Approximate and Private String Set Intersection

Paolo D'Arco[1], Vito Troisi[1,*]

[1]Università degli Studi di Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano SA

Abstract

The Private Set Intersection (PSI) problem certainly represents one of the most interesting subjects of study in Secure Multi-Party Computation (SMPC). Beyond being extremely relevant from a theoretical standpoint, solutions to PSI have different practical applications, from genetic to marketing, from healthcare to homeland security, just to name a few. One particular variant deals with computing the intersection of sets of strings. However, in numerous settings, computing the *exact* intersection of the sets is overabundant and even unnecessary. Let us consider, for example, genetics: it is sometimes enough to identify pairs of genomic sequences which are *sufficiently similar*, where the similarity between the sequences is measured by metrics such as the *edit distance*. In this paper, we describe a two-party protocol that solves this approximate PSI variant. Specifically, we present a meta-protocol that can be instantiated with any secure subprotocol for computing the edit distance. Our concrete choices are then two strategies that represent the state-of-the-art: the first one is based on Yao's garbled circuits and breaks the computation of the edit distance into its main operations of sum, comparison, minimum and table-lookup. The second, based on secret sharing schemes, minimizes the number of formulas to be evaluated to compute the edit distance. The two strategies are compared in order to evaluate, when used in our protocol, time and communication complexity. Finally, we will conclude the paper summarizing our findings and discussing some open challenges.

Keywords

SMPC, PSI, LDA-PSI, edit distance

1. Introduction

The main objective of **Secure Multi-Party Computation** is to enable a set of independent parts - which trust neither a third common party nor each other - to compute a known function depending of their private inputs, while keeping the inputs private. As quoted in the excellent introduction to the subject by Evans et al.[1], SMPC has gradually evolved during the last years, on both the theoretical and pragmatical fronts.

Amongst the various problem in SMPC, the **Private Set Intersection (PSI)** problem is certainly one of the cornerstones. Let us consider two parties, Alice and Bob (we will assume that an authenticated, secure channel has already been established between the two). Alice and Bob respectively hold the sets A and B. We can assume their elements belong to a given universal set $\mathbb{U}$, and thus $A, B \subseteq \mathbb{U}$. The parties' objective is to compute $A \cap B$, while ensuring that:

- At the end of the protocol, Alice will not know any other element of B out of those in $A \cap B$;
- Likewise, Bob will not know any other element of A out of those in $A \cap B$;

A concise summary of different solutions to the PSI problem can be found in [2]. However, in some settings, it may be satisfactory to the parties that an element of A is merely *sufficiently similar* to an element of B, in order to put it in the computed intersection. That is, we are interested in finding close pairs of elements $(a, b) \in A \times B$ such that $dist(a, b)$, which is a properly-selected distance metric, is a "small quantity". This variant is known as **Distance-Aware PSI** [3].

Our Contribution. In this paper, A and B are sets of strings and, speaking of metrics, we will deal with the *edit distance*. As in [4], we shall refer to our problem as to the **Levenshtein Distance-Aware**

Joint National Conference on Cybersecurity (ITASEC & SERICS 2026), February 09-13, 2026, Cagliari, IT

*Corresponding author.

pdarco@unisa.it (P. D'Arco); vito.troisi19@gmail.com (V. Troisi)

Private Set Intersection (LDA-PSI) problem. We follow the same treatment detailed in [4]. We will present a meta-protocol for solving the LDA-PSI problem that can be instantiated with any secure subprotocol for computing the edit distance. Then, we will focus on two efficient methods to compute the edit distance in the SMPC setting. The first, based on circuit garbling, by Zhu et al. [5], and the second, based on SSS by Vanegas et al. [6]. Finally, we discuss the feasibility of our LDA-PSI protocol, employing the two strategies.

2. Preliminaries

Let us introduce the basic theory surrounding the computation of the edit distance.

Strings. We shall refer to a set Σ of "characters" (or "symbols") as an *alphabet*. A *string* w over Σ is a sequence:

$$w = \sigma_1\sigma_2...\sigma_n, \text{ such that } \sigma_i \in \Sigma,\ 1 \leq i \leq n$$

The integer n is the *length* of w, denoted with $|w|$. Furthermore, let ϵ denote the *empty string*, devoid of characters. Finally, let Σ^* represent the set of all strings that can be built over Σ.

Edit Distance. The edit distance (or Levenshtein distance) between two strings a and b intuitively represents the minimum number of operations (insertions, substitutions or deletions of a symbol) required to transform a into b. For instance, in order to turn "park" into "dark", one operation is sufficient: we can substitute the "p" for a "d". Thus, the edit distance is $D(\text{park}, \text{dark}) = 1$.

Wagner-Fischer Algorithm. Computing the edit distance between two given strings is a well-known algorithmic problem. Let $a = a_1...a_n$ and $b = b_1...b_m$ be two strings over an alphabet Σ. In general, $n \neq m$. We define the following operations on a string $\sigma_1...\sigma_n \in \Sigma^*$ and an arbitrary $i \in [n]$, where $[n] \stackrel{\text{def.}}{=} \{1, ..., n\}$:

- **Insertion** of $x \in \Sigma$ in position i: the output is the string $\sigma_1...\sigma_{i-1}x\sigma_i...\sigma_n$;
- **Substitution** of symbol σ_i for $x \in \Sigma$: the resulting string is $\sigma_1...\sigma_{i-1}x\sigma_{i+1}...\sigma_n$;
- **Deletion** of σ_i: the resulting string is $\sigma_1...\sigma_{i-1}\sigma_{i+1}...\sigma_n$.

The most celebrated algorithm for the computation of the edit distance is the dynamic programming algorithm proposed by Wagner and Fischer[7]. Using the notation of [6], let $a^{(i)}$ denote the substring $a_1...a_i$, $i \in [n]$. Let us also define $D(i, j) \stackrel{\text{def.}}{=} D(a^{(i)}, b^{(j)})$. Thus, our goal is to find $D(n, m) = D(a, b)$. The algorithm implements the following recurrence relation:

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

The idea is to transform the prefix $A[1...i]$ into the prefix $B[1...j]$ in the cheapest way. The details of the **WF** algorithm are given below.

Levenshtein Distance-Aware Private Set Intersection. Let Alice and Bob be the two parties of our protocol, whose respective sets of strings are $A, B \in \Sigma^*$. Let $h \in \mathbb{N}$ be a *threshold* value on which Alice and Bob mutually agree. Lastly, let $D : \Sigma^* \times \Sigma^* \to \mathbb{N}$ be the edit distance function. A **LDA-PSI protocol** is any protocol returning $S \subseteq A \times B$ such that the following properties hold:

- **Correctness**: $\forall (a, b) \in A \times B : \quad (a, b) \in S \iff D(a, b) \leq h$
  That is, the pair is added to the "quasi-intersection" if and only if the edit distance between the strings does not exceed the threshold.
- **Privacy**: Alice gains no information on the strings $b \in B$ s.t. $\nexists (x, b) \in S : x \in A$. Likewise, Bob gains no information on any $a \in A$ s.t. $\nexists (a, x) \in S : x \in B$.

