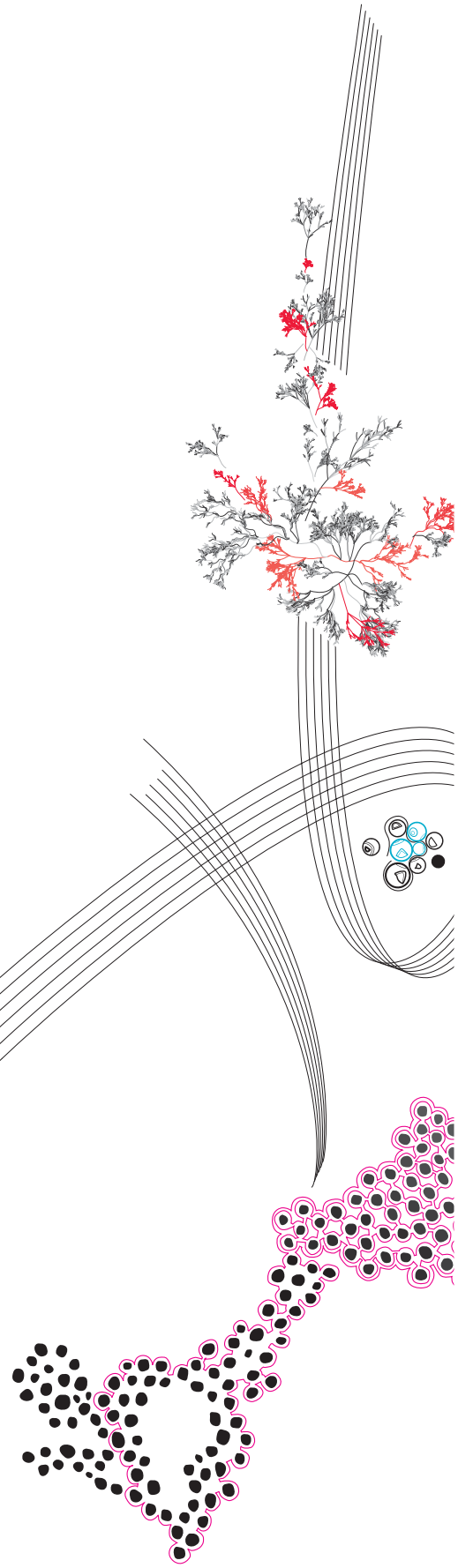

Master Thesis

Sub-Quadratic Privacy-Preserving Cohort Selection

Antoine Gansel

Supervisors: Florian Hahn
Yoep Kortekaas

August, 2023

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Contents

I. Introduction **1**

II. Preliminaries **2**
- i. Notions . . . 2
  - 1) Multi Party Computation (MPC) . . . 2
  - 2) Secret Sharing . . . 2
  - 3) Composition Theorem[26, Section 7.3.1]. . . . 3
  - 4) The Arithmetic Black Box . . . 4
  - 5) Phases of the protocol . . . 5
- ii. Security model . . . 5
- iii. Notations . . . 6

III. Problem Statement **6**
- i. Formal setting definition . . . 7
- ii. Initialising the ABB . . . 8

IV. Solution **9**
- i. Main Protocols . . . 10
- ii. Odd-Even Merge permutation network . . . 16
- iii. Alternative Solutions . . . 18

V. Results **19**
- i. Theoretical analysis . . . 19
- ii. Empirical analysis . . . 22

VI. Potential optimisations **26**
- i. Optimising comparison with constants : Function Secret Sharing[7] . . . 26
- ii. Discarded optimisations . . . 27

VII. Related Work **28**

VIII. Conclusion **29**

IX. Code availability **30**

A. ABB primitives **34**

B. Odd-even Merge: Ideal Functionality by Batcher[4] **39**

C. Intuition Composition theorem **39**

Abstract

In this work, we present a Privacy Preserving Cohort Selection (PPCS) protocol for vertically partitioned data running in sub-quadratic time. Cohort selection is used in case-control studies to match a control group in a distant database, given the knowledge of a test group. Such studies allow one to efficiently put in evidence the effect of a variable (e.g. a medicine) on a situation (e.g. a disease) and are thus of significant importance in the medical field. By providing a tool to easily and efficiently respect the privacy of test subjects in such studies, we aim at mitigating concerns that would naturally arise when processing the data. In this work, we aspire to bridge a gap in the literature that mainly focused on PPCS for horizontally partitioned data until now, as well as to improve on the previous result running in quadratic time. In the following, we formally prove the privacy of our solution and show it achieves a complexity of $\mathcal{O}(n \log_2(n)^2)$. We show that it results in a concrete improvement on the result of previous research considering cohort selections at a European level, and we elaborate on the impact of the bandwidth bottleneck on real case executions of our protocol.

Keywords: Case control, Cohort Matching, Multi-Party Computation, additive secret sharing; sub-quadratic complexity

I. Introduction

Cohort Selection is a tool used in observational studies to limit biases[10, 32] that may have been induced by the initial choice of population. In particular, it is often used in epidemiology to understand the evolution of an epidemic in the population or the effect of a drug on a disease[17, 20]. Given an entity owning a very large database D_1, executing a 1:1 (resp. 1:N) matching cohort selection with respect to a smaller database D_2 means selecting when possible, for every individual in D_2, the most (resp. the N most) similar element in D_1. In a typical example, a research centre has been conducting an observational study on the reaction to a disease given the influence of a new medicine, and now owns some results that need to be put in perspective with people who did not take the drug. In order to verify the usefulness of the treatment, the laboratory will therefore want to obtain data on the behaviour of the disease in a control group of individuals (a matched cohort), that is, a group of individuals with similar features to their initial studied population, but who did not take the treatment [2, 16, 29]. Such a cohort could typically be obtained through a hospital that would possess a large number of records of individuals who got contaminated by the disease. Data involved (whether it originates from the research centre or the hospitals) is, however, extremely sensitive, and as such, sharing their databases for processing would violate the privacy rights of the patients (and the GDPR in European countries). During this research, we came up with a protocol for Privacy-Preserving Cohort Selection (building on the previous research by Kortekaas et al.[23]) in order to mitigate this issue. In particular, we use Multi-Party Computation techniques to compute and keep hidden the intersection between the two data sets (D_1 and D_2), before finding (but keeping it hidden) a match in D_1 for every element of the intersection.

In this work we:

- Present a new protocol for Privacy-Preserving Cohort Selection. We implement it in the Arithmetic Black Box model and prove it private in the semi-honest model such that no party learns more than their own dedicated inputs and outputs of the protocol

- Show that our protocol runs in $\mathcal{O}(n \log_2(n)^2)$ with $n = |D_1| + |D_2|$, efficiently improving on the complexity of the previous solution.
- **Results:** We present a theoretical and empirical run-time analysis of our solution. We put in evidence the runtime that we can realistically expect given some standard settings and present what factors are limiting it, and to what extent. Our solution achieves sub-quadratic complexity and the online phase is executed in 3.3 days for the FULL setting (table 12), representing cohort selection at a national level. Our protocol improves on the run time of the previous solution when executed on 3M records doing a 1:1 matching, or on the FULL setting doing a 1:N matching with $N > 3$.

II. Preliminaries

i. Notions

1) Multi Party Computation (MPC)

MPC denotes the set of schemes allowing a set of parties to perform some computations on a distinct set of data without revealing their personal input to the other involved parties and without learning anything else than their intended output. Additionally, MPC schemes usually want to ensure that the final output will always be correct and received (given that the process terminated), as well as having some guarantees against a dishonest user (called an adversary) that may take part in the computation[24]. The two most important properties to know when it comes to defence against adversaries are:

- The **privacy** property, for which it needs to be proven that an adversary who faithfully follows the protocol won't be able to learn anything else than its view of the protocol's execution and its intended output. Adversaries who faithfully follow the protocol are called *semi-honest* adversaries.
- The **security** property, for which it needs to be proven that, in addition to the privacy property, the adversary is also not able to deviate from the protocol without notifying the other involved parties. Adversaries who may try to deviate from the protocol are usually referred to as *malicious* adversaries.

There are different options when it comes to hiding the data that takes part in the computation. In particular, for our Privacy-Preserving Cohort Selection algorithm, we will use the concept of Secret Sharing.

2) Secret Sharing

Secret sharing is a cryptographic tool used to split a secret between multiple servers in such a way that it can only be retrieved if specific subsets of the involved servers come together. We refer by N-out-of-M secret sharing to any secret sharing protocol where you share data between M servers and need at least a subset of N of those servers to reconstruct the initial secret.

The subsets of parties allowed to retrieve the shared secret are defined through an Access Structure, which is described by a (non-empty) monotone collection of authorised sets of parties $\mathcal{A}$ (only the sets of parties specified in $\mathcal{A}$ can retrieve a secret). We call $\mathcal{A}$ monotone

if $\forall B \in \mathcal{A}, B \subseteq C \implies C \in \mathcal{A}$. The access structure also possesses a distribution scheme $\Sigma = \langle \Pi, \mu \rangle$, where μ can be summarised as a pseudo-random generator over a set $\mathbb{R}$ and Π is a function that given a secret k and a random number $r \in \mathbb{R}$ return (and secretly send) the secret shares to all n parties: $\Pi(k, r) = (s_1, ..., s_n)$. Formally, we have the following definition :

Definition 1. (Access Structure, Distribution Scheme [6]) **-**
Let $\{p_1, ..., p_n\}$ be a set of parties. A collection $\mathcal{A} \subseteq PowerSet(\{p_1, ..., p_n\})$ is monotone if $\mathcal{B} \in \mathcal{A}$ and $\mathcal{B} \subseteq \mathcal{C}$ implies that $\mathcal{C} \in \mathcal{A}$. An access structure is a monotone collection $\mathcal{A} \subseteq 2^{\{p_1,...,p_n\}}$ of non-empty subsets of $\{p_1, ..., p_n\}$. Sets in $\mathcal{A}$ are called authorized, and sets not in $\mathcal{A}$ are called unauthorized.

