.2 Usage Scenarios

**Registration:** Users enter a username and password. The password is validated using Prover9, encrypted, and stored.

**View Password:** Users retrieve their decrypted password by entering their username.

**Change Password:** Users provide their old password for validation before updating it.

—

# Chapter 5

# Conclusion

The Password Manager project demonstrates the seamless integration of AI-driven tools like Prover9 and Mace4 with practical programming. It validates the encryption-decryption logic mathematically and checks for logical consistency, ensuring a robust and secure system.

The use of real-time file updates and dynamic input file generation showcases the practicality and flexibility of the system, making it an excellent example of applied artificial intelligence.

# Bibliography

Artificial Intelligence, Fall, 2024-2025

https://www.geeksforgeeks.org/