Let t be a $n \times m$ matrix (indices starting from 1)
for $(i, j) \in [n] \times [m]$ **do**
  if $a_i \neq b_j$ **then**
    $t(i, j) \leftarrow 1$;
  end
  $t(i, j) \leftarrow 0$;
end
Let D be a 0-initialized $(n + 1) \times (m + 1)$ matrix (indices starting from 0)
for $i = 0$ *to* n **do**
  $D(i, 0) \leftarrow i$
end
for $j = 0$ *to* m **do**
  $D(0, j) \leftarrow j$
end
for $i = 1$ *to* n **do**
  for $j = 1$ *to* m **do**

$$D(i, j) \leftarrow \min \begin{cases} D(i-1, j) + 1, & \text{(deletion)} \\ D(i, j-1) + 1, & \text{(insertion)} \\ D(i-1, j-1) + t(i, j) & \text{(substitution)} \end{cases}$$

  end
end
return $D(n, m)$

Algorithm 1: Wagner-Fischer Algorithm (**WF**) for the edit distance

3. Cryptographic primitives

The protocols for computing the edit distance we consider, use several cryptographic primitives. Let us briefly overview the relevant ones.

Oblivious Transfer. The *oblivious transfer* (OT) represents one of the most important cryptographic primitive in SMPC. Intuitively, a 1-out-of-n OT protocol allows two parties, the *sender* – who holds n secrets $s_1, ..., s_n$ – and the *receiver* – who is interested in one of them, s_j, $j \in [n]$ – to share s_j while ensuring that:

- The receiver gains *no information* on other secrets $s_i, i \neq j$.
- The sender does not know which is the secret of interest, i.e. the sender gains no information on j.

In the specific case in which $n = 2$, i.e. the receiver selects exactly one of two possible secrets to receive, we refer to the protocol as to 1-out-of-2 oblivious transfer.
It is well-know how such a primitive is essentially equivalent to the concept of Secure Multi-Party Computation [1]. More formally, it is indeed possible to realize SMPC without any other assumption but the existence of oblivious transfer. Conversely, oblivious transfer can be realized by assuming the existence of a protocol for SMPC. In the very same source, an efficient 1-out-of-2 OT protocol by Ishai et al. [8] (generalized to 1-out-of-n by Kolesnikov et al. [9]) is also described.

Secure garbling. First proposed by A. Yao [10] following the birth of SMPC [11], the idea of *garbling* consists in a protocol that allows parties to evaluate a function, described as a boolean circuit, without disclosing their inputs or employing trusted third parties. The main cryptographic primitives we will need for garbling are *oblivious transfer* and a symmetric encryption scheme using two keys. Let η be the security parameter of our two-party garbling protocol. Alice will act as the *garbler*, whose job will be to “encrypt" the circuit gates.

- Let b be a circuit wire. Alice sample two cryptographic keys $k_0^b, k_1^b \in \{0, 1\}^\eta$ uniformly at random, representing boolean values 0 e 1, and assigns them to wire b;

- Alice examines each circuit gate, replacing truth values 0 and 1 with keys $k_0^{\cdot}$ and $k_1^{\cdot}$. Then Alice encrypts the key representing the output with the pair of keys representing the input- For instance, the truth table of a XOR gate whose input wires are a and b and whose output value is c is transformed in:

In	Out
$0,0$	0
$0,1$	1
$1,0$	1
$1,1$	0

$\rightarrow$

In	Out
k_0^a, k_0^b	k_0^c
k_0^a, k_1^b	k_1^c
k_1^a, k_0^b	k_1^c
k_1^a, k_1^b	k_0^c

$\rightarrow$

Ciphertexts
$Enc_{k_0^a,k_0^b}(k_0^c)$
$Enc_{k_0^a,k_1^b}(k_1^c)$
$Enc_{k_1^a,k_0^b}(k_1^c)$
$Enc_{k_1^a,k_1^b}(k_0^c)$

- Alice then permutes the encrypted rows, preventing the output key to be deduced from the order of the rows.

Now, Alice has to send Bob the garbled circuit (i.e. the encrypted–and–permuted truth tables), as well as her input. Alice's input must of course remain private. Let $a = a_1, .., a_k$ be the binary string representing Alice's input. Alice has substituted 0 for $k_0^{\cdot}$ and 1 for $k_1^{\cdot}$ in her input and sends the string:

$$k_{a_1}^{a_1}, ..., k_{a_k}^{a_k}$$

to Bob. Bob is unaware of the true value of Alice's input since the keys were sampled uniformly at random.

In order to evaluate the circuit, Bob must also obtain the representation of his input as a sequence of keys. However, Alice must not know Bob's private input. Let b_i be the i-th bit of Bob's input string. Let $k_0^{b_i}, k_1^{b_i}$ the uniformly-sampled keys that Alice has prepared to describe such input. Alice e Bob execute the 1-out-of-2 oblivious transfer protocol, $\forall i$, with Alice acting as the *sender* and Bob as the *receiver*. Thus, Bob executes the protocol by requesting the key corresponding to bit b_i, i.e. $k_{b_i}^{b_i}$. For instance, if $b_i = 1$, Bob will receive $k_1^{b_i}$ and will gain no information on $k_0^{b_i}$. On the other hand, Alice will not know anything about b_i. Notice how the properties of oblivious transfer have been leveraged to maintain privacy.

Bob can now evaluate the circuit. Notice how, for each gate to be evaluated, Bob can decipher one row of the corresponding truth table, and one only. Namely, Bob can decrypt the row correspoding to the ciphertext obtained with the keys that match the gate's actual input. For instance, let us suppose that Bob evaluates the XOR gate between his bit $b_1 = 0$ and Alice's bit $a_1 = 1$:

Ciphertexts (that Bob sees permuted)
$Enc_{k_0^{a_1},k_0^{b_1}}(k_0^c)$
$Enc_{k_0^{a_1},k_1^{b_1}}(k_1^c)$
$Enc_{k_1^{a_1},k_0^{b_1}}(k_1^c)$
$Enc_{k_1^{a_1},k_1^{b_1}}(k_0^c)$

Bob can only decrypt the third row, since $\text{Enc}_{k_1^{a_1},k_0^{b_1}}(k_1^c)$ is the ciphertext obtained with keys $k_1^{a_1}$ and $k_0^{b_1}$, matching the above inputs. Bob does not gain any information on the other rows. Bob continues to evaluate the circuit until k_b^p, matching the output wire p, where $b \in \{0, 1\}$, is the circuit's output. Of course, Bob does not know which binary value k_b^p resolves into, but Alice does, having generated the keys. Therefore, Bob sends k_b^p to Alice, who can finally disclose output b.

Garbling-based approaches to SMPC tend to have a low round complexity (since the # of rounds does not depend on the circuit), but the amount of data sent between parties is considerable: the entire garbled representation of the circuit has to be sent over to the other party.

Secret Sharing Schemes. A secret sharing scheme (SSS)[12] is a cryptographic primitive that enables to split a secret s in *shares*, each of which is assigned to one of n parties. In a threshold secret sharing scheme, the goal is to allow for the reconstruction of the secret if and only if $k \geq n$ parties combine their shares. On the other hand, the knowledge of $< k$ shares yields no information on s.
We shall use the same notation as the one employed by Vanegas et al.[6]. Thus, given a string s, we will denote a *sharing* of s as:

$$[\![s]\!] = (s_1, ..., s_n)$$

where s_i represents the share assigned to P_i.
Numerous SMPC protocols, such as SPD$\mathbb{Z}_{2^k}$[13] are based on secret sharing. Jumping ahead, the underlying intuition is to allow the parties to distribute shares of their private inputs and evaluate the arithmetic circuit that describes the function, thus obtaining a share of the output. The complete output is then reconstructed by combining the shares. Another important protocol, Tinier [14] works similarly, but uses binary circuits.