A distribution scheme $\Sigma = \langle \Pi, \mu \rangle$ with domain of secrets K is a pair, where μ is a probability distribution on some finite set R called the set of random strings, and Π is a mapping from $K \times R$ to a set of n-tuples $K_1 \times ... \times K_n$ where K_j is called the domain of shares of p_j. A dealer distributes a secret $k \in K$ according to the distribution scheme Σ by first sampling a random string $r \in R$ according to μ, computing a vector of shares $\Pi(k, r) = (s_1, ..., s_n)$ and privately communicating each share s_j to party p_j. For a set $\mathcal{A} \subseteq \{p_1, ..., p_n\}$, we denote $\Pi(s, r)_{\mathcal{A}}$ as the restriction of$\Pi(s, r)$ to its $\mathcal{A}$-entries.

Secret Sharing is then defined with respect to a monotone Access Structure $\mathcal{A}$:

Definition 2. (Secret Sharing[6]) **-**
*Let K be a finite set of secrets, where $|K| \geq 2$. A distribution scheme $\Sigma = \langle \Pi, \mu \rangle$ with domain of secrets K is a **Secret Sharing scheme** realizing an access structure $\mathcal{A}$ if the following requirements hold:*

(1) Correctness. — *The secret $k \in K$ can be reconstructed by any authorized set of parties. That is, for any $B \in \mathcal{A}$ (where $B = \{p_{i1}, ..., p_{i|B|}\}$) there exists a reconstruction function $R : K_{i1} \times ... \times K_{i|B|} \to K$ so that for every $k \in K$ (and r a already existing randomly generated string),*

$$Pr[R(\Pi(k, r)_B) = k] = 1$$

(2) Perfect Privacy. — *Every unauthorized set cannot learn anything about the secret (in the information theoretic sense) from their shares. Formally, for any set $T \notin \mathcal{A}$, for every two secret $a, b \in K$ and for every possible vector of shares $\langle s_j \rangle_{p_j} \in T$:*

$$Pr[\Pi(a, r)_T = \langle s_j \rangle_{p_j \in T}] = Pr[\Pi(b, r)_T = \langle s_j \rangle_{p_j \in T}]$$

Note that the set $\mathcal{A}$ in the property **(2) Perfect Privacy** in Definition 2. directly defines the constant N in N-out-of-M Secret sharing. Indeed, if every set of parties in $\mathcal{A}$ contains at least N elements, then you will be able to retrieve a shared secret if and only if you manage to bring at least N of the involved servers together.

3) Composition Theorem[26, Section 7.3.1].

Having a tool such as secret sharing to privately handle data is not sufficient to say that any protocol made with this tool will be private, you first need to prove you are using it properly before being able to say your protocol is private. When using sub-protocols, those proofs can quickly become quite tedious in order to properly show that the privacy of your sub-protocols correctly propagates to your main algorithm.

The composition theorem defined by Goldreich aims at simplifying the proof of privacy of a protocol given the privacy of its sub-protocols. Intuitively, it states that, in the semi-honest model, given a protocol a that uses a protocol b, the privacy of a is dependent on the privacy of b. Formally, it is defined through the following definitions and theorem :

Definition 3. (protocols with oracle access [26]) -
An oracle-aided protocol is an ordinary protocol augmented by pairs of oracle-tapes, one pair per each party, and oracle-call steps defined as follows. Each of the parties may send a special oracle request message, to the other party. Such a message is typically sent after this party writes a string, called its query, on its own write-only oracle-tape. In response, the other party also writes a string, called its query, on its own oracle-tape and responds to the requesting party with an oracle call message. At this point, the oracle is invoked and the result is that a string, not necessarily the same, is written by the oracle on the read-only oracle-tape of each party. This pair of strings is called the oracle answer.

Definition 4. (privacy reductions [26]) -

- *An oracle-aided protocol is said to be using the oracle-functionality f if the oracle answers are according to f. That is, when the oracle is invoked, so that the requesting party writes the query q_1 and the responding party writes the query q_2, the answer-pair is distributed as $f(q_1, q_2)$, where the requesting party gets the first part (i.e., $f_1(q_1, q_2)$).*
  We require that the length of each query be polynomially related to the length of the initial input
- *An oracle-aided protocol using the oracle-functionality f is said to privately compute g if there exist polynomial-time algorithms, denoted S1 and S2, Eq.1 and Eq.2, respectively, where the corresponding views of the execution of the oracle-aided protocol are defined in the natural manner.*
- *An oracle-aided protocol is said to privately reduce g to f if it privately computes g when using the oracle-functionality f. In such a case, we say that g is privately reducible to f.*

Theorem 1 (Composition Theorem for the semi-honest model [26]) -
Suppose that g is privately reducible to f and there exists a protocol for privately computing f. Then there exists a protocol for privately computing g.

However, note that every MPC operation makes for a single sub-protocol. As such, even with the help of the composition theorem, we might still very well end up with tedious and hardly readable proof of privacy, which by their length are also prone to human error.

4) The Arithmetic Black Box

The Arithmetic Black Box (ABB) model, first introduced by Damgård et al[14] and later generalised by Toft[33, Chapter 4], propose a framework to virtually group every private MPC operation under a single protocol with oracle access. It describes an ideal functionality behaving as an honest independent party that trustfully computes a result if every involved party agrees on what needs to be computed and aborts/gets corrupted otherwise. Intuitively, the usefulness of the ABB partly lies in the security proofs, as we can uniquely refer to the ABB when using the Comparison Theorem instead of needing to refer to each MPC operation as independent protocols (Section II.i.3)). Its usefulness also

lies in the generalisation of privacy proofs for any MPC primitives. Taking as example our current work, while we use secret sharing (reason of this choice Section IV.iii.), one could theoretically switch to Homomorphic Encryption without compromising the security proofs by swapping our private Secret Shared operations by private Homomorphic operations (or any other MPC primitives as long as it supports all the operations we need) in the ABB.

The ABB works in three phases. First (1) the **initialisation phase**, where it will generate the correlated randomness necessary for the full protocol execution. This phase is the only one that happens in the offline phase of the protocol (Section II.5)). Then (2) the **loading phase** where all parties input (in rounds) the data necessary to the protocol execution. if all party agrees, the party i inputting its data is expected to send the query $P_i : x \leftarrow s$, and all other party j is expected to send the query $P_j : x \leftarrow ?$. And last (3) the **computation phase** where, at each round, given an operation $\star$ defined in the ABB and two secrets s_1, s_2 previously loaded, each party i queries $P_i : x \leftarrow s_1 \star s_2$. If every parties agree (i.e. sent the same query), the ABB computes the result and outputs it to its respective party.

5) Phases of the protocol

Private protocols, in particular those using secret sharing, need an extensive amount of computational power to generate random elements (i.e. the correlated randomness) that will be used to share the secret values or perform operations. While computing the correlated randomness is heavy, it does not depend on the inputs of the protocol and thus there is no obligation to generate it while executing the program. Protocols are thus usually split into two phases in order to generate correlated randomness during off-peak hours.

Offline phase Also called the pre-processing phase, it computes every data that do not need the knowledge of the parties' inputs to be computed (such as the correlate randomness). Despite its name, this phase can imply communication between servers.

Online phase By opposition to the Off-Line phase, it is during this phase that the two servers actually execute the protocol and communicate with each other. The elements exchanged as well as the computation done during this phase are *dependent on the inputs* of the protocol.

ii. Security model

We assume the existence of an honest but curious adversary $\mathcal{A}$ able to corrupt any one of our two available parties P_1 and P_2. We prove the security of our protocol in the semi-honest model using the simulation paradigm defined by Goldreich[26]. We define an ideal two-party functionality $\mathcal{F}_{PPCS} : (\{0,1\}^*)^2 \rightarrow (\{0,1\}^*)^2$, where the input of P_1 and P_2 are their respective database and the output of each party is the results they expect from the protocol's execution.
We prove that our protocol can be efficiently simulated by a simulator S so that the real and simulated execution views are computationally indistinguishable. In particular, we follow [26, def. 7.2.1]. We define $\Pi = (\Pi_1, \Pi_2)$ as a 2-party protocol computing $\mathcal{F}_{PPCS} = (f_1, f_2)$. Given $\mathbf{x} = (x_1, x_2)$ and $\mathbf{z} = (z_1, z_2)$ so that $\mathcal{F}_{PPCS}(\mathbf{x}) = \mathbf{z}$ we define the respective view and output of each party P_i when executing a protocol as $view_i^{\Pi}(\mathbf{x}) = \{x_i, r, m, ..., m_n\}$ (with $m, ..., m_n$ the messages exchanged during the protocol execution) and $out_i^{\Pi}(\mathbf{x})$. From there, we show :

$$\{(S_1(x_1, f_1(\mathbf{x})), z_1\} \stackrel{c}{\equiv} \{view_1^{\Pi}(\mathbf{x}), out_1^{\Pi}(\mathbf{x})\} \tag{1}$$

$$\{(S_2(x_2, f_2(\mathbf{x})), z_2\} \stackrel{c}{\equiv} \{view_2^{\Pi}(\mathbf{x}), out_2^{\Pi}(\mathbf{x})\} \quad (2)$$

To prove the security of our protocol, we base ourselves on the composition theorem [26, Theorem 7.3.3] (Section II.i.3)), where we reduce our protocol Π_{PPCS} to oracle calls to the two sub-protocols Π_{USG} and Π_{MCS} (Section IV.i.). Additionally, and similarly to the work by Kortekaas et al.[23], we define our private operations in the Arithmetic Black-Box Model (ABB)(Section II.i.4)).

iii. Notations

We introduce a couple of notations we will be using throughout this report. First, we denote a secret share of an element a as $\langle a \rangle$, specifying when necessary $\langle a \rangle^{\mathbb{B}}$ or $\langle a \rangle^{\mathbb{A}}$, depending on whether we shared the binary ($\mathbb{B}$) or arithmetic ($\mathbb{A}$) representation of a. Similarly, given an operation $\star$, we assume $a \star b$ is **the plain-text operation** returning a plain-text result and $\langle a \rangle \star \langle b \rangle$ or $\langle a \rangle \star b$ is the operation **in the ABB** returning a hidden result. We summarise the symbols that can be taken by $\star$ in Table 1.

