Privacy-preserving edit distance computation using secret-sharing two-party computation

Hernan Vanegas[1] Daniel Cabarcas[2] Diego F. Aranha[3]

[1]HashCloak Inc. - Universidad Nacional de Colombia, Medellín, Colombia.

[2]Universidad Nacional de Colombia, Medellín, Colombia.

[3]Aarhus University, Aarhus, Denmark.

LATINCRYPT 2023

Overview

- Edit distance and its privacy concerns
- Wagner-Fischer algorithm (WF)
- Our solution:
  - Division of WF into two sections: preamble and arithmetic section.
  - Preamble computation
  - Arithmetic section
  - Automatic generation of formulas for the arithmetic section
- Experiments

Edit Distance

String A	A	C	G	A	A	T	T	A

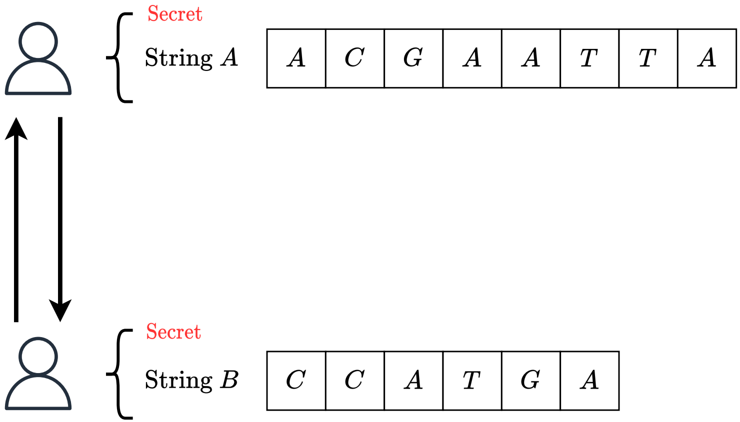
String B	C	C	A	T	G	A

- Minimum number of **insertions**, **deletions**, and **changes**.
- Wagner-Fischer algorithm [WF74].

Privacy Concerns in Edit Distance

- The edit distance is used in genomics to compare two DNA chains.
- Revealing DNA chains have risks for the chain owner:
  - Re-identification.
  - Attribute disclosure attacks via DNA.
  - Ancestry identification.

Privacy-Preserving Setup



Warning!

One of the parties can behave maliciously to steal the string of the other party.

Our Contributions

Current solutions:

- Garbled circuits (GC)
- Homomorphic encryption (HE)

But what about protocols based on secret-sharing schemes (SS)?

Our Contributions

Current solutions:

- Garbled circuits (GC)
- Homomorphic encryption (HE)

But what about protocols based on secret-sharing schemes (SS)?

Our approach:

- Two-party protocol.
- Separation of the Wagner-Fischer algorithm into two sections:
  - Preamble: Tinier [Fre+15] – binary domain.
  - Arithmetic part: SPD$\mathbb{Z}_{2^k}$ [Cra+18] and edaBits [Esc+20] – arithmetic domain.
- Mixed-circuit solution: daBits [RW19; Aly+19]
- The arithmetic part is computed in blocks. Generalization of Cheon et al. [CKL15].

The Wagner-Fischer Algorithm [WF74]

Input: two DNA chains $P = [p_1, \ldots, p_n]$ and $Q = [q_1, \ldots, q_m]$.

Let t be a matrix with dimensions $n \times m$.
for $i = 1$ to n **do**
  for $j = 1$ to m **do**
    if $p_i \neq q_j$ **then** $t(i, j) = 1$
    else $t(i, j) = 0$
Let D be a matrix with dimensions $(n + 1) \times (m + 1)$ initialized with zeros.
for $i = 0$ to n **do** $D(i, 0) = i$
for $j = 0$ to m **do** $D(0, j) = j$
for $i = 1$ to n **do**
  for $j = 1$ to m **do**

$$D(i,j) = \min \begin{cases} D(i-1,j)+1, \\ D(i,j-1)+1, \\ D(i-1,j-1)+t(i,j) \end{cases}$$

return $D(n, m)$

Preamble computation

Goal: to compute $[\![t(i,j)]\!]_2$.