daBits. Vanegas et al.[6] realize their edit distance protocol with Tinier and SPD$\mathbb{Z}_{2^k}$, which run on different domains (the binary field $\mathbb{F}_2$ and the arithmetic field $\mathbb{F}_{2^k}$, respectively). Some operations are more efficient on a binary domain, such as bitwise XOR. However, linear operations like addition and multiplication are more efficient on an arithmetic field. Thus, need arising, mixed approaches to SMPC are born. These strategies switch between representations of elements in different domains depending on the operation, enabling us to exploit the advantages of both; daBits (*double-shared authenticated bits*), formalized by Rotaru and Wood [15], are a cryptographic primitive that aims to convert an element from a binary domain to an arithmetic one.
Precisely, daBits transform elements of $\mathbb{F}_2$ in elements of $\mathbb{F}_{2^k}$ *while preserving authenticity*. In other words, an element $a \in \mathbb{F}_2$ bound to, for instance, a MAC, will retain the same authenticity guarantees after its conversion to an element of $\mathbb{F}_{2^k}$. Formally, a daBit is a pair $([\![b]\!]_2, [\![b]\!]_{2^k})$ where $b \in \{0, 1\}$ is a bit. That is, a daBit is a sharing of a bit in both a binary and an arithmetic domain. We can therefore use daBits for domain conversion, e.g., the conversion protocols described by Damgård et al.[16],

edaBits. An edaBit (*extended daBit*) is a generalization of the concept of daBit. Formally it consists in a tuple $((r_{m-1}, ..., r_0), r)$ such that:

- $(r_{m-1}, ..., r_0)$ is an m-uple of random bits, secret shared in $\mathbb{F}_2$;
- $r = \sum_{i=0}^{m-1} r_i 2^i$ is an integer (being *de facto* the integer representation of the string $r_{m-1}, ..., r_0$) secret shared in the arithmetic domain $\mathbb{F}_{2^k}$.

Notice how a edaBit of length $m = 1$ is a daBit. Escudero et al. [17] present a protocol to securely generate edaBits of length m, which is comparatively more efficient than generating m daBits. Furthermore, [17] illustrates a protocol that makes use of edaBits for efficient integer comparison, which is exactly what we will, referring to Vanegas et al.[6], need it for.

4. The Real-Ideal Paradigm

The Real-Ideal Paradigm can be used to define and prove security in the context of SMPC. At the very core of what will follow, stands the definition of *functionality*. Goldreich [18] defines an m-ary functionality as a random process that maps m inputs to m outputs. Notice how this generalizes the concept of an m-to-m function which, according to this definition, represents the special case of a deterministic functionality.

Ideal World. Let $\mathcal{F}$ be a functionality. In the ideal world, $\mathcal{F}$ acts a trusted third-party, which as we are in an SMPC context, does not exist in the real world. Rather, $\mathcal{F}$ is merely an ideal entity with respect to which we are going to define security. Let P_i, $i \in [n]$ be our parties, and x_i be the input of P_i. In the ideal world:

- P_i sends x_i to $\mathcal{F}$, $1 \leq i \leq n$;
- $\mathcal{F}(x_1, ..., x_n)$ is computed and returned to each party.

An adversary can corrupt any party, but not the trusted $\mathcal{F}$. Thus, the adversary can choose x_i independently of the inputs $x_j, j \neq i$ belonging to other uncorrupted parties, and can only know $\mathcal{F}(x_1, ..., x_n)$ which is indeed the only message received by P_i – the same as every other parties'. These requirements, easily obtainable in the ideal world, correspond to what we would like to build in the real world, in spite of the absence of a trusted third-party.

Real World. The best we can hope for in the real world is to let the parties execute a protocol Π that emulates the behaviour of the ideal $\mathcal{F}$. Every party holds a *view* of the protocol, which includes the input x_i, the *random bits* that the party can use for random choices and the messages received from other parties. Let us denote this view as $V_i^{\Pi}(x_1, ..., x_n)$ where $x_1, ..., x_n$ are the parties' inputs. In addition, let $o^{\Pi} = (o_1^{\Pi}, ..., o_n^{\Pi})$ be the output of the protocol on the very same input, and o_i^{Π} is the output returned to P_i.
In order to prove that Π is secure against given adversaries, we want to show that anything the adversary can achieve in the real world can also be achieved in the ideal world. Having argued security in the ideal world, it must follow that the real protocol has to be secure as well. Essentially, we are proving that protocol Π emulates $\mathcal{F}$ well. We will formalize this concept via *indistinguishability*.

Semi-Honest Adversarial Model. A *semi-honest adversary*, also known as *honest-but-curious* adversary, can corrupt parties and gain as much information as they can from the messages they receive. Still, a semi-honest adversary follows the intended protocol faithfully, without injecting, dropping or otherwise altering messages. Formally, we consider an adversary to be a probabilistic algorithm that runs in polynomial time (PPT). Therefore, any attack brought forth by the adversary must be an efficiently computable function of the adversary's view[1]. Thus, without loss of generality, we can restrict our consideration to the single attack in which the adversary outputs the whole view of the corrupted party. In order for the protocol to be secure, it must then be possible to generate a view in the ideal world that is indistinguishable from the adversary's view in the real world. In such a case, the adversary will retain the same information in both worlds, and thus since we argued security in the ideal world, the same will apply to its real counterpart. By indistiguishable, we are of course implicitly referring to computational indistinguishability. Any PPT adversary *cannot* distinguish the real view from the ideal view with non–negligible probability, due to its computational constraints.

By definition, an adversary corrupting party P_i in the ideal world can access the view:

$$(x_i, \mathcal{F}(x_1, ..., x_n))$$

since $\mathcal{F}(x_1, ..., x_n)$ is the sole message received by P_i. The adversary that generates a real view from an ideal view shall be referred to as a *simulator*. In order to simulate a real protocol, the simulator must also generate the output distribution of the simulated protocol. For deterministic functionalities, such as the one we are going to introduce, Lindell[19] provides a simple definition of security as a special case of probabilistic functionalities. According to such a definition, a protocol Π *realizes* the deterministic ideal functionality $\mathcal{F}$ in the semi-honest model if and only if the following hold:

- **Correctness**: the parties' output is correct with non–negligible probability:

$$\forall \eta,\ \forall (x_1, ..., x_n), \Pr[o^{\Pi} \neq \mathcal{F}(x_1, ..., x_n)] \leq \text{negl}(\eta) \quad (\text{negligible function}) \tag{2}$$

- **Privacy**: there exist PPT simulators for the protocol's parties' views:

$$\forall (x_1, ..., x_n), V_i^{\Pi}(x_1, ..., x_n) \equiv Sim_i(x_i, \mathcal{F}(x_1, ..., x_n)), 1 \leq i \leq n \tag{3}$$

  where the symbol $\equiv$ denotes computational indistinguishability by any PPT distinguisher.

LDA-PSI Functionality. Our deterministic functionality of interest is (D denotes the edit distance):

Ideal Functionality $\mathcal{F}_{\text{LDA-PSI}}$
Inputs. Alice inputs a set of strings A, Bob inputs a set of strings B.
Outputs. Alice and Bob both receive S, where $S = \{(a, b) \in A \times B \mid D(a, b) \leq h\}$. Furthermore, Alice receives $\lvert B\rvert$ and Bob receives $\lvert A\rvert$.

5. The LDA-PSI Protocol

Our approach for the computation of LDA-PSI according to the edit distance is detailed below:

Protocol 1 LDA-PSI

Input. Alice and Bob hold sets of strings A, B – respectively, on alphabet Σ. Alice and Bob also share a common threshold t.

Setup. Alice and Bob set the intersection $S = \emptyset$. A correct and private sub-protocol $\mathcal{D}(w, w')$, that returns the edit distance between input strings w and w', is provided to both Alice and Bob. In addition, Alice and Bob exchange set sizes $|A|$ and $|B|$.

