# SHA-256 Hashing Function Project
# ECRYP 23Z


**Authors**

Marcel Piotrowski - 303108
Aleksander Szweryn - 310723


December 10, 2023

# Contents

# 1 Theoretical introduction

## 1.1 Hashing

Before we start, it is important to distinguish between hashing and other types of functions in terms of cryptography. Hashing is a process of producing a fixed-sized string of characters from given input data. It is common to produce hexadecimal digest of an input string. There are two most important characteristics of hashing:

- Irreversibility - It should be computationally infeasible to reverse the process and obtain the original input from the hash value.
- Determinism - The same hashing function for the same input value should always produce the same result.

## 1.2 SHA-256

One of most popular example of hashing functions is SHA-256 (Secure Hash Algorithm 256-bit), which is a member of SHA-2 family of cryptographic hash functions. It was designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) as a part of the Digital Signature Algorithm (DSA) in 2001. SHA-256 is widely used for various cryptographic applications and is considered to be secure. The "256" name part indicates, that the output is of 256-bit length.

# 2 Functional description of the application

## 2.1 Usage

The designed usage of our application project is a Python3 command line interface (CLI) tool. The program requires installed Python environment, and additionally *pytest* module for running implemented test cases. Usually, preparation before using the application involves installing Python virtual environment (*venv*), enabling it, then downloading requirements using Python installation manager (*pip*). Example using Windows 11 command line below:

```
python -m venv env
env\Scripts\activate
pip install -r requirements.txt
```

Then to use the CLI tool:

```
python app_cli.py --help
python app_cli.py textToBeHashed
```

There is also a script version of the application, where user runs the script and is prompted to enter input text to be hashed:

```
python app.py
```

## 2.2 Input

Expected input is a string which theoretically should be of any length.

## 2.3 Output

The output will be given in a hexadecimal string representation of the digest.

# 3 Description of designed code structure

## 3.1 TODO - Alek

# 4 Tests

## 4.1 SHA-256

Our tests are conducted with usage of *pytest* and *hashlib* Python modules. The first one is for convenience of testing the source code, the second gives us some reference values of the hexadecimal SHA-256 digest to verify correctness of our algorithm implementation. All our test cases were successful and their parameters are provided below:

| INPUT | EXPECTED AND TESTED OUTPUT |
|---|---|
| "Hello, World!" | todo |
| "123456" | todo |
| "ECRYP CRYPT testing" | todo |
| "Hash Function Test" | todo |
| "python" | todo |
| "987654321" | todo |
| "abcdefghijklmnopqrstuvwxyz1234567890" | todo |

## 4.2 Operators

We have also prepared test cases for additional bit wise operators that we had to implement in Python. Those were also successful and provided below.

### 4.2.1 ROTATE RIGHT

| INPUT | EXPECTED AND TESTED OUTPUT |
|---|---|
| 0x00000010 rotate by 4 | 0x00000001 |
| 0x00000001 rotate by 4 | 0x10000000 |
| 0x000000F0 rotate by 4 | 0x0000000F |

### 4.2.2 CH - (a AND b) XOR (NOT a AND c)

| a | b | c | RESULT |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

### 4.2.3 MAJ - (a AND b) XOR (a AND c) XOR (b AND c)

| **a** | **b** | **c** | **RESULT** |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

# 5 List of sources


- Secure Hash Standard (SHS) - Federal Information Processing Standards Publication 180-4.