- Nucleotide representation:

$$A \mapsto 00, \quad C \mapsto 01, \quad G \mapsto 10, \quad \text{and} \quad T \mapsto 11$$

Preamble computation

Goal: to compute $[\![t(i,j)]\!]_2$.

- Nucleotide representation:

$$A \mapsto 00, \quad C \mapsto 01, \quad G \mapsto 10, \quad \text{and} \quad T \mapsto 11$$

- Nucleotide secret-sharing: $[\![N]\!]_2 \stackrel{\text{def}}{=} \langle [\![b_0]\!]_2, [\![b_1]\!]_2 \rangle$.
- The XOR can be extended naturally: $[\![N]\!] \oplus [\![N']\!] \stackrel{\text{def}}{=} \langle [\![b_0]\!]_2 \oplus [\![b'_0]\!]_2, [\![b_1]\!]_2 \oplus [\![b'_1]\!]_2 \rangle$.

Preamble computation

Goal: to compute $[\![t(i,j)]\!]_2$.

- Nucleotide representation:

$$A \mapsto 00, \quad C \mapsto 01, \quad G \mapsto 10, \quad \text{and} \quad T \mapsto 11$$

- Nucleotide secret-sharing: $[\![N]\!]_2 \stackrel{\text{def}}{=} \langle [\![b_0]\!]_2, [\![b_1]\!]_2 \rangle$.
- The XOR can be extended naturally: $[\![N]\!] \oplus [\![N']\!] \stackrel{\text{def}}{=} \langle [\![b_0]\!]_2 \oplus [\![b'_0]\!]_2, [\![b_1]\!]_2 \oplus [\![b'_1]\!]_2 \rangle$.
- Equality test:

$$\left[\!\!\left[N \stackrel{?}{=} N' \right]\!\!\right]_2 = 1 - \left[([\![b_0]\!]_2 + [\![b'_0]\!]_2) + ([\![b_1]\!]_2 + [\![b'_1]\!]_2) + ([\![b_0]\!]_2 + [\![b'_0]\!]_2) \cdot ([\![b_1]\!]_2 + [\![b'_1]\!]_2) \right].$$

- Communication cost: $4nm$ bits (without considering active security mechanisms).

Arithmetic section I

We use **daBits** [RW19; Aly+19] to transform $[\![t(i,j)]\!]_2$ into $[\![t(i,j)]\!]_{2^k}$.

Goal: to compute $[\![D(n,m)]\!]_{2^k}$

We have that

$$D(i,j) = \min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+t(i,j) \end{cases}$$

Arithmetic section II

Applying the equation recursively, we end up with

$$D(i,j) = \min \begin{cases} D(i-2,j)+2 \\ D(i-2,j-1)+t(i-1,j)+1 \\ D(i-2,j-1)+3 \\ D(i-1,j-2)+3 \\ D(i-2,j-2)+t(i-1,j-1)+2 \\ D(i,j-2)+2 \\ D(i-1,j-2)+t(i,j-1)+1 \\ D(i-2,j-1)+t(i,j)+1 \\ D(i-1,j-2)+t(i,j)+1 \\ D(i-2,j-2)+t(i,j)+t(i-1,j-1) \end{cases}$$

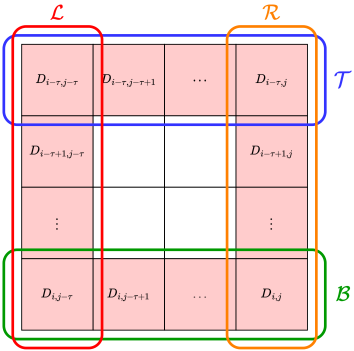
Arithmetic section III

Reducing the redundant terms, we obtain that

$$D(i,j) = \min \begin{cases} D(i-2,j)+2 \\ D(i-2,j-1)+t(i-1,j)+1 \\ D(i-2,j-1)+t(i,j)+1 \\ D(i-2,j-2)+t(i-1,j-1)+t(i,j) \\ D(i,j-2)+2 \\ D(i-1,j-2)+t(i,j-1)+1 \\ D(i-1,j-2)+t(i,j)+1 \end{cases}$$

Later, we will present an algorithm to do this!

