USENIX Security '25 Artifact Appendix: Leuvenshtein: Efficient FHE-based Edit Distance Computation with Single Bootstrap per Cell

Wouter Legiest, Jan-Pieter D'Anvers, Bojan Spasicm Nam-Luc Tran, Ingrid Verbauwhede

A Artifact Appendix

The following is an Artifact Appendix for the USENIX Security '25 conference, detailing the Leuvenshtein algorithm for efficient Fully Homomorphic Encryption (FHE)-based edit distance computation. This document provides a roadmap for evaluating the artifact, including hardware, software, and configuration requirements, as well as instructions for reproducing claims.

USENIX Security '25 Artifact Appendix: Leuvenshtein: Efficient FHE-based Edit Distance Computation with Single Bootstrap per Cell

A.1 Abstract

Our paper develops a new algorithm to efficiently calculate the edit distance on encrypted data using Fully Homomorphic Encryption. The artifact is a Rust project that implements the Leuvenshtein algorithm. The code is available via a GitHub link and a Zenodo repository. Both repositories are equivalent, but the GitHub repository is easier to clone due to its folder structure.

A.2 Description & Requirements

This section provides the necessary information to recreate the experimental setup used for this artifact. It includes the minimal hardware and software requirements , and details benchmarks used to produce the results.

A.2.1 Security, Privacy, and Ethical Concerns

There are no security, privacy, or ethical concerns as the artifact evaluation involves running a Rust or Python program.

A.2.2 How to Access

Our codebase can be accessed at: `https://github.com/WoutLegiest/leuvenshtein_ae`. For completeness, the code is also available at: `https://zenodo.org/records/15871491`. However, we recommend using the GitHub repository due to its support for a folder structure.

A.2.3 Hardware Dependencies

Our experiments were conducted on a dual AMD EPYC 9174F 16-Core Processor (64 threads in total) running Ubuntu 22.04, with 512 GiB RAM. As we are aiming only for the functional badge, the exact setup and outcomes are not critical.

A.2.4 Software Dependencies

Python: Versions 3.8 to 3.12. Rust: Version 1.80.

A.2.5 Benchmarks

Within the main file, example strings of lengths 8, 100, or 256 can be selected for testing.

A.3 Set-up

Our experiments were run using rustup version 1.80.

A.3.1 Installation

Rust Our experiments were conducted using Rust version 1.80. To install this specific Rust version, use `rustup install 1.80.0`. Further instructions can be found at `https://www.rust-lang.org/tools/install`. Rust can be installed using the following command:

Can be installed through:

- `curl -proto '=https' -tlsv1.2 -sSf https://sh.rustup.rs | sh`
- After installation, reload the terminal or source the Rust environment file as indicated in the final step of the installation. Then, install the specific Rust version:
- `rustup install 1.80.0`

The remaining dependencies will be downloaded by the Cargo tool.

Concrete The concrete library is a python library that needs to be installed with

1. Install the concrete library: `pip install concrete-python==2.8.1`
2. Clone the correct Concrete version `git clone https://github.com/zama-ai/concrete ; cd concrete ; git reset --hard fd9db128869818293d3b4336f44e5938cfe5c480`
3. Copy the patch file to the folder and apply our patch to the repo: `git apply ../concrete_ascii.patch`

A.3.2 Basic Test

The basis implementation can be run with

- `cargo +1.80 run -release`

Our preprocessing version can be run with:

- `cargo +1.80 run -release -bin leuvenshtein_preprocess`

This will run the algorithm and the plaintext algorithm and will time the encrypted execution.

Concrete

- Go to Levenshtein distance folder: `cd concrete/frontends/concrete-python/examples/levenshtein_distance/`
- Run the wanted size of concrete instance
  - `python3 levenshtein_distance.py --distance abcdefgh adcdefgf --alphabet ascii --max_string_length 8`
  - `python3 levenshtein_distance.py --distance BEgfEHGfShHtvKazXNeEvNWmvfbrAWyAYZjkXvNkmEajQNCTKZnkPeEDadvQtUnGhJRpWZASUMfXArGZFSUYgFeCAWxSvKNdpsnV EEhjZMnFsMFCsKnnyZvtrPeKxfmJfJVJNcYAwYmrGNgTUUSAgduNQZttWFdYFKddcKjkUEpPUmGkszZSVVNWkThxSFRgMzrbqATe --alphabet ascii --max_string_length 100`
  - `python3 levenshtein_distance.py --distance mizf30WE1Pzqu0HtZKMMCE6f3NE2TGPPYmgSunfkBJqGJqveg97i61fu5KG7z8UmR5DVVALk5CCL0fzEv687LdRuunZ8SYUFmQEf66dXZ6vejGKR7HhSiY5XLWbCYFddqtFEX2QjmHRmqB7tngfG7m0CBX1D6wcvLyYp0rpiv1GSw5T1ZfuT2mcjHUi05zd6N4EqLmTP8ETEc2vQaJcVGSaXRETejNebwwb4m9wUUMd0abrEQeLw3Ubn9un6tTeP yVL0hi1V8mUQakFWZGEHwQBduvaTtKC6dNMbnyEiU29pKdwLfCnY7jWceXQizTiwhGxiAkuwvdcgpTtaZW3XJ6t75HL86nV2QPUWaxipf6x7JE8NPQT0RrzcheaydjLdPyUhAQ3UhJ2bLVE5wtNDAgdBgX3N5Ru4iXqbFXWD5ZAbzniVaWr5iE0wQenwt8QjqERf6A67P9rLkmGPKS8LHJxttCKRBWM3qr1F93JZtEhGKcQ1079pUXcCgAvLhCWi --alphabet ascii --max_string_length 256`

A.4 Evaluation workflow

The basic workflow for processing two strings of length 8 is described in Section A.3.2. For larger string sizes, the main function needs to be adapted. In both (`main.rs` and `main_prepros.rs`) files, within the `fn main()` function, alternative inputs for ASCII encoding of lengths 100 and 256 are provided in comments. These must be uncommented to test them.

A.4.1 Major Claims

(C1): The outcome of the Leuvenshtein algorithm is the correct edit distance, calculated on encrypted data.

(C2): The execution of the Leuvenshtein algorithm is remarkably faster than the execution of the Concrete Compiler.

A.4.2 Experiments

Run the experiments as described in Section A.3.2 for string lengths 8, 100, and 256. Note that the 256-length input for Concrete is expected to take over a week to run and should not be completed. We anticipate that the decrypted outcome of Leuvenshtein will be identical to the outcome of the plaintext version and the Concrete implementation.

A.5 Notes on Reusability

The algorithm itself can be implemented in any industry context, provided the cryptographic scheme and parameters remain consistent.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.