TABLE 1: Operation and related symbols used along this report. If one of those symbols is used together with one (or more) secret shared elements, it is executed by the ABB.

Symbol	$\wedge$	$*$	$=$	!	$\oplus$	$\vee$	$+$	$-$	$<$
Operation	and	Mult	Equal	Not	Xor	or	Add	Sub	Lower-Than

To limitate redundancy in our pseudocode algorithms, we write $col_{i/i+1} = col_{i/i+1} + col_i$ **instead of writing in two lines** (1) $col_i = col_i + col_i$ (2) $col_{i+1} = col_{i+1} + col_i$. Given an integer $n \geq 1$, we also write $0..n = [0, 1, ..., n-2, n-1]$.

Specifically related to the databases involved during the execution of the protocols, we denote by $\mathbf{db}_i.\langle\mathbf{Field}\rangle$ the access of data **Field** at the i^{th} row of the database. We summarise the values that can be taken by **Field** in Table 2.

TABLE 2: Possible **Field** names in the secret shared database (Section III.i.).

ID	The identifiers of the record as in the initial database
part	Partition label (if from D_1) or 0
y	Special feature (if from D_1) or 0

us	under-study bit. Initially zero, then depends on execution.
imc	in-matched-cohort bit. Initially zero, then depends on execution.

III. Problem Statement

In this work, and similarly to the previous research by Kortekaas et al.[23], we focus on the problem of Privacy-Preserving cohort selection in the context of vertically partitioned data. This implies that our two involved parties, P_1 and P_2, each knows different features for their respective sets of entities. In particular, we consider that P_1 (the hospital in our previous example) stores in its database D_1 the identifiers (such as the Social Security Number) together with some features and a target variable y (in our example, it would be "reaction to the disease") for every individual. We consider that P_2 (the research centre in our example) stores in its database D_2 the minimum amount of information needed for

their case study, which thus reduces for generalisation purposes to the identifier (also the Social Security Number) and a sensitive attribute X (in our example it could be "followed the treatment").

In order to enable the "Privacy-Preserving" component of our work, we use secret sharing to obliviously identify both the intersection between D_1 and D_2 and a set of individuals in D_1 who do not possess that attribute X but are closely related to the records in the intersection. Similarly, the computation needed for the case study is made obliviously thanks to secret sharing, before publishing the final result to the involved parties. Moreover, we assume that both P_1 and P_2 have some interest in executing the cohort selection, and thus will not try to deviate from the protocol. We thus stick to the semi-honest model when proving our protocol to be private.

i. Formal setting definition

The notation and variable names used in this section and throughout this report are (when applicable) the ones used in Kortekaas et al.[23]. Our setup takes into account two parties:

- P_1 owns database D_1 composed of identifiers and a set of features **feat** describing the individual (e.g. age, sex, and other socio-economic or health-related information), as well as a special feature/outcome y. P_1 May be any organisation that collects user-related data. Formally, we write $\mathcal{I}_1 = [1, 2, ..., |D_1|]$ the set of identifiers in D_1, and describe the database as $D_1 = \{(i, \boldsymbol{feat_i}, y_i)\}_{i \in \mathcal{I}_1}$. This database typically represents about 99% of the protocol input.

- P_2 owns the set of identifiers $\mathcal{I}_2$. It also owns a variable of interest X and want to test how much X influences y. P_2 typically represent a research centre. We assume $|\mathcal{I}_2| \ll |\mathcal{I}_1|$ and that if $i \in \mathcal{I}_1$, $j \in \mathcal{I}_2$ are the IDs of a same individual in both databases, then $i = j$ (this **does not** imply $\mathcal{I}_2 \subseteq \mathcal{I}_1$).

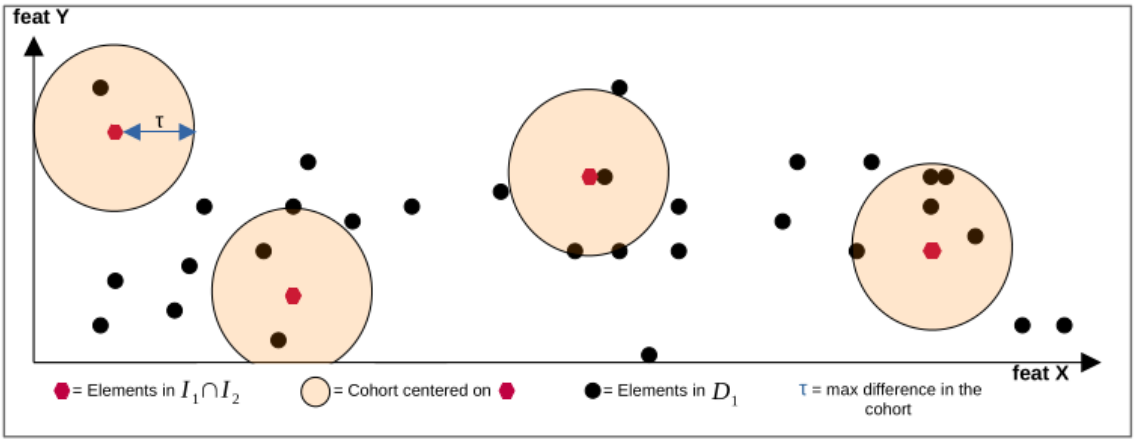

FIGURE 1: Graphical representation of a cohort selection with distance τ defined on two features X and Y. Every black element in an orange zone is suitable for the matched cohort.

Following the idea of cohort selection, P_2 wants to proceed with the creation of cohorts (Figure 1) to compute statistics on the correlation of X on y (for example, the correlation

between the effect of taking medicine X on the "symptoms induced by a specific disease" y) in different groups of people (cohorts).

The creation of potential cohorts (we refer to these "potential cohorts" specifically as partitions) is done with respect to a certain notion of similarity between the vector $\boldsymbol{feat}$ of each individual. Formally, we describe it by the distance comparison function:

$$match_\tau(\boldsymbol{feat_a}, \boldsymbol{feat_b}) = \begin{cases} True, & \text{if } dist(\boldsymbol{feat_a}, \boldsymbol{feat_b}) < \tau \\ False, & \text{otherwise} \end{cases}$$

Using this function, P_1 creates partitions in plain text data so that they do not overlap each other, and every element of the partition is not further than the distance τ of any other element in the partition. We represent these subsets of rows through a variable $part_i, \forall i \in \mathcal{I}_1$ so that,$\forall i, j \in \mathcal{I}_1$, $part_i = part_j \implies match_\tau(\boldsymbol{feat_i}, \boldsymbol{feat_j})$.

We then proceed with matching the cohorts for every element in $\mathcal{I}_2 \cap \mathcal{I}_1$, to which we refer as **under_study** (or **us**) elements. For each of those **us** elements, we want to select up to $N \geq 1$ similar (i.e. in the same partition) elements (1:N Matching). The final selection of elements is referred to as $matchedCohort_\tau$, of which we give the following recursive definition by Kortekaas et al.[23][1] :

Definition 5. ($matchedCohort_\tau$) **-**
$matchedCohort_{\tau,0} := \emptyset$
for $1 < k < |underStudy|$ ***do***
Let I a fixed arbitrary set so that :
(1) $|I| < N$
(2) $\{i \in \mathcal{I}_1 \wedge i \notin underStudy \wedge$
$i \in matchedCohort_{\tau,k-1} \wedge partition_i = partition_{underStudy[k]}\}_{\forall i \in I}$
$matchedCohort_{\tau,k} := matchedCohort_{\tau_k - 1} \cap I$
end for
$matchedCohort_\tau := matchedCohort_{\tau,|underStudy|}$

However, notice that the scope of our work delegates the relevance of matched elements to the pre-processing phase during which P_1's database is partitioned. Indeed, as mentioned in [31] the interest of a cohort selection depends on the similarity between the under-study element and its matched records. In our work, we assume that the distance τ used by P_1 to create the partitions implies that, given an under-study element in a partition A, any other non-under-study element in A is acceptable as a first-level match. Before our protocol's execution, it is thus of P_1's responsibility to define the distance metric and value τ to feat its data.

ii. Initialising the ABB

We define our 2-out-of-2 additive secret sharing operations in the Arithmetic Black Box Model as generalised by Toft[33]. Moreover, we extend the basic ABB model to allow more complex operations (e.g. lower than) and to distinguish between binary (over $\mathbb{Z}_2$) and arithmetic (over $\mathbb{Z}_{2^{64}}$) operations. Intuitively, as some operations are more expensive on binary values than on arithmetic values (e.g. subtraction), this allows us to choose in

[1] modified in (2) as $match_\tau(\boldsymbol{feat_i}, \boldsymbol{feat_j}) \not\implies partition_i = partition_j$ and we want I in the same partition as $underStudy[k]$

which format to share our data depending on which operations we want to execute on it. In practice, every field of the database is shared as binary values except the field "y".

Most of the operations we define in the Arithmetic Black Box come from the previous solution by Kortekaas et al.[23]. In fact, for this work we only needed to add the Compare-Switch operation as suggested by Abspoel et al.[1]. Refer to table 3 for a list of the operations we use and to Appendix.A. for details concerning their implementation and intuition about their proof of privacy.