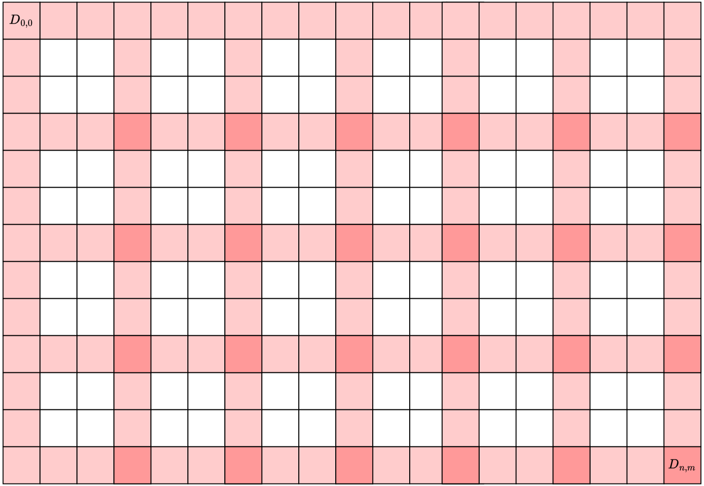
Arithmetic section IV



$$\mathcal{T} \stackrel{\text{def}}{=} \{D_{i-\tau,j-\tau}, D_{i-\tau,j-\tau+1}, \dots, D_{i-\tau,j}\}\,,\ \mathcal{B} \stackrel{\text{def}}{=} \{D_{i,j-\tau}, D_{i,j-\tau+1}, \dots, D_{i,j}\}\,,$$

$$\mathcal{L} \stackrel{\text{def}}{=} \{D_{i-\tau,j-\tau}, D_{i-\tau+1,j-\tau}, \dots, D_{i,j-\tau}\}\,,\ \mathcal{R} \stackrel{\text{def}}{=} \{D_{i-\tau,j}, D_{i-\tau+1,j}, \dots, D_{i,j}\}\,.$$

Arithmetic section V



Arithmetic section VI

To compute $[\![D(i,j)]\!]_{2^k}$, we can use the protocol $\textsc{Min}_q$ [Dam+19; Tof07]. The protocol computes securely the minimum of a list of q elements using:

- $O(\log_2 q) \cdot (c_r + 2)$ online rounds (c_r is the number of rounds for one comparison),
- $q - 1$ comparisons, and
- $2q - 2$ multiplications.

Warning!

Comparisons and multiplications need pre-processing material.

Arithmetic section VII

Using the protocol $\textsc{Min}_7$,

$$[\![D(i,j)]\!]_{2^k} = \textsc{Min}_7 \begin{cases} [\![D(i-2,j)]\!]_{2^k} + 2 \\ [\![D(i-2,j-1)]\!]_{2^k} + [\![t(i-1,j)]\!]_{2^k} + 1 \\ [\![D(i-2,j-1)]\!]_{2^k} + [\![t(i,j)]\!]_{2^k} + 1 \\ [\![D(i-2,j-2)]\!]_{2^k} + [\![t(i-1,j-1)]\!]_{2^k} + [\![t(i,j)]\!]_{2^k} \\ [\![D(i,j-2)]\!]_{2^k} + 2 \\ [\![D(i-1,j-2)]\!]_{2^k} + [\![t(i,j-1)]\!]_{2^k} + 1 \\ [\![D(i-1,j-2)]\!] + [\![t(i,j)]\!]_{2^k} + 1 \end{cases}$$

The same holds for computing $[\![D(i-1,j)]\!]_{2^k}$ and $[\![D(i,j-1)]\!]_{2^k}$ using the $\textsc{Min}_4$ protocol.

