Efficient Secure Computation of Edit Distance on Genomic Data

Andrea Migliore[a], Stelvio Cimato[b] and Gabriella Trucco[c]
Department of Computer Science, Università degli Studi di Milano, Milan, Italy

Keywords: Secure Multi-Party Computation, Garbled Circuit, Edit Distance.

Abstract: Genetic data are the most sensitive information for a person, containing many specific features that uniquely determine an individual and also make it possible to trace relationships with other people or evaluate the predisposition to particular diseases. For this reason, any processing of genetic data should be carefully performed and any threat to their privacy properly considered. A very important computation in medical and public health domains involves the evaluation of the edit distance between human genomes, that can eventually lead to a better diagnosis of several diseases. To maintain the privacy of the genetic data, it is possible to apply secure computation protocols and then, in this context, the improvement of the computational performance of such techniques is a key factor for real-world application scenarios.
In this paper we focus on the application of the garbling circuit technique for the computation of the edit distance, showing its efficiency. We apply the technique considering four different algorithms and compare their performances to the best previous results found in literature. We show that the Ukkonen algorithm with generalized cut-off is the one that performed better among the considered algorithms, reporting some experimental results obtained considering datasets composed of both randomly generated and real genomic strings.

1 INTRODUCTION

Bionformatics is a rapidly advancing field, where the application of information technology to the treatment of biological data is helping in better analyzing and understanding the various types of data resulting from different biological processes. Last years have registered a huge advance in speed and cost reduction, allowing the completion of the Human Genome Project and the possibility to sequence a full genome for a small amount of money.

However the intersection of genomics and security arises stimulating ethical and social issues that need to be addressed, since genetic information can be considered *the* most sensitive data for a person and consequently, it must be protected from any kind of malicious attack or disclosure. Since genomes are usually represented as strings, the computation of the similarity between genomes can be easily mapped to the application of efficient methods for string comparison. In particular, the computation of different metrics such as edit distance or Needleman-Wunsch distance can help in the diagnosis of several genetic diseases.

A natural way to protect genetic data is to ensure that computations are performed on encrypted strings, so that sensitive information remain protected even in case of misbehaviour from one of the collaborating parties. *Secure Multi-Party Computation* (SMPC) (Yao, 1982) is a branch of cryptography whose goal is to enable a group of independent data owners, who do not trust each other or any common third party, to jointly compute a function that depends on all of their private inputs. SMPC can be defined as the problem of n players who want to compute an agreed function of their inputs in a secure way. Formally, we assume $x_1, \ldots, x_n$ inputs, where player i knows x_i, and we want to compute $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ such that player i is guaranteed to learn y_i and nothing more than that.

There are a number of efficient implementation of these protocols based on ad-hoc techniques developed to solve specific problems or on the generic transformation of the computed function, such as in the case of *garbled circuits* protocol (Yao, 1986) (Micali et al., 1987). Secure multi-party computation can be used to solve a wide variety of real-life problems where sensitive data may be compromised.

In this paper, we focus on the application of secure multi-party computation to bioinformatics and more

[a] https://orcid.org/0009-0008-1494-9547
[b] https://orcid.org/0000-0003-1737-6218
[c] https://orcid.org/0000-0002-5904-0969

Migliore, A., Cimato, S. and Trucco, G.
Efficient Secure Computation of Edit Distance on Genomic Data.
DOI: 10.5220/0012459400003648

In *Proceedings of the 10th International Conference on Information Systems Security and Privacy (ICISSP 2024)*, pages 878-883
ISBN: 978-989-758-683-5; ISSN: 2184-4356

specifically to the computation of the edit distance, generically used to compute the distance between two strings. The human genome is composed by two complementary strands, with 3 billion DNA bases each. Each unit consists of Adenine (A), Cytosine (C), Guanine (G), or Thymine (T), the nitrogenous bases that constitute nucleotides.

Edit distance is an important metric to quantify how dissimilar two DNA strings are, by counting the minimum number of operations required to transform one string into the other. It is one of the most used and well established metrics among genetic similarity indicators because it is very useful for the diagnosis and treatment of many genetically based diseases such as cancer, Alzheimer's, schizophrenia and others (Koboldt et al., 2012) (Taylor et al., 2001) (Waddell et al., 2015) (Evans and Relling, 2004).