In the following, when describing the loading of the data in the ABB we abstract away the call $P_j : x \leftarrow ?$ from the party receiving the share. Moreover, when an operation involves secret shared data we assume that it uses the ABB by default.

TABLE 3: Summary of operation defined in the ABB and their origin.

Operation(s)	Taken from	Originates from
AND, Multiplication	Kortekaas et al.[23]	Beaver et al.[5]
Equal	Kortekaas et al.[23]	Kolesnikov et al.[22]
Load, Reveal, Not, XOR, OR, Add, Sub, Lower-than	Kortekaas et al.[23]	unknown
Compare-Switch	Abspoel et al.[1]	unknown

IV. Solution

To achieve sub-quadratic complexity, we identify the under-study elements and then the elements to match in the cohort using collisions. Intuitively, this works for the under-study Group identification (USG) because we assume the IDs to be coherent across both databases. Similarly, it works for the Matched Cohort Selection (MCS) because the partitions have been created by P_1 in a pre-processing phase (Section III.i.). In both cases, we obliviously sort the database (which can be done in $\mathcal{O}(n \log_2(n)^2)$) on the field of interest (resp. ID and $part$) and then go linearly through the sorted database to compute an equality circuit on every neighbouring element. Notice that it results in a total complexity in $\mathcal{O}(n \log_2(n)^2)$, thereby satisfying our objective of sub-quadratic complexity.

We argue that such an idea indeed allows us to perform a cohort selection. It works first to identify the under-study elements: after sorting on ID, if an individual with ID i is present in the databases from P_1 and P_0, we will find two records next to each other with ID i. We can thus detect such cases and set every such individual as being under-study. It then works in a similar fashion to select the matched cohort: after sorting on $part$, given any under-study element with partition j its potential matches (individual with the same partition label) will be in his M closest neighbours on the right and M closest neighbours on the left with M the biggest possible cohort size.

As mentioned in Section II.ii., we use the ABB model coupled with the composition theorem by Goldreich to prove our protocol to be private. Intuitively, given the organisation of our protocol (Figure 2) the privacy property should propagate through the following structure:

(1) We implement the ABB privately and initialise it correctly according to [33]. (2) Π_{USG} and Π_{MCS} subprotocols are made to have an access pattern independent of the actual data, and every operation on the data is delegated to the ABB. (3) Π_{PPCS} only but coordinates the initialisation of the ABB and the execution of Π_{USG} and Π_{MCS}. It does not execute anything else than the initialisation round of the ABB and some calls to the two subprotocols.

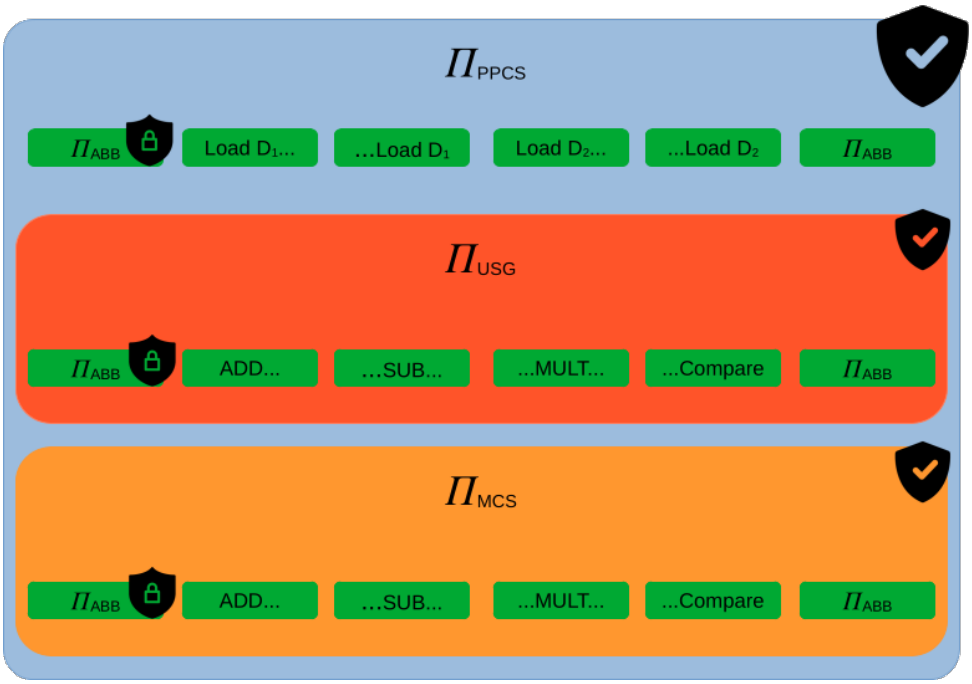

FIGURE 2: Organisation of our PPCS protocol and propagation of the privacy proof.

i. Main Protocols

Ideal Functionality 1. $\mathcal{F}_{USG}$: Under Study Group identification.

Inputs:
— Secret shared database $db = (id, feat, cost, us, imc)$ containing every element from the secret shares of both the private database of P_1 (D_1) and the set of identifiers of P_2 (D_2) padding the empty fields with 0.

Output:
— Secret shared database db containing every element from both party's private database, where every line whose id field accepts a duplicate has their us field set to 1.

Trusted party $\mathcal{T}_1$ executes the following:

1. $\mathcal{T}_1$ Obtains the database from the ABB
2. $\mathcal{T}_1$ Computes for every $1 < j < |D_1| + |\mathcal{I}_2|$:
   $db_j.us = db_j \in D_1 \wedge db_j.id \in \mathcal{I}_2$.

Out. $\mathcal{T}_1$ output the updated database to the ABB

Under-Study Group Identification Protocol Π_{USG} (Protocol 1.) is a 2-party protocol implementation of our ideal functionality $\mathcal{F}_{USG}$ (Ideal Functionality 1.). This protocol is responsible for identifying the under-study elements in D_1 (i.e. the intersection between both databases). It works by using the odd-even merge sort [3, 21] together with the compare-switch operation as suggested by Abspoel et al.[1, 3.1]) to order the secret shared database $db = \{(id, part, cost, us, imc)\}^*$ on the id. It then computes an equality circuit on the identifiers of every direct neighbour in the database, efficiently detecting any duplicated ID (and thus under-study element). The protocol uses the result to generate a fresh share of the under-study bit of both elements depending on the (secret shared) comparison's result and the current value of the under-study bit.

Protocol 1. Π_{USG} : Under Study Group identification

Inputs: Database structure **db** containing at least:
— all binary identifiers from P_1 and P_2 : $\forall i \in 0..|\mathcal{I}_1| + |\mathcal{I}_2|,\ \mathbf{db}_i.\langle id\rangle^{\mathbb{B}} \neq 0$
— Private bit for every element: $\forall i \in 0..|\mathcal{I}_1| + |\mathcal{I}_2|,\ \mathbf{db}_i.\langle us\rangle^{\mathbb{B}} = \langle 0\rangle^{\mathbb{B}}$
Output:
— Updated private bits for all $i \in 0..|\mathcal{I}_1| + |\mathcal{I}_2|$ indicating whether there exists $j \in 0..|\mathcal{I}_1| + |\mathcal{I}_2|$ so that $\mathbf{db}_i.id = \mathbf{db}_j.id \implies \mathbf{db}_i.us = 1$

The Protocol:

1: rec_sort_id(**db**, 0, $|\mathcal{I}_1| + |\mathcal{I}_2|$)
2: **for** i in $0..|db|$ **do**
3: $\quad \mathbf{db}_{i/i+1}.\langle us\rangle^{\mathbb{B}} \leftarrow \mathbf{db}_{i/i+1}.\langle us\rangle^{\mathbb{B}} \oplus \mathbf{db}_i.\langle id\rangle^{\mathbb{B}} = \mathbf{db}_{i+1}.\langle id\rangle^{\mathbb{B}}$
4: **end for**

rec_sort_id:
Inputs:
— database **db** to sort (as specified above)
— starting index *start*
— length n fo the sub-list
Output: sorted database

1: **if** $n > 1$ **then**
2: $\quad n\_l \leftarrow \lceil n/2 \rceil$
3: $\quad n\_r \leftarrow \lfloor n/2 \rfloor$
4: $\quad$ rec_sort(**db**, *start*, n_l)
5: $\quad$ rec_sort(**db**, *start*, n_r)
6: $\quad \Pi_{merge}$(**db**, n_l, n_r, 1) *//Section IV.ii., with compare-switch on id*
7: **end if**

Lemma 1 (Correctness of Π_{USG}) **-**
Protocol Π_{USG} (Protocol 1.) is a correct 2-party computation protocol for computing $\mathcal{F}_{USG}$ (Ideal Functionality 1.).

Proof. In the first line of the protocol, Π_{USG} executes the odd-even merge sort protocol, whose correctness depends on the correctness of our odd-even merge implementation (Lemma 5). It follows that from line 2., our shared database *db* is correctly ordered on the IDs of the records.
Then for any line $1 < i < |D_1| + |\mathcal{I}_2|$, $db_i \in D_1$ if and only if $db_i.part \neq 0$ (as every element of D_1 has a partition value greater than 0), and moreover $db_i.id \in \mathcal{I}_2$ if

$\exists j \in [1; |D_1| + |\mathcal{I}_2|] \cap [i-1, i+1]$ so that $db_j.id = db_i.id$ (by correctness of our sorting step). It follows that the "for" loop lines 2 to 4 correctly set the us field to 1 (= true) for any element originating from D_1 and whose ID also exists in $\mathcal{I}_2$, and 0 otherwise.
Therefore, Π_{USG} correctly implements $\mathcal{F}_{USG}$. □

Lemma 2 (Privacy of Π_{USG}) **-**
The protocol Π_{USG} (Protocol 1.) is private in the security model defined in III.ii.