Arithmetic section VIII

Fact

The number of formulas needed to compute $D(i,j)$ is $O(\tau \cdot 2^{3\tau})$ **(we will see this later)**.

Fact

All the positions in $\mathcal{R} \cup \mathcal{B}$ inside a $(\tau+1)$-box can be computed in parallel.

The arithmetic part can be computed in:

- $O\left(\frac{nm}{\tau^2} \cdot (3\tau + \log_2 \tau) \cdot (c_r + 2)\right)$ rounds,
- $O\left(\frac{nm}{\tau^2} \cdot (\tau^2 \cdot 2^{3\tau} - 1)\right)$ comparisons, and
- $O\left(\frac{nm}{\tau^2} \cdot (\tau^2 \cdot 2^{3\tau+1} - 2)\right)$ multiplications.

Removing Redundant Formulas I

$$D(i,j) = \min \begin{cases} D(i-2,j)+2 \\ D(i-2,j-1)+t(i-1,j)+1 \\ D(i-2,j-1)+3 \\ D(i-1,j-2)+3 \\ D(i-2,j-2)+t(i-1,j-1)+2 \\ D(i,j-2)+2 \\ D(i-1,j-2)+t(i,j-1)+1 \\ D(i-2,j-1)+t(i,j)+1 \\ D(i-1,j-2)+t(i,j)+1 \\ D(i-2,j-2)+t(i,j)+t(i-1,j-1) \end{cases}$$

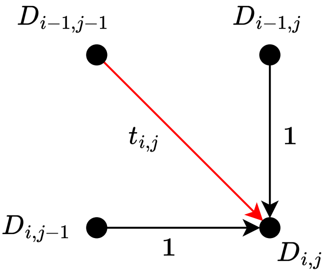
Removing Redundant Formulas II

$$D(i,j) = \min \begin{cases} D(i-2,j)+2 \\ D(i-2,j-1)+t(i-1,j)+1 \\ D(i-2,j-1)+3 \\ D(i-1,j-2)+3 \\ D(i-2,j-2)+t(i-1,j-1)+2 \\ D(i,j-2)+2 \\ D(i-1,j-2)+t(i,j-1)+1 \\ D(i-2,j-1)+t(i,j)+1 \\ D(i-1,j-2)+t(i,j)+1 \\ D(i-2,j-2)+t(i,j)+t(i-1,j-1) \end{cases}$$

$$D(i-2,j-1)+t(i-1,j)+1 \leq D(i-2,j-1)+3$$

$D(i-2,j-1)+3$ **can be deleted safely!**

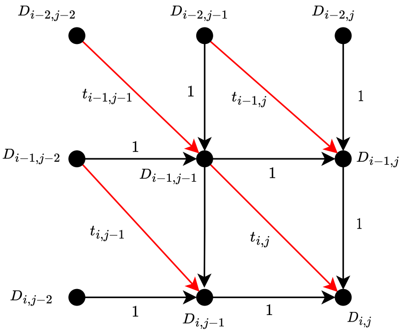
The Depedency Graph I



$$D(i,j) = \min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+t(i,j) \end{cases}$$

A similar abstraction was considered by [Ukk85] without edge coloring.

The Depedency Graph II



- A path in the graph induces a formula.
- Let P and Q be two paths.
  - $r_{P,Q}$: number of red edges in P that are not in Q.
  - b_P: number of black edges in the path P.

Algorithm for formula generation

- We defined a notion of **"optimality"** of a set of formulas.
- Correctness: $r_{Q,P} + b_Q \leq b_P$.

Algorithm for formula generation

- We defined a notion of **"optimality"** of a set of formulas.
- Correctness: $r_{Q,P} + b_Q \leq b_P$.

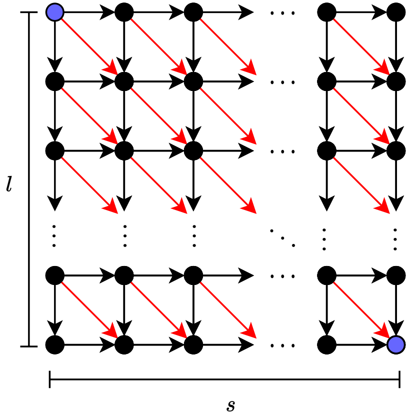
Input: a dependency graph G. Two endpoints $U \in \mathcal{T} \cup \mathcal{L}$ and $W \in \mathcal{B} \cup \mathcal{R}$.