Execution. For each pair of strings $(a, b) \in A \times B$:

(1) Alice and Bob execute the sub-protocol $\mathcal{D}(a, b)$ and get the edit distance t';

(2) If $t' \leq t$, Bob sends b to Alice and Alice sends a to Bob; they both set $S \leftarrow S \cup \{(a, b)\}$;

Output. Alice and Bob output the set S;

The correctness of the protocol is elementary. If the sub-protocol $\mathcal{D}$ is correct, then Alice and Bob add (a, b) to set S if and only if $D(a, b) \leq t$. Thus, S will contain every pair of strings whose edit distance is at most t.
We show that the protocol is also private using a hybrid model: let us assume that Alice and Bob can access an ideal functionality $\mathcal{F}_{\text{Edit-Dist}}$ defined as follows:

Ideal Functionality $\mathcal{F}_{\text{Edit-Dist}}$
Inputs. Alice inputs a string a, Bob inputs a string b.
Outputs. Alice and Bob receive $d \stackrel{\text{def.}}{=} D(a, b)$.

In this model, we define the simulators $Sim_{\text{Alice}}(A, S, |B|)$ and $Sim_{\text{Bob}}(B, S, |A|)$ as follows:

$Sim_{\text{Alice}}(A, S, |B|)$: for each $a \in A$,

- sends a to $\mathcal{F}_{\text{Edit-Dist}}$, $|B|$ times, and receives d (for each different $b \in B$);
- if $d \leq t$, then sends a to Bob.

$Sim_{\text{Bob}}(B, S, |A|)$: for each $b \in B$,

- sends b to $\mathcal{F}_{\text{Edit-Dist}}$, $|A|$ times, and receives d (for each different $a \in A$);
- if $d \leq t$, then sends b to Alice.

Notice that $\mathcal{F}_{\text{Edit-Dist}}$ receives a string from Alice and one from Bob and returns the edit distance to both parties. The parties know, as part of the setup phase, the cardinality of both sets[1].

The transcript of the simulators is indeed identical to the transcript produced by a real execution of the protocol run between Alice and Bob in the hybrid model, in which the parties access the ideal functionality. Naturally, in a real execution, the parties cannot access $\mathcal{F}_{\text{Edit-Dist}}$ and must run a protocol to privately compute the edit distance. But, due to the composition theorem [18], the full protocol that uses the secure implementation of $\mathcal{F}_{\text{Edit-Dist}}$ is secure as well. More formally, we can obtain the complete simulators for Alice and Bob by modifying the ones provided: specifically, we substitute the ideal calls to $\mathcal{F}_{\text{Edit-Dist}}$ with calls to the simulators $Sim_{\text{Alice}}^{\text{Edit-Dist}}(a, d)$ and $Sim_{\text{Bob}}^{\text{Edit-Dist}}(b, d)$ that exist – if we suppose the existence of a secure protocol that computes the edit distance. Notice how the ideal functionality $\mathcal{F}_{\text{Edit-Dist}}$ can indeed be implemented by any of the two strategies presented below. Therefore, our composite protocol securely implements the ideal functionality $\mathcal{F}_{\text{LDA-PSI}}$.

6. Computing the Edit Distance in SMPC

We describe two strategies. The first is garbling-based. It revolves around generalizing the idea of garbling to arithmetic circuits, in order to describe the main operations used by the **WF** algorithm. This strategy has been described by Zhu et al.[5]. The second involves secret sharing schemes, as well as the primitives of daBits and edaBits. Its starting point is, yet again, the **WF** algorithm. This strategy has been described by Vanegas et al.[6].

6.1. Garbling-based Computation of the Edit Distance

Zhu et al.[5] present a protocol to evaluate the *exact* edit distance in a secure and generic way. This is, however, not the first protocol of its kind. Wang et al. [21] show a different approach, based on heuristics, that is efficient, but prone to *input-specific* errors.
From the pseudocode of the Wagner-Fischer algorithm, it is clear that the four operations that the algorithm performs are **sum**, **comparison**, **minimum** and **table-lookup** (on the matrices t and D). The arithmetic circuit computing the edit distance will then use these same operations. Let us review how Zhu et al.[5] describe their oblivious execution.

Regarding the computation of the minimum, observe that, given integers a, b, c:

$$\min\{a, b, c\} = \min\{\min\{a, b\}, c\}$$

Let us define an "n-min gate" the arithmetic gate that computes the minimum among n inputs. The above observation tells us that any 3-min gate can be reduced to two 2-min gate. This is quite crucial as Zhu et al. show how the edit distance can be computed in such a way that, given any 2-min gate $\mathcal{M}(a, b)$, it holds that:

$$\exists \lambda_1, \lambda_2 : \lambda_1 \leq a - b \leq \lambda_2$$

where λ_1, λ_2 **are independent** of $|a|$ and $|b|$. That is to say, the difference between the inputs is bounded by constants that *do not* depend on the inputs themselves. This will result in an increase in efficiency for minimum gates.

Garbling. The underlying field is obviously not $\mathbb{Z}_2$ anymore, but rather a generic $\mathbb{Z}_p$, where p is any prime (or power of a prime). Let us assume that p is prime, and let us denote with $+_p$ and $\times_p$ the operations of addition and multiplication modulo p on this field. The garbler must now choose, uniformly at random and for each wire x, cryptographic keys k_z^x, representing integers $z \in \mathbb{Z}_p$. This can be made more efficient by sampling only two values. The garbler uniformly samples k_0^x, where 0 is the neutral element of $+_p$. Then, the garbler chooses Δ, as the *global secret* of the circuit. The key k_z^x, for $z \neq 0$, will be calculated as:

$$k_z^x \leftarrow k_0^x \ +_p \ z \ \times_p \ \Delta$$

[1]For the size-hiding issue see [20]

Notice how the uniformly sampled k_0^x and Δ perfectly mask z. Garbling the actual output table of a gate follows the exact same rule of binary garbling. Finally, we will refer to the wire label of wire x, on which value z is transmitted, as $w_z^x = 0^\eta || k_z^x$, where η is the security parameter and 0^η refers to a string of η 0s. Preposing the label allows the evaluator to confirm that the correct row has been deciphered without errors (otherwise, a garbage string would have been obtained).

Addition. Given $a, b \in \mathbb{Z}_p$ on wires A, B respectively, wire-labels are constructed:

$$w_a^A = 0^\eta \,||\, k_0^A \;+_p\; a \;\times_p\; \Delta$$
$$w_b^B = 0^\eta \,||\, k_0^B \;+_p\; b \;\times_p\; \Delta$$

The garbler also constructs $w_0^C = w_0^A +_p w_0^B$, so that on output wire C the evaluator computes:

$$w_c^C = w_a^A +_p w_b^B = 0^\eta \,||\, k_0^C \;+_p\; (a +_p b) \;\times_p\; \Delta$$

which is exactly the ciphertext that corresponds to $(a +_p b)$ on wire C. This means that addition is essentially free. No expensive cryptographic computation is performed. These considerations naturally extend to the multiplication by a constant. Letting w_a^A be the label of wire A and given a constant k, the garbler sets $w_0^C = k \times_p w_0^A$ and the evaluator computes:

$$w_c^C = k \times_p w_a^A = 0^\eta \,||\, k_0^C \;+_p\; (k \times_p a) \;\times_p\; \Delta$$

which is exactly the ciphertext corresponding to $k \times_p w_a^A$. Notice that, as long as $a + b < p$ (resp., $k \times a < p$), then $a + b = a +_p b$ (resp., $k \times a = k \times_p a$), and therefore the arithmetic in $\mathbb{Z}_p$ is realized correctly.