In this work, we propose some secure implementations of the most efficient edit distance algorithms, achieving better performances over existing protocols found in literature, without sacrificing security and accuracy. Moreover, a secure implementation of the *Ukkonen*'s algorithm with the generalized cut-off technique using all possible state-of-the-art optimizations for garbled circuits is proposed. Also, the proposed algorithms are applied on both random, high-entropy, and real genomics, low-entropy strings and are provably secure with respect to the standard definition of security for SMPC protocols. Finally, the methodology, the experiment setup details, and the source codes used for this study are described and provided, defining a clear baseline for future works and enabling anyone to reproduce the experiments independently.

The paper is organized as follows: first we introduce the basic notions about edit distance and secure multi-party computation.

Next, we describe the optimized techniques for the computation of the edit distance, and their implementation in SMPC. Finally we report the basic results and draw some conclusions.

2 BACKGROUND

We now provide some basic definitions and quick overview of the algorithms used for the computation of edit distance.

2.1 Edit Distance

Given two strings s_1 and s_2 on an alphabet Σ, the *edit distance* $d(s_1, s_2)$ is the minimum number of edit operations (insertion, deletion and substitution) required to transform s_1 into s_2 and measures the similarity between the two sequences. An edit operation is the basic step in transforming a word into another word. There are different types of edit distance that allow different combinations of editing operations. In computational biology, one of the most frequently used type of edit distance is the **Levenshtein edit distance** (Levenshtein et al., 1966).

In more details, a word over the finite alphabet Σ is a sequence $a_1, ..., a_n$ of symbols, where $a_i \in \Sigma$ for $i = 1, 2, ..., n$. The empty word is denoted by the symbol ε. An edit operation is a pair (a, b) with $a, b \in \Sigma \cup \{\varepsilon\}$ and $ab \neq \varepsilon$ and it is a basic step in transforming a word into another word. There are three types of operations: insertion, deletion and substitution. The edit operation is called *insertion* if $a = \varepsilon$, *deletion* if $b = \varepsilon$, and *substitution* if $a \neq \varepsilon \neq b$.
Each of these operations has a cost associated with it. Usually, the cost $c(a \to b)$ for $a \neq b$ is 1, whereas for $a = b$ it is 0. This results in a cost equal to 1 in the case of insertion, deletion or replacement and a cost equal to 0 when the two letters are the same.
An edit sequence S is a sequence of edit operations $S = ((a_1, b_1), ..., (a_n, b_n)), n \geq 1$. The final cost of an edit sequence S is defined as $C(S) = \sum_{i=1}^{n} c(a_i, b_i)$ (Aziz et al., 2017).

In 2014 it has been proved (Backurs and Indyk, 2014) that the Levenshtein distance of two strings of length n cannot be computed in time $O(n^{2-\mu})$ for any μ greater than zero unless the strong exponential time hypothesis (SETH) is violated. This issue introduces strong limitations on the research for exact optimization of edit distance because, despite $O(n^2)$ is a polynomial time, it is not acceptable for specific applications or for very long strings.

2.2 Algorithms for Computing the Edit Distance

Over the years, several exact algorithms, which seek the optimal answer to the problem without considering margins of error or specific conditions, have been proposed. The most famous exact algorithm is the *Wagner-Fischer algorithm* (Wagner and Fischer, 1974), based on dynamic programming that has a time complexity of $O(nm)$, where n and m are the string lengths to compare.

2.3 Wagner-Fischer Algorithm

The Wagner-Fischer algorithm is based on dynamic programming and starts from the observation that it is possible to create a matrix with a number of rows and columns corresponding to the length of the

two strings to be processed. The final edit distance $D(n,m)$, where n and m represent the lengths of the two strings respectively, is obtained by resolving all the edit distances of the substrings that constitute the final strings. In other words, the computation proceeds with each $D(i,j)$ where i and j are values smaller than n and m. The key idea is to solve all the sub-problems relying on the values obtained from the previous computation. Basically, we have to compute $D(i,j)$ for each $0 < i < n$ and $0 < j < m$. Each cell represents the edit distance of two substrings and, consequently, one of these sub-problems.