Output: an "optimal" set of paths $\mathcal{S}$ to compute the edit distance correctly.

Generate the set $\mathcal{P}_{U,W}$.
$\mathcal{S} \leftarrow \emptyset$.
for $P \in \mathcal{P}_{U,W}$ **do**
  $r \leftarrow$ True
  for $Q \in \mathcal{P}_{U,W} \setminus \{P\}$ **do**
    if $r_{Q,P} + b_Q \leq b_P$ **then**
      $r \leftarrow$ False
      break
  if $r =$ True **then**
    Append P to $\mathcal{S}$
return $\mathcal{S}$

Upper Bound for the Number of Formulas

Delanoy graph



$$\mathcal{D}(l,s) = \sum_{i=0}^{\min\{l,s\}} \binom{l}{i}\binom{s}{i} \cdot 2^i.$$

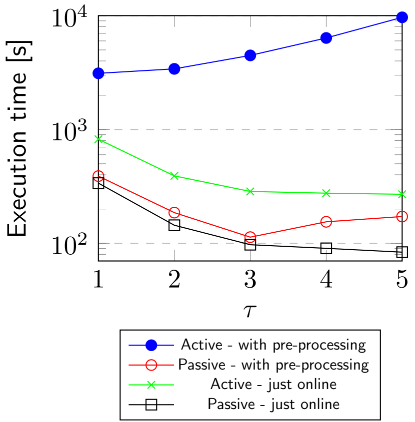
An upper bound for the number of formulas is

$$\sum_{k=1}^{\tau} [\mathcal{D}(\tau,\tau-k) + \mathcal{D}(\tau-k,\tau)] + \mathcal{D}(\tau,\tau) = O(\tau \cdot 2^{3\tau})$$

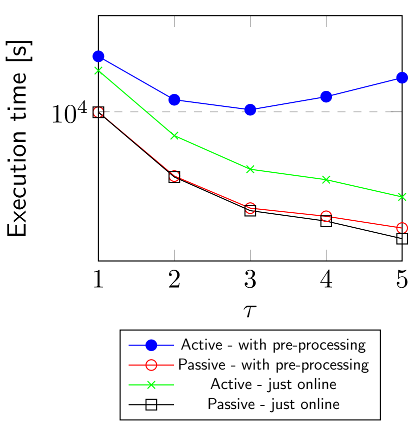
Experiments

- MP-SDPZ framework [Kel20].
- AWS EC2 instance of type `c6a.4xlarge`.
- We simulated a LAN architecture:
  - Bandwidth: 1.6 GBps.
  - Latency: 0.3 milliseconds.
- We consider a bit-length of 16, which allows 16-bit integer computations.

Effect of the Box Size τ



(a) No network limits.



(b) LAN.

GC-Based vs. SSS-Based Protocols

Network	Security	Protocol	Time [s]	Data sent [MB]
No limit	Passive	Yao's GC	2.6	345.8
		Semi2^k	4.9	113.1
	Active	BMR-MASCOT	5,968.6	2.09×10^6
		SPD$\mathbb{Z}_{2^k}$	140.1	14,893.8
LAN	Passive	Yao's GC	2.7	345.8
		Semi2^k	103.0	113.1
	Active	BMR-MASCOT	$9,034.0$	2.09×10^6
		SPD$\mathbb{Z}_{2^k}$	368.5	14,893.8

Here, we used $\tau = 1$ for GC-based protocols and $\tau = 3$ for SSS-based protocols.

Additional Results

Comparison with field-based protocol

On a 1020 long DNA-chain Semi2^k sends 85% less data than Semi and SPD$\mathbb{Z}_{2^k}$ sends 86% less data than MASCOT [KOS16].