Equality Comparison. This operation is necessary in order to compute the terms of matrix t of the **WF** algorithm, which holds the costs of substituting a symbol for another in the two input strings. Let w_a^A and w_b^B be the labels of wires A and B on which values a and b are transmitted, respectively. Let $d = a - b$. Obviously it holds that $d = 0 \iff a = b$. We want to compute d and exploit the property that $d \in \{-\zeta, ..., +\zeta\}$. Garbling happens as usual:

- Let D be the output wire of the computation of d. We employ the previous construction for the addition operation to calculate d, so that the garbler computes $w_0^D = w_0^A - w_0^B$. The evaluator knows w_d^D.
- Let Z be the output wire of the equality comparison operation. The garbler uniformly samples w_0^Z e w_1^Z (representing "not equal" and "equal") and sends $2\zeta + 1$ ciphertexts in random order. Namely, $\text{Enc}_{w_0^D}(w_1^Z)$ and $\text{Enc}_{w_d^D}(w_0^Z),\ \forall d \in \{-\zeta, ..., +\zeta\}, d \neq 0$ are sent.
- The evaluator can decrypt the only ciphertext that was computed via the true value of w_d^D. Thus the evaluator will obtain as a label of the output wire $w_1^Z \iff d = 0 \iff a = b$ and w_0^Z otherwise.

This operation has linear cost with respect to the size of the interval to which $a - b$ belongs, i.e., $2\zeta + 1$. Since integer comparison has linear cost in the #bit used to represent a, b [22] this approach reduces complexity by a factor of $\min\{\log a, \log b\}$ with bounded $a - b$ [5].

Minimum. The comparison circuit is generalized in order to calculate the minimum between two elements. Let us define the quantity:

$$\langle x \rangle = \begin{cases} x & \text{, if } x \geq 0 \\ 0 & \text{, otherwise} \end{cases}$$

Then we can compute the minimum as:

$$\min\{a, b\} = a - \langle a - b \rangle$$

The minimum is then reduced to a subtraction, an evaluation of $\langle a - b \rangle$ and a second subtraction. Once more, this approach reduces time complexity by a factor of $\min\{\log a, \log b\}$ when $a - b$ is bounded by a constant $\pm\zeta$. Let us detail the circuit to evaluate the $\langle \cdot \rangle$ function:

- Let X be the input wire on which $x \in \{-\zeta, ..., +\zeta\}$ is sent; let Z be the output wire. The garbler sends the following $2\zeta + 1$ ciphertext in random order: $\text{Enc}_{w_i^X}(w_0^Z),\ \forall i \in \{-\zeta, ..., -1\}$ and $\text{Enc}_{w_i^X}(w_i^Z),\ \forall i \in \{0, ..., +\zeta\}$.
- The evaluator can decrypt the sole ciphertext obtained with the label associated to the value effectively sent through wire X. Thus, the evaluator will obtain w_0^Z if $\langle\cdot\rangle$ has a negative argument, i.e., if $i \in \{-\zeta, ..., -1\}$ and w_i^Z otherwise, i.e., if $i \in \{0, ..., +\zeta\}$.

Table-lookup. A unidimensional table can be viewed as an associative map in which, for every pair, the first entry is an index $0 \leq i \leq n-1$ and the second entry is the value a_i, stored in position i :

$$\{(0, a_0), (1, a_1), ..., (i, a_i), ..., (n-1, a_{n-1})\}$$

A table-lookup operation, can be formalized as a gate with one input wire I, with label w_i^I representing index i and output wire O labeled $w_{a_i}^O$. Garbling is carried forth as usual:

- The garbler samples $w_{a_i}^O$, for $0 \leq i \leq n-1$, uniformly at random; the garbler also generates n ciphertexts sent to the evaluator in random order: $\text{Enc}_{w_i^I}(w_{a_i}^O), \forall i \in \{0, ..., n-1\}$.
- The evaluator knows the encryption of the input, w_i^I, and decrypts the only ciphertext obtained with w_i^I, which is $w_{a_i}^O$.

In order to generalize the above to two-dimensional tables, it suffices to transform t of size $m \times n$, in a unidimensional table t' of size mn: every index (i, j) of t is mapped to index $im + j$ of t'. We are essentially concatenating multiple rows of t in a single term of t'.

Initial inputs. We assume that the initial circuit inputs to the arithmetic circuit are in bits and the processing of these binary input values resembles that in binary garbled circuit protocols, i.e., the circuit generator's private input bits are encoded by wire-labels that are directly sent to the evaluator while the circuit evaluator's private input bits are translated to their corresponding wire-labels through oblivious transfer. See [5] for details.

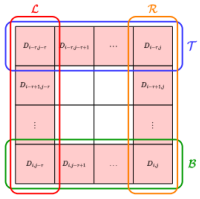
Complexity. The dominant cost for time complexity comes from comparisons and minimum calculations. As reported by Zhu et al., the described approach reduces complexity by a factor of $O(\max\{\log m, \log n\})$ [5], where $m \times n$ is the size of the matrix used by the **WF** algorithm.

6.2. SSS-based Computation of the Edit Distance

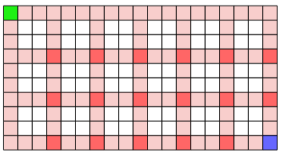
In this section we focus on the work of Vanegas et al.[6], who also report the performance of their construction against malicious adversaries, compared to what is done by Zhu et al. To simplify the presentation, we will not alter their assumption of working with nucletotides, i.e., with the alphabet {A,C,T,G}. The solution uses *daBits* and *edaBits* in order to convert binary shares $[\![x]\!]_2$ in arithmetic shares $[\![x]\!]_{2^k}$, and for the efficient computation of integer comparisons, respectively. Implementation is delegated to protocols such as TinyOT for binary operations and SPD$\mathbb{Z}_{2^k}$ for arithmetic ones [6]. This is not a limitation: the protocols for generating daBits and edaBits treat SMPC as a black box: thus, in order to instance the solution of Vanegas et al., any protocol that can instance $\mathcal{F}_{\text{edaBits}}$ [17] can be leveraged.

The protocol follows two main phases: during the **preamble**, the matrix t holding the substitution costs for the **WF** algorithm is computed. This happens on domain $\mathbb{Z}_2$. During the **arithmetic part**, the **WF** algorithm is executed in the SMPC context, i.e., the matrix D will be computed, on domain $\mathbb{Z}_{2^k}$.

Preamble. Given the alphabet $\{A, C, G, T\}$, observe that two bits are sufficient to encode every symbol. For instance, we can arbitrarily define $A := 00$, $C := 01$, $G := 10$, $T := 11$. Let $\langle b_0, b_1\rangle$ the two bits that encode the symbol N. Its sharing is then $[\![N]\!]_2 = \langle [\![b_0]\!]_2, [\![b_1]\!]_2\rangle$. In order to make comparison efficient, let us extend the XOR operations to the nucleotides: $N \oplus N' = \langle b_0 \oplus b_0', b_1 \oplus b_1'\rangle$. Notice



(a) Positions inside a $(\tau+1)$-box.



(b) Division of the edit distance matrix into boxes.

that $N \oplus N' = 0 \iff N = N'$, and thus XOR acts as an equality test. We then want to determine if $(N \oplus N')$ is 0. Let us define:

$$S(N) = b_0 \vee b_1 = (b_0 + b_1 + b_0 b_1) \mod 2$$

as the logical OR of the bits. Thus, it holds: $N = N' \iff N \oplus N' = 0 \iff S(N \oplus N') = 0$. The test should return 1 if the symbols are equal, so we merely invert the result. Denoting with $[\![N \stackrel{?}{=} N']\!]_2$ the equality test between N and N', we get:

$$\begin{aligned}[\![N \stackrel{?}{=} N']\!]_2 &= [\![1 - S(N \oplus N')]\!]_2 \\ &= 1 - [([\![b_0]\!]_2 + [\![b'_0]\!]_2) + ([\![b_1]\!]_2 + [\![b'_1]\!]_2) + ([\![b_0]\!]_2 + [\![b'_0]\!]_2)([\![b_1]\!]_2 + [\![b'_1]\!]_2)]\end{aligned}$$

Given n, m lengths of the input strings, matrix t has size nm. Therefore, we can compute t with nm products in $\mathbb{Z}_2$ transmitting 4 bits (b_0, b_1, b'_0, b'_1) to the other party, i.e., $4nm$ bits in totals.

