Encrypted Edit Distance for Real-World Applications

Wouter Legiest, Jan-Pieter D'Anvers, and Ingrid Verbauwhede

COSIC KU Leuven, Belgium

Edit Distance

- Check the similarity of two strings a and b
- Count number of operations to transform a into b
- Common **Operations** in academic research:
  - **Insertion** of a character
  - **Deletion** of a character
  - **Substitution** of a character by another
  - **Transposition** of two adjacent characters
- Each edit distance has its allowed operations

Distance	Ins/Del	Subs	Trans
Hamming	○	●	○
Longest Common Subsequence	●	○	○
Levenshtein	●	●	○
Damerau-Levenshtein	●	●	●

- Ex.: Levenshtein distance of *'kitten'* and *'sitting'* is 3

Edit Distance in the FHE World

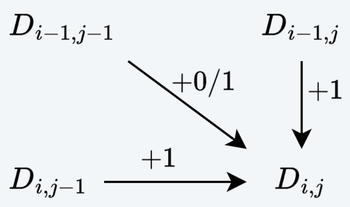
- Lots of the optimised plaintext edit distance algorithm uses and/or
  - data-depended branching
  - data-depended preprocessing
  - small-alphabet specific optimisations
  - optimised for unit-costs

$\Longrightarrow$ Two candidates:

- Wagner-Fisher algorithm
- bit-vector algorithm of Myers [1]
  - Only for unit costs operations; has $\mathcal{O}(n)$ time complexity

Wagner-Fisher Algorithm

- Textbook method to calculate edit distance: build up a matrix D
- Each element in the matrix is calculated as:
  $D_{i,j} = \min(D_{i-1,j} + 1; D_{i,j-1} + 1; D_{i-1,j-1} + (a_i == b_j))$
- Hard to parallelise; uses dynamic programming
  has $\mathcal{O}(n^2)$ time complexity



Real-World Requirements

- Non-unit cost of operations:
  - Insertion and Deletion: 2
  - Substitution: 0/1/2
- Alphabet using all numbers, lower and uppercase letters, and special characters, i.e. $|\Sigma| > 64$
- Strings length of $|a| = |b| = 132$
- Additional (symmetrical) operations, with reduced or zero cost
  - Substitution of vowels `a` $\longleftrightarrow$ `e`
  - Substitution of 'close letters' `m` $\longleftrightarrow$ `n`
  - Transposition of 2chars to 2chars `cc` $\longleftrightarrow$ `ch`
  - Transposition of 2chars to char `dd` $\longleftrightarrow$ `d`

Unit Cost Implementation Results

Algorithm: WF and Myers for unit-cost Levensthein distance
Scheme: TFHE-rs v0.5.3 [2]
Params: Integer: $4\times$ `PARAM_MESSAGE_2_CARRY_2_KS_PBS`
HL WF: `FheUint8` and Myers: `FheUint256`
System: Dual AMD EPYC™ 73F3 16-Core @ 3.5GHz

Algorithm	TFHE-rs API	Time per $D_{i,j}$ [ms]	Tot time [min]
WF	Integer - `smart`	1067.6	5h 10m
	Integer - `smart_par`	421.0	2h 3m
	High Level	495.5	2h 24m
Myers	High Level	-	10m 55s

Non-unit Implementation Results

Algorithm: Our extended Wagner-Fisher
Scheme: TFHE-rs v0.5.3 [2]
Params: Integer: $4\times$ `PARAM_MESSAGE_2_CARRY_2_KS_PBS`
High Level: `FheUint8`
System: Dual AMD EPYC™ 73F3 16-Core @ 3.5GHz

Algorithm	TFHE-rs API	Time per $D_{i,j}$ [ms]	Tot time [min]
WF	Integer - `smart`	1226.2	5h 56m
	Integer - `smart_par`	480.6	2h 20m
	High Level	608.0	2h 56m

References & Funding

[1] G. Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming," *J. ACM*, vol. 46, no. 3, pp. 395–415, 1999.

[2] Zama, *TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data*, `https://github.com/zama-ai/tfhe-rs`, 2022.

ERC Adv. Grant No. 101020005; CSF No. VR20192203; FWO Post-doc No. 133185

`wouter.legiest@esat.kuleuven.be`