Additional Results

Comparison with field-based protocol

On a 1020 long DNA-chain Semi2^k sends 85% less data than Semi and SPD$\mathbb{Z}_{2^k}$ sends 86% less data than MASCOT [KOS16].

Comparison with homomorphic encryption solutions

Cheon et al. [CKL15]: DNA chains of length 8 at 80 bits of security.

- Key generation: 27.54 seconds.
- Encryption: 16.45 seconds.
- Computation: 27.5 seconds

Additional Results

Comparison with field-based protocol

On a 1020 long DNA-chain Semi2^k sends 85% less data than Semi and SPD$\mathbb{Z}_{2^k}$ sends 86% less data than MASCOT [KOS16].

Comparison with homomorphic encryption solutions

Cheon et al. [CKL15]: DNA chains of length 8 at 80 bits of security.

- Key generation: 27.54 seconds.
- Encryption: 16.45 seconds.
- Computation: 27.5 seconds

In our case, considering both the pre-processing and the online phase on a LAN, using $\tau = 2$:

- Semi2^k: 0.3 seconds.
- SPD$\mathbb{Z}_{2^k}$: 5.92 seconds

Additional Results

Comparison with field-based protocol

On a 1020 long DNA-chain Semi2^k sends 85% less data than Semi and SPD$\mathbb{Z}_{2^k}$ sends 86% less data than MASCOT [KOS16].

Comparison with homomorphic encryption solutions

Cheon et al. [CKL15]: DNA chains of length 8 at 80 bits of security.

- Key generation: 27.54 seconds.
- Encryption: 16.45 seconds.
- Computation: 27.5 seconds

In our case, considering both the pre-processing and the online phase on a LAN, using $\tau = 2$:

- Semi2^k: 0.3 seconds.
- SPD$\mathbb{Z}_{2^k}$: 5.92 seconds

For chains of length 100:

- Cheon et al. [CKL15]: 1 day 5 hours (62-bit security).
- **Our work:** 96.69 seconds on a LAN using the SPD$\mathbb{Z}_{2^k}$.

Thanks

Gracias

Obrigado

Bibliography I

[Aly+19] Abdelrahaman Aly et al. *Zaphod: Efficiently Combining LSSS and Garbled Circuits in SCALE*. Cryptology ePrint Archive, Paper 2019/974. 2019.

[CKL15] Jung Hee Cheon, Miran Kim, and Kristin E. Lauter. "Homomorphic Computation of Edit Distance". In: *Financial Cryptography Workshops*. Vol. 8976. LNCS. Springer, 2015, pp. 194–212.

[Cra+18] Ronald Cramer et al. "SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority". In: *CRYPTO (2)*. Vol. 10992. LNCS. Springer, 2018, pp. 769–798.

[Dam+19] Ivan Damgård et al. "New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning". In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2019, pp. 1102–1120.

[Esc+20] Daniel Escudero et al. "Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits". In: *CRYPTO (2)*. Vol. 12171. LNCS. Springer, 2020, pp. 823–852.

[Fre+15] Tore Kasper Frederiksen et al. "A Unified Approach to MPC with Preprocessing Using OT". In: *ASIACRYPT (1)*. Vol. 9452. LNCS. Springer, 2015, pp. 711–735.

Bibliography II

[Kel20] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *CCS*. ACM, 2020, pp. 1575–1590.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer". In: *CCS*. ACM, 2016, pp. 830–842.

[RW19] Dragos Rotaru and Tim Wood. "MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security". In: *INDOCRYPT*. Vol. 11898. LNCS. Springer, 2019, pp. 227–249.

[Tof07] Tomas Toft. "Primitives and Applications for Multi-party Computation". PhD thesis. Aarhus University, 2007.

[Ukk85] Esko Ukkonen. "Algorithms for Approximate String Matching". In: *Inf. Control.* 64.1-3 (1985), pp. 100–118.

[WF74] Robert A. Wagner and Michael J. Fischer. "The String-to-String Correction Problem". In: *J. ACM* 21.1 (1974), pp. 168–173.