Arithmetic Part. We can use *daBits* to transform binary shares $[\![\cdot]\!]_2$, in arithmetic shares $[\![\cdot]\!]_{2^k}$. From this point on, the parties will hold shares of the form: $[\![t_{ij}]\!]_{2^k} \quad \forall (i,j)$. Recall that we want to compute the terms of matrix D of the **WF** algorithm: in particular, we seek value $D(n,m)$ which is the edit distance between the input strings $a_1, ..., a_n$ and $b_1, ..., b_m$. The work of Vanegas et al.[6] also revolves around designing an efficient computation for D. Not every term of D is necessary in order to evaluate $D(n,m)$. The idea is based on approaches such as the one of Cheon et al. [23]. Consider the minimum figuring in the recurrence for D: $\min\{D(i-1,j), D(i,j-1), D(i-1,j-1) + t(i,j)\}$. Apply it recursively for the three terms and the following formulas will be obtained:

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)**

The bold terms are provably smaller [6] than some other in the list and thus computing the others is unnecessary. Notice how the terms depending on $D(i-1,j-1)$ disappear. This holds in general: it is always possible to express $D(i,j)$ as minimum of the sole terms that lie on the "border" of the rectangle of terms whose bottom-right corner is $D(i,j)$ [6]. Formally, the $(\tau+1)$-box of $D(i,j)$, where τ is a non-negative integer, is defined as the union of the following:

$$\mathcal{T} = \{D(i-\tau, j-\tau), D(i-\tau, j-\tau+1), ..., D(i-\tau, j)\}, \quad \mathcal{B} = \{D(i, j-\tau), D(i, j-\tau+1), ..., D(i,j)\}$$
$$\mathcal{L} = \{D(i-\tau, j-\tau), D(i-\tau+1, j-\tau), ..., D(i, j-\tau)\}, \quad \mathcal{R} = \{D(i-\tau, j), D(i-\tau+1, j), ..., D(i,j)\}$$

Thus, we want to express $D(i,j)$ as minimum of formulas in $\mathcal{T} \cup \mathcal{L} \cup \mathcal{B} \cup \mathcal{R}$. Notice the redundance between these sets. In fact, the terms in $\mathcal{B} \cup \mathcal{R}$ can in turn be expressed as the minimums of terms in $\mathcal{T} \cup \mathcal{L}$[6]. Vanegas et al. supply us with a helpful, intuitive visual description $(\tau+1)$-boxes: Figure (a) shows the above sets relating to $D(i,j)$. Figure (b) shows $D(n,m)$, our term of interest in blue, the $(\tau+1)$-boxes in pink and the "most complex" terms to compute in red, since they are at the bottom-right corner of their respective boxes, assuming to have fixed the value of τ. It is entirely unnecessary to evaluate the white terms.

The protocol makes use of the MIN_q routine, as defined in [24] and [16]. It can compute the minimum between q shared integers, with $q-1$ comparisons and $2(q-1)$ multiplications, in $O(\log q)\cdot(r+1)$ round, where r is the #rounds required for a comparison. The work of Vanegas et al. also employs a clever approach for the automatic generation of fomulas in the $(\tau+1)$-boxes. In this context, the minimum number of formulas to evaluate is given as $O(\tau\cdot 2^{3\tau})$ using nm/τ^2 boxes, where nm is the size of D. Given the complexity of MIN_q, the protocol is run in $O(nm/\tau^2\cdot(3\tau+\log\tau)\cdot(r+2))$ rounds and makes $O(nm/\tau^2\cdot(\tau^2\cdot 2^{3\tau}-\tau))$ comparisons as well as $O(nm/\tau^2\cdot(\tau^2\cdot 2^{3\tau+1}-2\tau))$ multiplications. Once completed, the parties will hold shares of the form $[\![D(n,m)]\!]_{2^k}$ that can be revealed to each other (with protocols such as OPEN [25]). A key takeaway is the role of τ as a hyperparameter: it induces a tradeoff between the number of complex operations (and sent data) i.e., comparisons and products and the number of rounds.

7. Comparing the Two Strategies

Assuming we restrict ourselves to the semi-honest model, let us summarize the parameters we need to compare the two approaches. Let n, m be the lengths of Alice's and Bob's input (respectively). Let $q=|\Sigma|$ be the size of the alphabet. Thus, we need $b=\log q$ bits to encode each symbol (with fixed length encoding). Let C be the number of pairs to compare: in our case, we compare every string in Alice's set to every string in Bob'set; therefore $C=|A|\cdot|B|$.

In the garbling-based approach we use the $\langle\cdot\rangle$ projection operator. For each entry of $D_{i,j}$ we use an 8-row projection gate [5], and one more for computing the minimum. This schema requires $16nm$ total garbled rows. Each row is an 16 B AES block, thus the cumulative size of the garbled rows is $16\cdot 16nm = 256nm$ B. The scheme requires an OT per symbol, costing $O(m)$ for selections. Referring to the ABY framework [26] and to Zhu et al.[5] we can concretely estimate the time needed for domain conversion to sending $12B$ in 6μs, which is negligible with respect to the previous quantity $O(nm)$.

In the SSS-based approach $2bnm$ products are executed (in parallel) during the preamble. Let c_r be the #rounds required for a comparison, with respect to the MIN_q primitive. During the preamble we need to reveal $2b$ bit for every pair of symbols, and for each of the nm entries of matrix t - sending $2bnm$ bits in total. Furthermore, let $y \overset{\text{def.}}{=} \frac{nm}{\tau^2}$ be the #boxes. The # rounds executed during the arithmetic part is: $R_{\text{arit}} \overset{\text{def.}}{=} y\cdot\left(3\tau+\log\tau^2\right)\cdot(c_r+2)$. Let $N_{\text{mul}} \overset{\text{def.}}{=} y\cdot\left(\tau^2\cdot 2^{3\tau+1}-2\tau\right)$ be the number of required multiplications (using Beaver triples we need to send 2 shares per product). Operating in $\mathbb{Z}_{2^k}$ where 2^k is the smallest power of 2 that is greater than 2^b. Thus, if shares are k bit long, multiplications require sending $2k$ bit i.e. $1/4\cdot k$ byte. Finally, let $N_{\text{cmp}} \overset{\text{def.}}{=} y\cdot\left(\tau^2\cdot 2^{3\tau}-\tau\right)$ be the #comparisons, using $O(k)$ bits to generate edaBits.
The communication cost (in byte) and time complexity for each pair $(a_1...a_n, b_1...b_m)$ with the GC approach is then given by:

$$\text{COMM}_{GC}(n,m)=256nm+O(m) \qquad \text{T}_{GC}(n,m)=c\cdot RTT=O(RTT)$$

where c is a constant and RTT denotes the Round-Trip Time for remote communication. Vanegas et al.[6] note that the time needed for preprocessing is negligible with respect to the online phase of their protocol, thus for the SSS-based approach we observe:

$$\text{COMM}_{SSS}(n,m,k,\tau)=2bnm+1/4\cdot k\cdot N_{\text{mul}}+O(k)\cdot N_{\text{cmp}} \qquad \text{T}_{SSS}(n,m,\tau)=O(\text{RTT})\cdot R_{\text{arit}}$$

Two exemplifying cases are detailed in Appendix A.