Since exact algorithms require at least $O(nm)$ operations to compute the Levenshtein distance, over the years scientific research has been heavily involved in the search for better alternatives through approximate algorithms that are generally faster (Hyyrö, 2003).

2.4 Ukkonen Algorithm

Typically, approximation algorithms search for approximate matches of a pattern from a string using also a predetermined maximum error threshold that indicates the maximum edit distance allowed for an approximate match.

The most important algorithm in this category is the *Ukkonen's algorithm* (Ukkonen, 1985), that optimizes the computation of edit distance by trying to restrict the number of cells that must be filled in the dynamic programming table. Given m and n the two string lenghts, and i, j the coordinates of any cell in the matrix, from the diagonal and adjacency properties, Ukkonen observed that if $D(i,j) \leq k$ and $m \leq n$, then it is sufficient to fill only the cells in the diagonals: $-\lfloor\frac{(k-n+m)}{2}\rfloor, -\lfloor\frac{(k-n+m)}{2}\rfloor+1, \ldots, \lfloor\frac{(k+n-m)}{2}\rfloor$ of the dynamic programming matrix. He concluded that the $d(i,j)$ values form a non-decreasing sequence along any given diagonal, i.e. $d(i,j)-1 \leq d(i-1,j-1) \leq d(i,j)$ consequently, it's necessary to calculate only the values that do not exceed the chosen threshold k. Once $D(i,j) > k$, the cells $D(i+h, j+h)$ where $h \geq 0$ are irrelevant for computation purposes. This technique is called *cut-off* because, intuitively, it cuts out unnecessary values.

If we are not interested in an edit distance greater than some maximum threshold k, then it is necessary to calculate only the diagonal band of width $2k+1$ of the matrix because the other cells are irrelevant for the computation.

2.5 Generalized Ukkonen

An advantageous optimization of the Ukkonen's algorithm with a generalization of the *cut-off* technique, described and used in the final experiment, leverages an implicit upper bound of the Levenshtein distance and can be used without a pre-specified threshold parameter k (Ukkonen, 1985) (Seiji, 2019). This optimization is rarely, if ever, used in literature although it turns an approximation algorithm into an exact one.

	ε	A	C	G	T
ε	0	1			
G	1	1	2		
A		1	2	3	
C			1	2	3
T				2	2

Figure 1: Generalized Ukkonen.

Suppose we have two strings S and T whose lengths are n and m respectively, with $m \geq n$. We can guarantee that the Levenshtein distance cannot exceed $LevenshteinDist(S,T) \leq m$.
If we define the two strings as $S = S_1S_2\ldots S_{n-1}S_n$ and $T = T_1T_2\ldots T_{m-1}$, since $m \geq n$, we can rewrite $S = S_1S_2\ldots S_{n-1}S_n$ and $T = T_1T_2\ldots T_{n-1}T_nT_{n+1}\ldots T_m$. Now we can convert T into S by replacing the sequence $T_1\ldots T_n$ with $S_1\ldots S_n$ and deleting $T_{n+1}\ldots T_m$. The cost of performing this operation is at most m because it needs n substitution and $(m-n)$ deletions. Hence, the upper bound is proved.

This upper bound is quite useful because it allows fewer cells to be computed without specifying a precise threshold. Let X_c be the Manhattan distance from a cell to the upper right corner. Then, as a rule, we can say that as long as the expression

$$(n-X_c)+(m-X_c) \geq m \quad \text{with } X_c \geq 0$$

is valid, we can safely ignore the corresponding matrix cells. An example is given in Figure 1.

Assuming two strings have the same lenghts $n = m$, this optimization reduces by approximately $(\frac{n}{2}+1)(\frac{n}{2}+2)$ the number of cells to be computed, resulting in a substantial computational gain.