Proof. A party's execution view of our protocol can be summarised to $(in, r, abb_1, ..., abb_x)$, with in the party input, r its random tape and $abb_1, ..., abb_x$ all the messages exchanged with the oracle of the Arithmetic Black Box. We note that it is indeed the case as every element that is not involved with the ABB is independent of the user input, and thus trivially simulatable. Following Goldreich's composition theorem, it follows that the view generated by Π_{USG} can be efficiently simulated by oracle calls to $\mathcal{F}_{ABB}$. As such, Π_{USG} is private if and only if the ABB is private too.
Therefore, as the ABB is proven private, so is Π_{USG}. □

Ideal Functionality 2. $\mathcal{F}_{MCS}$: Matched Cohort Selection

---

Inputs:
— Secret shared database db containing every element from both party's private database, where every line originally from D_1 whose id field accepts a duplicate in D_2 has their us field set to 1.
— N the number of elements to match for every under-study elements
— M the biggest partition size
Outputs:
— Secret shared database db containing every element from both party's private database, where the field imc has been set with respect to the fields us and $feat$ as specified in definition 5.

Trusted party $\mathcal{T}_2$ executes the following:

1. $\mathcal{T}_2$ obtains from the ABB the database with the under-study bit set correctly.
2. $\mathcal{T}_2$ Computes for every $1 < j < |D_1| + |\mathcal{I}_2|$ with $db_j \in D_1$:
   $\mathcal{T}_2$ executes, for $i \in [j \pm M]$: $db_i.mcs \leftarrow (db_j.feat = db_i.feat) \wedge (db_i.us = db_i.mcs = 0) \wedge (\text{already-matched} < N)$.

Out. $\mathcal{T}_2$ outputs the updated database to the ABB.

Matching Cohort Identification Protocol Π_{MCS} (Protocol 2.) is a 2-party protocol implementation of our ideal functionality $\mathcal{F}_{MCS}$ (Ideal Functionality 2.). This protocol is responsible for, given the knowledge of every under-study element, selecting the matched cohort following the Definition 5.. It works by first ordering (again through the odd-even merge sort) the big secret shared database $db = \{(id, part, cost, us, imc)\}^*$ on its partition label. It then computes an equality circuit on the partition of every two lines i, j in the newly ordered database with $|i - j| \leq M$ (M the maximal partition size). Using the secret shared result of the comparison, a new share is generated for $db_j.imc$ so that it respects the definition 5.

Lemma 3 (Correctness of Π_{MCS}) **-**
Protocol Π_{MCS} (Protocol 2.) is a correct 2-party computation protocol for computing $\mathcal{F}_{MCS}$ (Ideal Functionality 2.).

Protocol 2. Π_{MCS} : Matched Cohort Selection

Inputs: Database structure **db** containing at least:
— all binary identifiers from P_1 and P_2 : $\forall i \in\ 0..|\mathcal{I}_1| + |\mathcal{I}_2|,\ \mathbf{db}_i.\langle id\rangle^{\mathbb{B}} \neq 0$
— Private binary label for every element: $\mathbf{db}_i.\langle part\rangle^{\mathbb{B}}$
— Private bit for every element: $\mathbf{db}_i.\langle us\rangle^{\mathbb{B}}$
— Private bit for every element: $\forall i \in\ 0..|\mathcal{I}_1| + |\mathcal{I}_2|,\ \mathbf{db}_i.\langle imc\rangle^{\mathbb{B}} = \langle 0\rangle^{\mathbb{B}}$
— Maximal partition size M (Public)
Output:
— Updated private bits for all $i \in [0..|\mathcal{I}_1| + |\mathcal{I}_2|]$ indicating whether i is in the matched cohort: $\mathbf{db}_i.\langle \mathbf{in\_mathced\_cohort}\rangle^{\mathbb{B}}$

The Protocol:

1: rec_sort_part(db, 0, $|\mathcal{I}_1| + |\mathcal{I}_2|$)
2: **for** i in $|\mathcal{I}_2|...|\mathcal{I}_1| + |\mathcal{I}_2|$ **do**
3: **for** j in $1..M$ **do**
4: $\mathbf{db}_{i-j}.\langle imc\rangle^{\mathbb{B}} \leftarrow (\mathbf{db}_{i-j}.\langle imc\rangle^{\mathbb{B}} \oplus \mathbf{db}_i\langle.us\rangle^{\mathbb{B}}) \wedge (\mathbf{db}_{i-j}.\langle part\rangle^{\mathbb{B}} = \mathbf{db}_i.\langle part\rangle^{\mathbb{B}}) \wedge (!\mathbf{db}_{i-j}.\langle us\rangle^{\mathbb{B}}) \wedge (\langle matched\rangle^{\mathbb{B}} < \langle N\rangle^{\mathbb{B}})$
5: $\langle matched\rangle^{\mathbb{B}} \leftarrow \langle matched\rangle^{\mathbb{B}} + \mathbf{db}_{i-j}.\langle imc\rangle^{\mathbb{B}}$
6: $\mathbf{db}_{i+j}.\langle imc\rangle^{\mathbb{B}} \leftarrow (\mathbf{db}_{i+j}.\langle imc\rangle^{\mathbb{B}} \oplus \mathbf{db}_i.\langle us\rangle^{\mathbb{B}}) \wedge (\mathbf{db}_{i+j}.\langle part\rangle^{\mathbb{B}} = \mathbf{db}_i.\langle part\rangle^{\mathbb{B}}) \wedge (!\mathbf{db}_{i+j}.\langle us\rangle^{\mathbb{B}}) \wedge (\langle matched\rangle^{\mathbb{B}} < \langle N\rangle^{\mathbb{B}})$
7: $\langle matched\rangle^{\mathbb{B}} \leftarrow \langle matched\rangle^{\mathbb{B}} + \mathbf{db}_{i+j}.\langle imc\rangle^{\mathbb{B}}$
8: **end for**
9: **end for**

rec_ sort_ part:
Inputs:
— database **db** to sort (as specified above)
— starting index *start*
— length n fo the sub-list
Output: sorted database

1: **if** $n > 1$ **then**
2: $n\_l \leftarrow \lceil n/2 \rceil$
3: $n\_r \leftarrow \lfloor n/2 \rfloor$
4: rec_sort(**db**, *start*, n_l)
5: rec_sort(**db**, *start*, n_r)
6: Π_{merge}(**db**, n_l, n_r, 1) *//Section IV.ii., with compare-switch on part*
7: **end if**

Proof. In the first line of the protocol, Π_{USG} executes the odd-even merge sort protocol, whose correctness depends on the correctness of our odd-even merge implementation (Lemma 3). It follows that from line 2., our shared database db is correctly ordered on the partition label. Then, setting M the largest partition size, for any $1 < i < |D_1| + |\mathcal{I}_2|$ the elements j potentially satisfying $db_j.part = db_i.part$ are in $[1; |D_1|+|\mathcal{I}_2|] \cap [i \pm max\_cohort]$ Π_{MCS} indeed access all $j \in [1; |D_1| + |\mathcal{I}_2|] \cap [i \pm max\_cohort]$ for every $1 < i < |D_1| + |\mathcal{I}_2|$, keeping a counter of the number of matched elements. When setting the imc bit, we argue that the range of the loop is correct because of the way we initialise the secret shared database (Protocol 3.). Indeed, the partition value of every element in P_2 is zero while the elements from P_1 have values greater than zero. It follows that after sorting, all the

elements from P_1 have indices in $|\mathcal{I}_2|...|\mathcal{I}_1| + |\mathcal{I}_2|$. Then, with "$\mathbf{db}_{i\pm j}.part = \mathbf{db}_i.part$" we indeed only set it to 1 if and only if both elements are in the same partition; with "$\wedge$ $matched < N$" we indeed only set it to 1 if and only if we matched less than N elements; and with "$\wedge$ $\mathbf{db}_i$.**us**" we indeed only set it to 1 if and only if db_i is under-study. Therefore, Π_{MCS} correctly implements $\mathcal{F}_{MCS}$. □

Lemma 4 (Privacy of Π_{MCS}) **-**
The protocol Π_{MCS} (Protocol 2.) is private in the security model defined in III.ii..

Proof. The proof for Lemma 4 is analogue to the proof for Lemma 2. As such, privacy of Π_{MCS} follows by privacy of the ABB □

Ideal Functionality 3. $\mathcal{F}_{PPCS}$: Privacy Preserving Cohort Selection

---

Inputs:
— Private database of entities of P_1: D_1 of length X
— Private set of identifiers of P_2: $\mathcal{I}_2$ of length Y
— Public integer $N \geq 1$ describing how many entities we match for every entity P_1 and P_2 share
Output:
— Private structure associating for every elements of P_1: (1) a binary value in the ABB indicating if it is shared with P_2 and (2) a binary value in the ABB indicating if it is in the matched cohort.

Trusted party $\mathcal{T}$ executes the following:

1. $\mathcal{T}$ Receives the private database D_1 and private set of identifier $\mathcal{I}_2$ in a single big database $db = (id, feat, cost, us, imc)$, padding for every missing field in a database with zeros. It then inputs it to the ABB together with the received N.
2. $\mathcal{T}$ instructs the ABB to compute the functionality $\mathcal{F}_{USG}$.
3. $\mathcal{T}$ instructs the ABB to compute the functionality $\mathcal{F}_{MCS}$.

Out. $\mathcal{T}$ outputs the result of step 3.