8. Conclusions

In order to evaluate the feasibility of our solution to the LDA-PSI problem, we studied two strategies to compute the edit distance in the SMPC setting. The garbling-based strategy revolves around implementing addition, equality comparison, minimum and table-lookup in order to construct the Wagner-Fischer

algorithm with garbled circuits. The SSS-based solution uses a sophisticated algorithmic approach to automatically generate the formulas to be evaluated, discarding unnecessary terms. This solution makes use of daBits and edaBits for domain conversion and integer comparison.

The results of our comparison are quite evident: the time complexity of the garbling-based approach depends strictly on the specification of the network as the strategy requires a constant number of rounds. However, the communication complexity is considerable, as the approach involves sending the garbled rows from one party to another. The SSS based approach uses τ as a hyperparameter that regulates time and communication complexity. The number of formulas to compute grows exponentially with τ while the #boxes (and thus the time) reduces quadratically with τ growing. The theoretical comparison in Appendix A shows this, as well as it remarks how choosing a good fit for τ is crucial. Too high a value and the advantage gained in terms of time is lost, due to the exponential growth of the number of formulas. With the right value ($= 1$ in Case 2) the data sent is more than halved, compared to the GC-based approach. However, it is now evident how this does not imply the strategies can be employed for brute-forcing the quasi-intersection of the input sets. Two sets made of the 16-character strings on the English alphabet (26 symbols), require $26^{16} \times 26^{16} = 26^{32} \approx 1.9 \cdot 10^{45}$ comparisons, too many for both strategies. Even shorter strings, e.g., 5 on an alphabet of size 8 – an all-too-elementary case – would require with the GC-based approach and a LAN network with RTT $= 0.2$ ms, around 60 hours. We conclude that, in spite of the progress made by SMPC, a naive approach to this problem is hardly feasible – although for reasonably small sets of strings, the garbling approach allows to perform computation swiftly. The SSS-based approach requires less bandwidth, provided a good fit for the value of τ, which would otherwise herald an excess of formulas to be evaluated.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] M. R. David Evans, Vladimir Kolesnikov, A Pragmatic Introduction to Secure Multi-Party Computation, NOW Publishers, 2018.

[2] P. D'Arco, A. D. Santis, Private computations on set intersection, in: C. J. Colbourn, J. H. Dinitz (Eds.), New Advances in Designs, Codes and Cryptography, Springer Nature Switzerland, Cham, 2024, pp. 77–95.

[3] A. Chakraborti, G. Fanti, M. K. Reiter, Distance-aware private set intersection, in: Proceedings of the 32nd USENIX Conference on Security Symposium, SEC '23, USENIX Association, USA, 2023.

[4] P. D'Arco, C. Maione, S. Masullo, R. Zizza, Levenshtein distance-aware private set intersection (manuscript), ????

[5] R. Zhu, Y. Huang, Efficient privacy-preserving general edit distance and beyond, Cryptology ePrint Archive, Paper 2017/683, 2017. URL: https://eprint.iacr.org/2017/683.

[6] H. Vanegas, D. Cabarcas, D. F. Aranha, Privacy-preserving edit distance computation using secret-sharing two-party computation, in: A. Aly, M. Tibouchi (Eds.), Progress in Cryptology – LATINCRYPT 2023, Springer Nature Switzerland, Cham, 2023, pp. 67–86.

[7] M. J. F. Robert A. Wagner, The string-to-string correction problem, Journal of the Association for Computing Machinery Vol. 21 (????).

[8] Y. Ishai, J. Kilian, K. Nissim, E. Petrank, Extending oblivious transfers efficiently, in: D. Boneh (Ed.), Advances in Cryptology - CRYPTO 2003, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 145–161.
[9] R. K. Kolesnikov V., Improved ot extension for transferring short secrets, Advances in Cryptology – CRYPTO 2013, Part II. (2013).
[10] A. C.-C. Yao, How to generate and exchange secrets, in: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986), 1986, pp. 162–167. doi:10.1109/SFCS.1986.25.
[11] A. C. Yao, Protocols for secure computations, in: 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), 1982, pp. 160–164. doi:10.1109/SFCS.1982.38.
[12] A. Shamir, How to share a secret, Commun. ACM 22 (1979) 612–613. URL: https://doi.org/10.1145/359168.359176. doi:10.1145/359168.359176.
[13] R. Cramer, I. Damgård, D. Escudero, P. Scholl, C. Xing, SPDZ2k: Efficient MPC mod 2^k for dishonest majority, Cryptology ePrint Archive, Paper 2018/482, 2018. URL: https://eprint.iacr.org/2018/482.
[14] T. K. Frederiksen, M. Keller, E. Orsini, P. Scholl, A unified approach to MPC with preprocessing using OT, Cryptology ePrint Archive, Paper 2015/901, 2015. URL: https://eprint.iacr.org/2015/901.
[15] D. Rotaru, T. Wood, MArBled circuits: Mixing arithmetic and boolean circuits with active security, Cryptology ePrint Archive, Paper 2019/207, 2019. URL: https://eprint.iacr.org/2019/207.
[16] I. Damgård, D. Escudero, T. Frederiksen, M. Keller, P. Scholl, N. Volgushev, New primitives for actively-secure mpc over rings with applications to private machine learning, in: 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 1102–1120. doi:10.1109/SP.2019.00078.
[17] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, P. Scholl, Improved primitives for MPC over mixed arithmetic-binary circuits, Cryptology ePrint Archive, Paper 2020/338, 2020. URL: https://eprint.iacr.org/2020/338.
[18] O. Goldreich, Foundations of Cryptography Volume II Basic Applications, Cambridge University Press, 2004.
[19] Y. Lindell, How to simulate it - a tutorial on the simulation proof technique, Cryptology ePrint Archive, Paper 2016/046, 2016. URL: https://eprint.iacr.org/2016/046.
[20] P. D'Arco, M. I. G. Vasco, A. L. P. del Pozo, C. Soriente, R. Steinwandt, Private set intersection: New generic constructions and feasibility results, Advances in Mathematics of Communications 11 (2017) 481–502.
[21] X. Wang, Y. Huang, H. Tang, X. Wang, D. Bu, Efficient genome-wide, privacy-preserving similar patient query based on private edit distance, 2015, pp. 492–503. doi:10.1145/2810103.2813725.
[22] V. Kolesnikov, A.-R. Sadeghi, T. Schneider, Improved garbled circuit building blocks and applications to auctions and computing minima, 2009. URL: https://eprint.iacr.org/2009/411.
[23] J. H. Cheon, M. Kim, K. Lauter, Homomorphic computation of edit distance, Cryptology ePrint Archive, Paper 2015/132, 2015. URL: https://eprint.iacr.org/2015/132.
[24] T. Toft, Primitives and applications for multi-party computation, 2007.
[25] R. Cramer, I. B. Damgård, J. B. Nielsen, Secure Multiparty Computation and Secret Sharing, Cambridge University Press, 2015.
[26] M. Z. Daniel Demmler, Thomas Schneider, Aby – a framework for efficient mixed-protocol secure two-party computation, 2015. URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/08_2_1.pdf.
[27] Ishai, Kilian, Nissim, Petrank, Extending oblivious transfers efficiently, 2003. URL: https://www.iacr.org/archive/crypto2003/27290145/27290145.pdf.
[28] M. Keller, E. Orsini, P. Scholl, Actively secure OT extension with optimal overhead, Cryptology ePrint Archive, Paper 2015/546, 2015. URL: https://eprint.iacr.org/2015/546. doi:10.1007/978-3-662-47989-6_35.
[29] X. Wang, S. Ranellucci, J. Katz, Authenticated garbling and efficient maliciously secure two-party computation, 2017. URL: https://eprint.iacr.org/2017/030.pdf.
[30] A. Malozemoff, X. Wan, Emp-toolkit, ???? URL: https://github.com/emp-toolkit.
[31] K. O., S. A., Secure computation with fixed-point numbers, 2010. URL: ifca.ai/pub/fc10/31_47.pdf.
[32] M. Keller, MP-SPDZ: A versatile framework for multi-party computation, Cryptology ePrint

Archive, Paper 2020/521, 2020. URL: https://eprint.iacr.org/2020/521. doi:10.1145/3372297.3417872.