2.6 Secure Multi-Party Computation

Secure Multi-Party Computation (SMPC) is a field of cryptography having the goal of enabling a group of independent data owners who do not trust each other or any common third party to jointly compute a function that depends on all of their private inputs. SMPC, introduced by Andrew Yao in the early 1980s (Yao, 1982), can be defined as the problem of n players who want to compute an agreed function of their inputs in a secure way. Formally, we assume $x_1,\ldots,x_n$ inputs, where player i knows x_i, and we want to compute $f(x_1,\ldots,x_n) = (y_1,\ldots,y_n)$ such that player i is guaranteed to learn y_i and nothing more than that.

Adversaries involved in the computation can be categorized into two types: *semi-honest* and *malicious*, according to how willing they are to deviate from the protocol. In (Yao, 1982), Yao introduced the idea of secure computation, in which n parties (or players) want to jointly compute a function $f(x_1, x_2, \ldots, x_m)$ where x_i is the i^{th} party's private input. In the next years he also introduced the Garbled Circuits Protocol that is the basis for many of the most efficient SMPC implementations still today.

At the beginning, secure computation was only a theoretical interest but in the 2000s, algorithmic improvements and computing costs reached a point where it became realistic to think about building practical systems using general-purpose multi-party computation.

The first project that implemented this type of system was Fairplay (Malkhi et al., 2004). For the first time, with Fairplay, it was possible to express a privacy-preserving program in a high level language and compile it to executables that could be run by the parties involved in the computation. However, its scalability and performance limited its use to toy programs.

The speed of the SMPC protocol significantly increased over time and today is several orders of magnitude higher due to a combination of cryptographic, protocol, network and hardware improvements. These improvements made possible the adoption of SMPC implementations in important contexts and applications (Evans and M. Rosulek, 2020). Thanks to the increasingly efficient protocols for SMPC that have been proposed in recent years, SMPC can now be considered as a practical solution to several real-life problems.

2.6.1 Yao's Garbled Circuits Protocol

Yao's Garbled Circuits protocol (GC) is one the most widely known SMPC technique (Yao, 1986) (Micali et al., 1987). Furthermore, Yao's GC runs in constant rounds and avoids the costly latency associated with approaches where the number of communication rounds scales with the circuit depth. The starting point for this and all other protocols is the same: we want to evaluate a given function $F(x, y)$ where party P_1 holds $x \in X$ and P_2 holds $y \in Y$.

First of all we have to convert the function we need into a boolean circuit. Then we have to evaluate each gate securely. To do this, Alice picks two random keys for each wire (inputs and outputs). One key corresponds to 0, the other to 1. There are a total of 6 keys for a 2-input gate. Alice encrypts each row of the truth table by encrypting the output key with the corresponding pair of input keys, then she randomly permutes ("garbles") the encrypted truth table and sends it to Bob. In this way Bob doesn't know which row of garbled table corresponds to which row of original table. Then Alice sends to Bob the corresponding key to her input bit. Since keys are random, Bob won't know what this bit is. The last step consists in running the oblivious transfer protocol between the two keys and Bob's 1-bit input in order to provide Bob with the correct key.

3 SECURE COMPUTATION OF THE EDIT DISTANCE

Edit distance, weighted edit distance and Needleman-Wunsch algorithms are often used and widely adopted in the bioinformatics research field. However, securely computing these metrics is a highly challenging research task. Many research works focused on secure and efficient implementations of edit distance algorithms and the analysis of their benchmarks (Zhu and Huang, 2020) (Zhu and Huang, 2017) (Kaghazgaran, 2017) (Aziz et al., 2017) (Wang et al., 2015) (Huang et al., 2011). However, almost none of these works precisely define and provide all the details used for the experiment setup, such as machine type, chosen framework, implementation and benchmarking settings, source codes, used input data, custom optimizations, etc.

In this work we implement novel secure implementations of existing edit distance algorithms, particularly the Ukkonen's generalized algorithm which has never been implemented before in literature, using the current state-of-the-art garbled circuits in the context of semi-honest model. We will also analyze their performances, and define a clear baseline for future works.

3.1 Experimental Setup

In order to provide meaningful and reproducible results, we decided to use Google Cloud Platform. For this work, the N1, a Compute Engine's first generation general-purpose machine type, was chosen; specifically the `n1-standard-1` instance, provided with 1 vCPU and 3.75 GB of memory and Ubuntu 21.04.