Cohort Selection Protocol Π_{PPCS} (Protocol 3.) is a 2-party protocol implementation of our ideal functionality $\mathcal{F}_{PPCS}$ (Ideal Functionality 3.). Its role is to implement a Privacy-Preserving Cohort Selection following our settings in Section III.i.. It starts by executing the initialisation round of the ABB, instructing P_1 (and then P_2) to secret share its database into the first available slots of an (initially empty) database of size $|\mathcal{I}_1| + |\mathcal{I}_2|$. It then delegates the rest of the ideal functionality's implementation: (1) to Π_{USG} (Protocol 1.) that will detect which elements in D_1 are under-study, (2) to Π_{MCS} (Protocol 2.) that, given the under-study elements will decide which elements need to be put in the matched cohort.

Theorem 2 (Correctness of Π_{PPCS}) **-**
Given that both protocols Π_{USG} and Π_{MCS} are correct 2-party computation protocol for computing functionalities $\mathcal{F}_{USG}$ and $\mathcal{F}_{MCS}$ respectively, Π_{PPCS} is a correct 2-party computation protocol for computing functionality $\mathcal{F}_{PPCS}$ (Ideal Functionality 3.).

Proof. During the **first** step of Π_{PPCS}, both parties input their private data in a single big database $db = (id, part, cost, us, imc)$ in the ABB using the Load functionality, padding

Protocol 3. Π_{PPCS} : Full Privacy Preserving Cohort Selection algorithm

Inputs:

— Private database of entities of P_1: D_1 of length X

— Private set of identifiers of P_2: $\mathcal{I}_2$ of length Y

— Public integer $N \geq 1$ describing how many entities we match for every entity P_1 and P_2 share

Output:

— Private structure associating for every elements of P_1: (1) a binary value in the ABB indicating if it is shared with P_2 and (2) a binary value in the ABB indicating if it is in the matched cohort.

The Protocol:

1. **Setup.**
   (a) P_1 Computes on plaintext D_1 a partition label $partition_i = 0, \forall i \in \mathcal{I}_1$ such that: $partition_i = partition_j \implies match_\tau(\mathbf{feat}_i, \mathbf{feat}_j) \forall j \in \mathcal{I}_1$
   (b) P_1 runs for all $(i, partition_i, y_i) \in D_1$:
   — Load $P_1 : \mathbf{db}_i.\langle id \rangle (= \langle i \rangle^{\mathbb{B}}) \leftarrow \text{BinRepr}(i)$
   — Load $P_1 : \mathbf{db}_i.\langle part \rangle (= \langle partition_i \rangle^{\mathbb{B}}) \leftarrow \text{BinRepr}(partition_i)$
   — Load $P_1 : \mathbf{db}_i.\langle y \rangle (= \langle y_i \rangle^{\mathbb{A}}) \leftarrow y_i$
   — Load $P_1 : \mathbf{db}_i.\langle us \rangle (= \langle 0 \rangle^{\mathbb{B}}) \leftarrow \text{BinRepr}(0)$
   — Load $P_1 : \mathbf{db}_i.\langle imc \rangle (= \langle 0 \rangle^{\mathbb{B}}) \leftarrow \text{BinRepr}(0)$
   (c) P_2 runs for all $j \in \mathcal{I}_2$:
   — Load $P_2 : \mathbf{db}_{X+j}.\langle id \rangle (= \langle j \rangle^{\mathbb{B}}) \leftarrow \text{BinRepr}(j)$
   — Load $P_2 : \mathbf{db}_{X+j}.\langle part \rangle (= \langle 0 \rangle^{\mathbb{B}}) \leftarrow \text{BinRepr}(0)$
   — Load $P_2 : \mathbf{db}_{X+j}.\langle y \rangle (= \langle 0 \rangle^{\mathbb{A}}) \leftarrow 0$
   — Load $P_2 : \mathbf{db}_{X+j}.\langle us \rangle (= \langle 0 \rangle^{\mathbb{B}}) \leftarrow \text{BinRepr}(0)$
   — Load $P_2 : \mathbf{db}_{X+j}.\langle imc \rangle (= \langle 0 \rangle^{\mathbb{B}}) \leftarrow \text{BinRepr}(0)$
2. **(Π_{USG})Under Study Group Identification.**
   // *Orders on $\langle id \rangle$ with the odd-even merge sort (Section III.ii.).*
   // *for all elements in the sorted database, P_1, P_2 instruct ABB to compute :* $\mathbf{db}_{i/i+1}.\langle us \rangle \leftarrow \mathbf{db}_{i/i+1}.\langle us \rangle \vee \mathbf{db}_{i+1}.\langle id \rangle = \mathbf{db}_i.\langle id \rangle$
3. **(Π_{MCS})Matched Cohort Selection**
   // *Orders on Partitions with the odd-even merge sort (Section III.ii.).*
   // *For every element i in the database, tries to set the M elements on the right and on the left of i (M = max cohort size) in the matched cohort according to N and definition 5.*
   $\mathbf{db}_j.\langle imc \rangle \leftarrow \mathbf{db}_j.\langle \mathbf{in\_matching\_cohort} \rangle \vee \ (\mathbf{db}_i.\langle us \rangle$
   $\wedge \ (\mathbf{db}_j.\langle part \rangle = \mathbf{db}_i.\langle part \rangle) \ \wedge (\text{already matched} < N) \ \wedge \ !\mathbf{db}_j.\langle us \rangle)$

the fields that were not existing in their private database with zeros. This is indeed a correct implementation of the first step of $\mathcal{F}_{PPCS}$. We emphasise that it is here assumed that P_1 created partitions on its plaintext data using a carefully defined distance τ.
In the **second** step, Π_{PPCS} queries Π_{USG} to compute $\mathcal{F}_{USG}$. It follows that the second step is correctly implemented if $\mathcal{F}_{USG}$ is correctly implemented by Π_{USG}, and the ABB

now "knows" which elements are under-study.
In the **third** step, Π_{PPCS} queries Π_{MCS} to compute $\mathcal{F}_{MCS}$. It follows that the third step is correctly implemented if $\mathcal{F}_{MCS}$ is correctly implemented by Π_{MCS}, and the ABB now also "knows" the matched cohort that needs to be returned to P_0.
Therefore, Π_{PPCS} correctly implements $\mathcal{F}_{PPCS}$ if and only if Π_{USG} and Π_{MCS} correctly implement $\mathcal{F}_{USG}$ and $\mathcal{MCS}$ respectively. Thus, we can state that Π_{PPCS} is a correct implementation of $\mathcal{F}_{PPCS}$ by correctness of Lemma 1 and Lemma 3. □

Theorem 3 (Privacy of Π_{PPCS}) **-**
The protocol Π_{PPCS} is private in the security model defined in III.ii., given that Π_{USG} and Π_{MCS} are.

Proof. A party's execution view of Π_{PPCS} can be summarised as $(in, r, abb_1, ..., abb_x, usg_1, ..., usg_y, mcs_1, ..., mcs_z)$, with in the party input, r the random tape, $abb_1, ..., abb_x$ the messages exchanged with an ABB oracle when sharing the elements, $usg_1, ..., usg_y$ the messages exchanged with the oracle answering according to $\mathcal{F}_{USG}$ and $mcs_1, ..., mcs_z$ the messages exchanged with the oracle answering according to $\mathcal{F}_{MCS}$. Following Goldreich's composition theorem[26, Theorem 7.3.3], Π_{PPCS} is privately reducible to Π_{ABB}, Π_{USG} and Π_{MCS}, and it thus follows that Π_{PPCS} is private if all three protocols are. As we assume the inputs to Π_{ABB} to be private, Π_{USG} is private by the correctness of Lemma 2 and Lemma 4. □

ii. Odd-Even Merge permutation network

While Batcher [4] speaks about the odd-even merge permutation network with two lists of different and arbitrary sizes, in the literature we only found algorithms that exclusively accept lists of equal length and in power of 2. In order for us to execute the odd-even merge sort algorithm on any list of arbitrary length we, therefore, created an odd-even merge protocol accepting any list composed of two halves A, B of respective size $p \geq q$, differing of at most 1. The only true difference of our protocol with respect to already available ones is that we do not step from one half to the other using the recursive step value. Indeed, while this value remains correct inside one half, it does not describe the distance between the two halves correctly if the initial list size was not of the form 2^n. We solve this issue by keeping the start index of both halves across the recursion and using those indexes when stepping over to the second half.
In Protocol 4., by 'compare_switch' we refer to the operation we detailed in Appendix A.xii.. Notice however that the operation detailed in Appendix is but a framework to adapt to the elements to swap. In our case, the swapping component needs to be executed on every field.

Lemma 5 (Correctness of Π_{merge}) **-**
The protocol Π_{merge} (Protocol 4.) correctly implements the ideal functionality specified by Batcher (Appendix B.) for any two lists A, B of respective size $p \geq q$, with $p - q \leq 1$.

Proof. Following the proof sketch by Batcher[4, Appendix A] given $[a_0, a_1, ...]$ and $[b_0, b_1, ...]$ the two ordered input lists, $[c_0, c_1, ...]$ their ordered merge (i.e. the result), $[d_0, d_1, ...]$ the ordered merge of their odd-indexed terms and $[e_0, e_1, ...]$ the ordered merge of their even-indexed terms. We have :

$$\begin{aligned} c_0 &= e_0 \\ c_{2i} &= max(e_i, d_{i-1}), \forall i \geq 1 \\ c_{2i-1} &= min(e_i, d_{i-1}), \forall i \geq 1 \end{aligned} \tag{3}$$

Protocol 4. Π_{merge} : Recursive odd-even merge for arbitrary length

Inputs:

— a vector Vec of size n with the sub-vectors $Vec[0..\lceil n/2 \rceil]$ and $Vec[\lceil n/2 \rceil..n]$ are ordered

— $n\_l = \lceil n/2 \rceil$ and $n\_r = n - n\_l$ the respective size of the left and right sub-vectors.