A. Comparison Case 1: Comparing Short Words

Let $\Sigma = \{a, b, ..., z\}$ bet the alphabet of the 26 English letters. Thus we use $b = \lceil \log(26) \rceil = 5$ bits per letter. We consider words on Σ of 16 letters at most. Let us employ the garbling based approach, using the semi-honest model and without conversions. We need to consider $16nm = 16n^2 = 16(16^2) = 4096$ rows; using 128 bit wire-labels, we need $B_{\text{row}} = 16$ bytes per row, for a grand total of 4096×16 B $= 65536$ B $= 64$ KB sent. This represents the dominant cost for communication. The cost of the oblivious transfer operation depends strictly on implementation. The classic IKNP framwork[27][28] needs $B_{\text{OT}} = 2(\kappa + 1)$ bits per bit sent, where $\kappa = 128$ is the security parameter. Thus, in our case, $258 \cdot bm = 258 \cdot 5 \cdot 16 = 20640$ bits are transmitted,i.e. ≈ 3 KB.
Therefore, it is required to transmit $\text{COMM}_{GC}(n, m) = 256 \cdot 16 \cdot 16 = 65,536$ B $= 64$ KB for each pair.

Furthermore, we need time $t_{\text{row}} = 86$ ns (43 ns for each side of the communication) to process each row with the approach of Wang et al. [29], whose implementation [30] is used by Zhu et al.[5]. For OT, using IKNP[27][28],we need approximately ≈ 150ns. In other words, executing the protocol on a LAN with RTT $= 0.2$ ms, time is dominated by RTT as we require $\text{T}_{GC}(n, m) = c \cdot 0.2$ ms for each pair.

For the SSS-based approach, let us consider $\tau \in \{1, 3, 5\}$ as typical values for τ [6]:

- Given $\tau = 1$ we have:
  $k = \frac{nm}{\tau^2} = \frac{16 \cdot 16}{1} = 256$ boxes.

  $N_{\text{mul}} = k\left(\tau^2 \cdot 2^{3\tau+1} - 2\tau\right) = 256 \cdot \left(1^2 \cdot 2^{3 \cdot 1+1} - 2 \cdot 1\right) = 3584.$

  $N_{\text{cmp}} = k\left(\tau^2 \cdot 2^{3\tau} - \tau\right) = 256 \cdot \left(1^2 \cdot 2^{3 \cdot 1} - 1\right) = 1792.$

  $R_{\text{arit}} = 256\left(3 \cdot 1 + \log 1^2\right) \cdot (3 + 2) = 3840$, using $c_r = 3$, if we employ the strategy of Catrina et al. [31] for comparison. For concrete values, let us consider SPDZ_{2^k}[16] [13], used for the implementation of the MP-SPDZ [32] employed by Vanegas et al. [6]. Given out security+statistical parameter 128 bits, Tinier requires ≈ 64 B of bandwidth to compute an AND gate [29]. Thus, executing on LAN with RTT = 0.2 ms, we have:

  $\text{COMM}_{SSS}(n, m, \tau, k) = 2 \cdot 5 \cdot 16 \cdot 16 + 1/4 \cdot 8 \cdot 3584 + 8 \cdot 1792 = 24,064$ B ≈ 24 KB.
  $\text{T}_{SSS}(n, m, \tau) = 0.2 \cdot 3840 = 768$ ms.

- Given $\tau = 3$:
  $k = \frac{nm}{\tau^2} = \frac{18 \cdot 18}{9} = 36$ boxes, having increasead n, m to the nearest larger multiple of τ.

  $N_{\text{mul}} = 331,560.$

  $N_{\text{cmp}} = 165,780.$

  $R_{\text{arit}} \approx 1837.$

  Therefore:
  $\text{COMM}_{SSS}(n, m, \tau, k) = 2 \cdot 5 \cdot 18 \cdot 18 + 1/4 \cdot 8 \cdot 331,560 + 8 \cdot 165,780 = 1,992,600$ B ≈ 1.9 MB.
  $\text{T}_{SSS}(n, m, \tau) = 0.2 \cdot 1837 \approx 367$ ms.

- Given $\tau = 5$:
  $k = \frac{nm}{\tau^2} = \frac{20 \cdot 20}{25} = 16$ box.

  $N_{\text{mul}} = 104,857,440.$

  $N_{\text{cmp}} = 13,107,120.$

  $R_{\text{arit}} \approx 1631.$

  Therefore:
  $\text{COMM}_{SSS}(n, m, \tau, k) = 2 \cdot 5 \cdot 20 \cdot 20 + 1/4 \cdot 8 \cdot 104,857,440 + 8 \cdot 13,107,120 = 314,575,840$ B ≈ 300 MB.
  $\text{T}_{SSS}(n, m, \tau) = 0.2 \cdot 1631 = 326$ ms.

The results are coherent to what has been experimented by Vanegas et al.[6] with words of 16 bit. The experimental values reported are all relative to networks with similar performance values, i.e. a bandwidth of $1 - 1.6$ Gbps and $0.1 - 0.3$ ms of RTT.

B. Comparison Case 2: Comparing Longer Words

Consider $n = m = 128$. Let us observe, in the GC-based approach, the variation which the communication complexity undergoes:

$$\text{COMM}_{GC}(n, m) = 256 \cdot 128 \cdot 128 + O(m) \approx 4,194,304\text{B} = 4\text{MB}$$

Similarly:

- $\tau = 1$:
  $k = \frac{nm}{\tau^2} = \frac{128 \cdot 128}{1} = 16,384$ boxes.

  $N_{\text{mul}} = 229,376.$
  $N_{\text{cmp}} = 114,688.$

  $\text{COMM}_{SSS}(n, m, \tau, k) = 2 \cdot 5 \cdot 128 \cdot 128 + 1/4 \cdot 8 \cdot 229,376 + 8 \cdot 114,688 = 1,540,096$ B ≈ 1.47 MB.

- $\tau = 3$:
  $k = \frac{nm}{\tau^2} = \frac{129 \cdot 129}{9} = 1,849$ boxes.

  $N_{\text{mul}} = 17,029,290.$
  $N_{\text{cmp}} = 8,514,645.$

  $\text{COMM}_{SSS}(n, m, \tau, k) = 2 \cdot 5 \cdot 129 \cdot 129 + 1/4 \cdot 8 \cdot 17,029,290 + 8 \cdot 8,514,645 = 102,342,150$ B ≈ 97.6 MB.

- $\tau = 5$:
  $k = \frac{nm}{\tau^2} = \frac{130 \cdot 130}{25} = 676$ boxes.

  $N_{\text{mul}} = 1,107,551,640.$
  $N_{\text{cmp}} = 553,775,820.$

  $\text{COMM}_{SSS}(n, m, \tau, k) = 2 \cdot 5 \cdot 130 \cdot 130 + 1/4 \cdot 8 \cdot 1,107,551,640 + 8 \cdot 553,775,820 = 6,645,478,840$ B ≈ 6.2 GB.