To develop all secure implementations we chose the EMP-toolkit framework (Wang et al., 2016), because it integrates all existing applicable optimizations for garbled circuits including efficient OT extension (Kolesnikov and Kumaresan, 2013) (Ishai et al., 2003), FreeXOR technique (Kolesnikov and Schneider, 2008) and Half-Gates garbling (Zahur et al.,

Table 1: Algorithms performance on different string lengths. Computation inputs are two n-nucleotides genomes. Times are in seconds.

	String Lengths							
	Randomly Generated					Genomic (iDash)		
Algorithm	**200**	**1000**	**2000**	**3000**	**4000**	**3456**	**3465**	**3475**
Wagner-Fischer	0.2	8.1	34.9	-	-	-	-	-
Wagner-Fischer (opt.)	0.2	7.8	34.0	82.6	146.7	109.2	108.6	108.6
Ukkonen (thr.)	0.1	5.0	21.9	52.7	93.7	71.0	70.4	70.9
Ukkonen (gen.)	0.1	4.8	20.9	50.6	90.2	67.4	67.7	68.6

2015). EMP-toolkit also provides a 127-bit computational security (κ). For the purpose of this work a semi-honest model and two-party computation was chosen. All algorithms were developed in C/C++.

3.2 Benchmark Settings

In computer science there is no common consensus regarding benchmark measurement. Many variables can distort the effective result of a process execution, mainly due to I/O operations, task switches, time spent on other processes, interrupt handling, etc. running in the same time span. Generally, two methods are taken into account to measure how much time has passed: Wall time and CPU time.

In this work we decided to mainly adopt the CPU time, using the `clock_gettime` function with the `CLOCK_PROCESS_CPUTIME_ID` option, available in the GNU C Library. However some comparisons have been carried out using also the wall time. To obtain a significant value, each test has been executed 10 times, and the arithmetic mean among all the results has been calculated.

3.3 Experimental Results

For the experimental results, we considered strings of approximately 3000-4000 characters, as done in most of the related works on the secure computation of the exact edit distance of genomic strings (Zhu and Huang, 2020) (Zhu and Huang, 2017) (Kaghazgaran, 2017) (Aziz et al., 2017) (Huang et al., 2011). These lengths are already useful for many types of applications and comparisons on DNA and RNA strings. Of course, the provided code can manage longer strings, comparable to the human DNA lenghts (human chromosomes range in size from about 50 million to 300 million base pairs).

In particular, the proposed algorithms were evaluated using two datasets of DNA data. The first dataset consists of randomly generated strings of various lenghts containing $\{A,C,G,T\}$ elements. The considered lengths were 200, 1000, 2000, 3000, and 4000 elements. The second dataset consists of real DNA strings from a genome database released by "iDASH Security and Privacy Workshop 2016" (Tang et al., 2016). The database includes 50 strings of approximately $3400-3500$ characters each.

Table 1 reports the performance of the four privacy-preserving algorithms that have been implemented: the Wagner-Fischer algorithm, using the entire dynamic programming matrix; the Wagner-Fischer algorithm, optimized to use the minimum needed columns; the Ukkonen's algorithm, considering a threshold of about 60% of the longest string; the Ukkonen's algorithm, using the generalized cut-off technique. The source code is available publicly at GitHub (Migliore, 2021) for interested readers.

The symbol "-" means that the algorithm was terminated by Linux `Out of Memory Killer` process before it ended.

In Table 2 we compared the results obtained in this work with the best prior results found in literature (Zhu and Huang, 2020) (without considering particular and specific customization to the garbling scheme with the EMP-Toolkit), revealing a significant increase in performance with a 36% speedup.

Table 2: Performance comparison. Computation inputs are two 4000-nucleotide genomes.

	CPU time (s)	**Wall time (s)**
Best Prior	N/A	286
This Work	**90.2**	**183**

4 CONCLUSIONS

The computation of the edit distance between human genomes has become a very important task in medical domain. In this paper we have proposed some novel

techniques to securely compute the edit distance on human genomes and proposed an efficient implementation reporting some experimental results on both artificial and public datasets.

Our techniques show improved efficiency over state of the art, reducing the overall time needed and providing a 36% speedup over best prior result found in literature.