— s_l and s_r the respective starting position od the left and right sub-vectors

— $step$ the number of elements between each neighbouring elements in the sub-vectors

Output:

— the vector Vec fully ordered

The Protocol:

1: **if** $n\_l > 1$ **then**
2: $n\_even\_l \leftarrow \lceil n\_l/2 \rceil$
3: $n\_even\_r \leftarrow \lceil n\_r/2 \rceil$
4: $n\_odd\_l \leftarrow \lfloor n\_l/2 \rfloor$
5: $n\_odd\_r \leftarrow \lfloor n\_r/2 \rfloor$
6: $ns \leftarrow step * 2$
7: $\Pi_{merge}(Vec, n, n\_even\_l, n\_even\_r, s\_l, s\_r, ns)$
8: $\Pi_{merge}(Vec, n, n\_odd\_l, n\_odd\_r, s\_l + step, s\_r + step, ns)$
9: $i, j \leftarrow 0, 0$
10: **while** $i + 1 \leq (n\_r + n\_l - 1)/2$ **do**
11: **if** $(i * ns) + step >= n_l * step$ **then**
12: compare_switch(Vec, $s\_r + (j * step)$, $s\_r + (j * step) + step$)
13: $j \leftarrow j + 2$
14: **else if** $(i * ns) + 2 * step >= n\_l * step$ **then**
15: compare_switch(Vec, $s\_l + (i * ns) + step$, s_r)
16: $j \leftarrow 1$
17: **else**
18: compare_switch(Vec, $s\_l + (i * ns) + step$, $s\_l + (i * ns) + 2 * step$)
19: **end if**
20: $i \leftarrow i + 1$
21: **end while**
22: **else if** $(n\_l = 1) \wedge (n_r = 1)$ **then**
23: compare_switch(Vec, s_l, s_r)
24: **end if**

We prove the correctness of our protocol by induction. Assume we are in the $\mathbf{n-1}^{th}$ recursive call. For this recursive step, the input lists A and B for the even-indexed recursion are respectively $[a_{0*2^{\mathbf{n-1}}}, a_{1*2^{\mathbf{n-1}}}, ..., a_{x*2^{\mathbf{n-1}}}]$ and $[b_{0*2^{\mathbf{n-1}}}, b_{1*2^{\mathbf{n-1}}}, ..., b_{y*2^{\mathbf{n-1}}}]$, of size x and y with $x * 2^{\mathbf{n-1}} \leq p$ and $y * 2^{\mathbf{n-1}} \leq q$. We only prove the even-indexed recursion as both odd and even-indexed recursion are symmetric. After calling the recursive merge on each odd and even element, we have (with $\|$ the concatenation) :

$$\begin{aligned} [e_0, e_1, ...] &= \mathbf{ORDER}([a_{0*2^{\mathbf{n}}}, a_{1*2^{\mathbf{n}}}, ...] \;||\; [b_{0*2^{\mathbf{n}}}, b_{1*2^{\mathbf{n}}}, ...]) \\ [d_0, d_1, ...] &= \mathbf{ORDER}([a_{(0*2^{\mathbf{n}})+1}, a_{(1*2^{\mathbf{n}})+1}, ...] \;||\; [b_{(0*2^{\mathbf{n}})+1}, b_{(1*2^{\mathbf{n}})+1}, ...]) \end{aligned} \tag{4}$$

Then, during the execution, the elements of our lists have been swapped as follows :

1. **If x is even,** we have $[a'_{0*2^{\mathbf{n-1}}}, a'_{1*2^{\mathbf{n-1}}}, ..., a'_{x*2^{\mathbf{n-1}}}] = [e_0, d_0, ..., e_{x/2-1}, d_{x/2-1}]$ and $[b'_{0*2^n}, b'_{1*2^n}, ...] = [e_{x/2}, d_{x/2}, ...]$

2. **If x is odd,** then $[a'_{0*2^{n-1}}, a'_{1*2^{n-1}}, ..., a'_{x*2^{n-1}}] = [e_0, d_0, ..., e_{\lfloor x/2 \rfloor}]$ and $[b'_{0*2^n}, b'_{1*2^n}, ...] = [d_{\lfloor x/2 \rfloor}, e_{\lfloor x/2 \rfloor+1}, ...]$

Following Eq. 3, for case 1. and 2. we want to obtain respectively:

1. $[c_0, c_1, ...] = [a'_0, min(a'_1, a'_2), ..., min(a'_{x-1}, b'_0), max(a'_{x-1}, b'_0), min(b'_1, b'_2), ...]$
2. $[c_0, c_1, ...] = [a'_0, min(a'_1, a'_2), ..., max(a'_{x-1}, a'_{x-2}), min(b'_0, b'_1), ...]$

In both cases, we continue alternating the compare swap until we can't anymore. If there remains an element, (i.e. the size $x + y$ of $A\|B$ is even) then the last element is b'_{y-1}. In such a case, b'_{y-1} either (1) the largest element of list d if x and y are odd or (2) the largest element of list e if x and y are even. Because of the initial ordering of both A and B, this implies that b'_{y-1} is indeed the largest element of $A\|B$.
In other words, we need to start by compare-switching $a_{1*2^{n-1}}$ and $a_{2*2^{n-1}}$ and continue two by two until we access the last elements of list a (managed "IF..Then..Else" instructions lines 17-18), then (1) if the last element of a is an "even" element we need to compare the last element of a with the first element of b (managed by the "IF..Then..Else" instruction over lines 14-16) and then compare-switch the elements of b two by two from b_{1*2^n} (managed by the "IF..Then..Else" instruction over lines 11-13); or (2) we directly compare-switch the elements of b two by two from b_{0*2^n} (managed by the "IF..Then..Else" instruction over lines 11-13).
As the protocol works for the first recursion (for $n = 1$), the proof by induction is successfully initialised and the correctness of our protocol holds for any recursive step n. □

Lemma 6 (Privacy of Π_{merge}) **-**
The protocol Π_{merge} (Protocol 4.) is privately given that "compare_switch" is.

Proof. The compare switch operation being the only one that is dependent on the input data, the execution view of the protocol can be written as $(in, r, comp_1, ..., comp_x)$ with in the party input, r its random tape and $comp_1, ..., comp_x$ the messages induced by every oracle call to the "compare_switch" functionality. As such, the privacy of our Odd-Even Merge protocol depends on whether or not "compare_switch" is private. The "compare_switch" being defined in the ABB, the privacy of our protocol follows by the privacy of the ABB. □

iii. Alternative Solutions

Before arriving at the solution we presented above, we had envisioned other potential ways to implement similar functionalities. We detail here the reasons why we decided not to go further with those potential solutions.

Choice of privacy-enhancing primitives Starting the project, we were seeing three possible choices of privacy-enhancing primitives to use: Secret Sharing, Homomorphic Encryption and Differential Privacy. First of all, we identified Differential Privacy as not suitable for a 6-month research project. It would indeed have required a lot of experimentation with the security parameters in order to produce cohort selection private and precise enough to be used, which was unlikely to produce usable results in such a short period of time. Then, we gave up on Homomorphic Encryption (HE) because of its heavy online phase(Section II.i.5)). Indeed, while the overall execution time would quite probably be faster with HE than with Secret Sharing[34], Secret Sharing allows us to pour most of its computation during the pre-computation phase which is less restrictive to execute than the online phase.

Ordering with Hamda et al.[19] This technique is the one we were planning to use in the Research Project. In this paper, they propose an oblivious shuffling pre-processing step to be able to then use traditional sorting methods (such as the quicksort algorithm), which implies knowing the result of the comparison between two elements. Hamda et al. based their technique on the idea that, in order to know the ordering of the list after executing the sorting protocol, you need to know both the initial configuration of the list and the permutation that ordered the list. By first shuffling obliviously, the initial configuration is hidden from the potential adversary and the final ordering thus ends up being hidden too, even though leaking the comparison bits leaks the permutation. However, we ended up doubting the privacy of this scheme in a context where one party knows the big majority of the list to sort. Taking our context as an example, as P_1 knows about 99% of the joint database, an adversary that would corrupt P_1 could use the knowledge of the permutation to deduce some probability for a given element to be under-study. Indeed, knowing if an element is under-study is the same as knowing the result of the "lower-equal" comparison with its neighbouring elements. Leaking this permutation (i.e. the comparison bits when sorting) is thus potentially breaking the privacy of our protocol. Intuitively, on lists of sizes two and three (note that in bigger lists this would happen in every step while recursively sorting) we have :

- list of size 2 with n, m any two elements that can be compared :
  - $[n, m] \implies$ with $n > m$ We will know n is not under-study
  - $[n, m] \implies$ with n only case where we do not get information
- list of size 2 with n, m, p any three elements that can be compared :
  - $[n, m, p] \implies$ with $n, m < p$ we will know p is not under-study (if previous recursion was on part of the list before n)
  - $[n, m, p] \implies$ with $n, m > p$ we will know p is not under-study
  - $[n, m, p] \implies$ with $n > p \geq m$ we will know n is not under-study (if previous recursion was on part of the list after p)
  - $[n, m, p] \implies$ with $m > p \geq n$ we will know m is not under-study

The example of this scheme in particular led us to decide to search for another sorting protocol that would not introduce any leakage distinct from the final output. We also added the requirements to avoid schemes adding one (or more) independent third party, to reduce the execution cost of our final protocol, which led us to focus on sorting networks[21], finally choosing to use the Odd-Even Merge Sort by Batcher et al.[4].

V. Results

i. Theoretical analysis

We determine the theoretical runtime of our protocol by counting the number of operations processed during one execution. In particular, we reduce all those operations to the number of equalities, scalar multiplications and logical XOR, AND, NOT and OR executed. From this point on, we extract the number of bits that needs to be communicated throughout the protocol execution and we determine the runtime by counting our heaviest operation, the secret shared multiplication (i.e. the number of multiplication triple used).