Some more optimizations, on both the computation of the distance and the usage of secure computation frameworks can be pursued.

ACKNOWLEDGEMENTS

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

REFERENCES

Aziz, M. M. A., Alhadidi, D., and Mohammed, N. (2017). Secure approximation of edit distance on genomic data. *BMC medical genomics*, 10(2):55–67.

Backurs, A. and Indyk, P. (2014). Edit distance cannot be computed in strongly subquadratic time (unless seth is false).

Evans, David Kolesnikov, V. and M. Rosulek, M. (2020). *A Pragmatic Introduction to Secure Multi-Party Computation*. NOW Publishers.

Evans, W. E. and Relling, M. V. (2004). Moving towards individualized medicine with pharmacogenomics. *Nature*, 429(6990):464–468.

Huang, Y., Evans, D., Katz, J., and Malka, L. (2011). Faster secure {Two-Party} computation using garbled circuits. In *20th USENIX Security Symposium (USENIX Security 11)*.

Hyyrö, H. (2003). Practical methods for approximate string matching.

Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. (2003). Extending oblivious transfers efficiently. In *International Cryptology Conference (CRYPTO)*.

Kaghazgaran, P. (2017). Privacy-preserving edit distance on genomic data. *arXiv preprint arXiv:1711.06234*.

Koboldt, D. C., Fulton, R., McLellan, M., Schmidt, H., Kalicki-Veizer, J., McMichael, J., Fulton, L., Dooling, D., Ding, L., Mardis, E., et al. (2012). Comprehensive molecular portraits of human breast tumours. *Nature*, 490(7418):61–70.

Kolesnikov, V. and Kumaresan, R. (2013). Improved ot extension for transferring short secrets. In *Advances in Cryptology (CRYPTO)*.

Kolesnikov, V. and Schneider, T. (2008). Improved garbled circuit: Free xor gates and applications. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*, ICALP '08, page 486–498, Berlin, Heidelberg. Springer-Verlag.

Levenshtein, V. I. et al. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union.

Malkhi, D., Nisan, N., Pinkas, B., and Sella, Y. (2004). Fairplay—a secure two-party computation system. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, page 20, USA. USENIX Association.

Micali, S., Goldreich, O., and Wigderson, A. (1987). How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, pages 218–229. ACM.

Migliore, A. (2021). Smpc-ed. https://github.com/AndreaMigliore/smpc-ed.

Seiji, F. (2019). Can we optimize the wagner-fischer algorithm? https://ceptord.net/wagner-fischer/index.html.

Tang, H., Wang, X., Kuo, T.-T., Ohno-Machado, L., Jiang, X., Harmanci, A. O., and Kim, M. (2016). Idash privacy & security workshop. http://www.humangenomeprivacy.org/2016/index.html.

Taylor, J. G., Choi, E.-H., Foster, C. B., and Chanock, S. J. (2001). Using genetic variation to study human disease. *Trends in molecular medicine*, 7(11):507–512.

Ukkonen, E. (1985). Algorithms for approximate string matching. *Information and Control*, 64(1):100–118.

Waddell, N., Pajic, M., Patch, A.-M., Chang, D. K., Kassahn, K. S., Bailey, P., Johns, A. L., Miller, D., Nones, K., Quek, K., et al. (2015). Whole genomes redefine the mutational landscape of pancreatic cancer. *Nature*, 518(7540):495–501.

Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173.

Wang, X., Malozemoff, A. J., and Katz, J. (2016). Emp toolkit. https://github.com/emp-toolkit.

Wang, X. S., Huang, Y., Zhao, Y., Tang, H., Wang, X., and Bu, D. (2015). Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 492–503.

Yao, A. C. (1982). Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE.

Yao, A. C. (1986). How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE.

Zahur, S., Rosulek, M., and Evans, D. (2015). Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, pages 220–250.

Zhu, R. and Huang, Y. (2017). Efficient privacy-preserving general edit distance and beyond. *Cryptology ePrint Archive*.

Zhu, R. and Huang, Y. (2020). Efficient and precise secure generalized edit distance and beyond. *IEEE Transactions on Dependable and Secure Computing*.