We discuss in table 5, table 6 and table 7 the number of operations executed by each component of our protocols before presenting in table 10 what it implies regarding the bits communication and in table 11 what constants it creates in our computational complexity.

During this section, we denote by l the number of bits used to represent our elements, by n the number of records processed during the execution, by M the maximum cohort size and by N for the constant in "1:N matching".

TABLE 4: Summary of the variables in our complexity analysis

n	l	M	N
number of records	size of our data (bits)	maximum cohort size	1:N matching

Sorting network: According to Batcher et al.[4], the odd-even merge sort executes $2^{\log_2(n)-2}(\log_2(n)^2 - \log_2(n) + 4) - 1$ ($\approx \frac{n}{2}\log_2(n)^2 + 2n - 1 = \mathcal{O}(n\log_2(n)^2)$) comparison-switches (Section A.xii.). In our protocol, each time we execute a compare switch on a record we do one lower-than comparison on l bits binary values (implying $2 * l$ XOR, $2 * l$ AND, $2 * l$ NOT and l OR) before using this result to perform a switch on the four fields of the records. We thus have 4 binary switches (on the ID, $feats$, us and imc fields) and one Arithmetic switch (on the *cost* field), which implies 16 AND, 16 XOR, and 4 scalar multiplications.

TABLE 5: Summary of ABB operations to execute an Odd-Even Merge Sort on n records with fields encoded on l bits.

Protocol	total
EQ	0
XOR	$(16 + 2l) \cdot (2^{\log_2(n)-2}(\log_2(n)^2 - \log_2(n) + 4) - 1)$
AND	$(16 + 2l) \cdot (2^{\log_2(n)-2}(\log_2(n)^2 - \log_2(n) + 4) - 1)$
NOT	$(2l) \cdot (2^{\log_2(n)-2}(\log_2(n)^2 - \log_2(n) + 4) - 1)$
OR	$l \cdot (2^{\log_2(n)-2}(\log_2(n)^2 - \log_2(n) + 4) - 1)$
Arit Mult	$4 \cdot (2^{\log_2(n)-2}(\log_2(n)^2 - \log_2(n) + 4) - 1)$

Setting the under-study bit We compute an equality circuit between all pairs of neighbouring elements, which amounts to $n - 1$ executions of the said circuit. For each individual execution, we use one equality and one XOR operation (Protocol 1., line 3).

TABLE 6: Summary of ABB operations to set the under-study bits of a list of n records.

Protocol	total
EQ	$n - 1$
XOR	$2(n - 1)$
AND	0
NOT	0
OR	0

Setting in-matched-cohort We execute an equality circuit on every individual element of our list to verify if the element is under-study and, if it is, set the imc bit to 1 (= true). We thus execute n instances of the equality circuit, during which we access M elements to

the right and M elements to the left of the current element (Protocol 2., line 3 to 7). Each time we access one of those $2M$ elements, we execute one equality, one binary lower-than with the constant N (implying $2 * l$ XOR, $2 * l$ AND, $2 * l$ NOT and l OR), one binary ADD (implying $2 * l$ XOR, $2 * l$ AND, $2 * l$ NOT and l OR), 4 AND, 2 NOT and 1 XOR.

TABLE 7: Summary of ABB operations to set the in-matched-cohort bits of a list of n records organised in partitions of at most M elements and which fields are encoded on l bits.

Protocol	total
EQ	$2Mn$
XOR	$5 \cdot 2Mln$
AND	$(4 + 4l)2Mn$
NOT	$(2 + 4l)2Mn$
OR	$(2l)2Mn$

Focusing on the operations used throughout our execution, they do not all have the same impact on the communication or multiplicative complexity. In particular, are involved in the communication complexity only the equality, AND, OR and Scalar multiplication, having a respective communication of $log_2(l)^2$, $2l$, $2l$ and $2l$ bits (table 8).
Similarly, only the equality, AND, OR and Scalar multiplication participate in the multiplicative complexity. Those operations use respectively $log_2(l)$, 1, 1 and 1 multiplication triples. (table 9).

TABLE 8: Summary of the communication (in bits) induced by each of the operations of interest considering elements of l bits.

Protocol	Communication (bits)
EQ	$\log_2(2l)^2$
XOR	0
AND	$2l$
NOT	0
OR	$2l$
Scal. Mult	$2l$

TABLE 9: Summary of the number of multiplication triples used by each of the operations of interest considering elements of l bits.

Protocol	# Mult. triple
EQ	$\log_2(2l)$
XOR	0
AND	1
NOT	0
OR	1
Scal. Mult	1

Combining the figures put in evidence up until now, we can thus compute the communication and multiplicative complexity of the sub-protocols Π_{USG} and Π_{MCS}, which will give us the complexity of our Π_{PPCS} protocol (table 10 and table 11). In particular, the communication and multiplicative complexity of Π_{USG} is made of both the complexity from one Odd-Even Merge Sort and from setting the under-study bits, and the communication and multiplicative complexity of Π_{MCS} is made from the second Odd-Even Merge Sort and from setting the in-matched-cohort bits.

For readability purposes, we replaced the formula $2^{\log_2(n)-2} \cdot (\log_2(n)^2 - \log_2(n) + 4) - 1$ ($\approx \frac{n}{2}\log_2(n)^2 + 2n - 1 = \mathcal{O}(n\log_2(n)^2)$) by O_{merge}. Note that this formula defines the complexity of the Odd-Even Merge Sort algorithm as being *sub-quadratic*, and by extension, as it is the heaviest complexity in the algorithms we use, it defines the complexity of our full protocol as being *sub-quadratic* too.

TABLE 10: Summary of the communication (in bits) generated by executing each of our (sub) protocols on a database of n records with partitions of at most M elements and data encoded on l bits.
We note $O_{merge} = (2^{\log_2(n)-2} \cdot (\log_2(n)^2 - \log_2(n) + 4) - 1)$

Protocol	communication (# transmitted bits)
Π_{PPCS}	$\log_2(l)^2 \cdot (n-1) + 2(40+6l)l \cdot O_{merge} + (\log_2(l)^2 + 8 + 12l)l \cdot 2Mn$
Π_{USG}	$\log_2(l)^2 \cdot (n-1) + (40+6l)l \cdot O_{merge}$
Π_{MCS}	$(\log_2(l)^2 + 8 + 12l)l \cdot 2Mn + (40+6l)l \cdot O_{merge}$

TABLE 11: Summary of the multiplicative complexity of our (sub) protocols considering a database of n records with partitions of at most M elements and data encoded on l bits. We note $O_{merge} = (2^{\log_2(n)-2} \cdot (\log_2(n)^2 - \log_2(n) + 4) - 1)$

Protocol	complexity (# multiplication triples)
Π_{PPCS}	$\log_2(l) \cdot (n-1) + 2(20+3l) \cdot O_{merge} + (\log_2(l) + 5 + 6l) \cdot (M-1)n$
Π_{USG}	$(20+3l) \cdot O_{merge} + \log_2(l) \cdot (n-1)$
Π_{MCS}	$(\log_2(l) + 4 + 6l) \cdot 2Mn + (20+3l) \cdot O_{merge}$

ii. Empirical analysis

As per Kortekaas et al.[23], we tested the implementation using a commodity laptop (i7-10750H CPU @ 2.60GHz, 16GB RQM), using two single-threaded processes emulating P_1 and P_2. We used randomly generated datasets D_1 and $\mathcal{I}_2$ with sizes ranging from 10.000 and 100 records to 2.2M and 25.000 respectively (see full test settings in table 12). We simulated both servers P_1 and P_2 on the commodity laptop and made them communicate through local host. In particular, this implies that the amount of communication we were able to transmit per second was higher than what we could expect in practice, as it was equivalent to having access to an internet connection of 15Gb/s. We discuss the impact of the bandwidth capacity on the execution time further down in this section.

TABLE 12: Dataset size for evaluation settings

	TOY	SMALL	MEDIUM	FULL
$\lvert D_1 \rvert$	10.000	22.000	220.000	2.2M
$\lvert \mathcal{I}_2 \rvert$	100	250	2.500	25.000

Looking at the runtime for the different settings and putting it in relation to the runtime of the solution by Kortekaas et al.[23] on the same set-up, we put in evidence the *sub-quadratic* complexity of our solution (Table 13). Yet, due to the huge constants in front of our sub-quadratic factor (Table 10 and 11), our current solution does not perform better than the previous one in the given settings. It is however important to notice that the FULL setting is only representative of a cohort selection made at a national level, considering a country of similar size to the Netherlands (about 17.5M inhabitants).

Figure 3 makes our *sub-quadratic* complexity more explicit by putting it side by side with the previous solution's quadratic complexity. Moreover, we can notice that our current solution would start outperforming the previous one in settings containing more than 3M

records. This leads us to observe that our solution should, at least, be providing a significant practical improvement considering countries about twice the size of the Netherlands.

TABLE 13: Run time of our protocol (in seconds) assuming $M = 10$ and $l = 32$, with comparisons to the run-time of Kortekaas et al.[23] in 1:1 matching

	TOY	SMALL	MEDIUM	FULL
Π_{USG}	$2.39 \cdot 10^2$	$6.44 \cdot 10^2$	$9.60 \cdot 10^3$	$1.37 \cdot 10^5$
Π_{MCS}	$2.97 \cdot 10^2$	$7.56 \cdot 10^2$	$1.07 \cdot 10^4$	$1.49 \cdot 10^5$
Π_{PPCS}	$5.36 \cdot 10^2$	$1.41 \cdot 10^3$	$2.03 \cdot 10^4$	$2.85 \cdot 10^5$
current Kortekaas et al.[23]	121.27	60.26	8.60	1.30

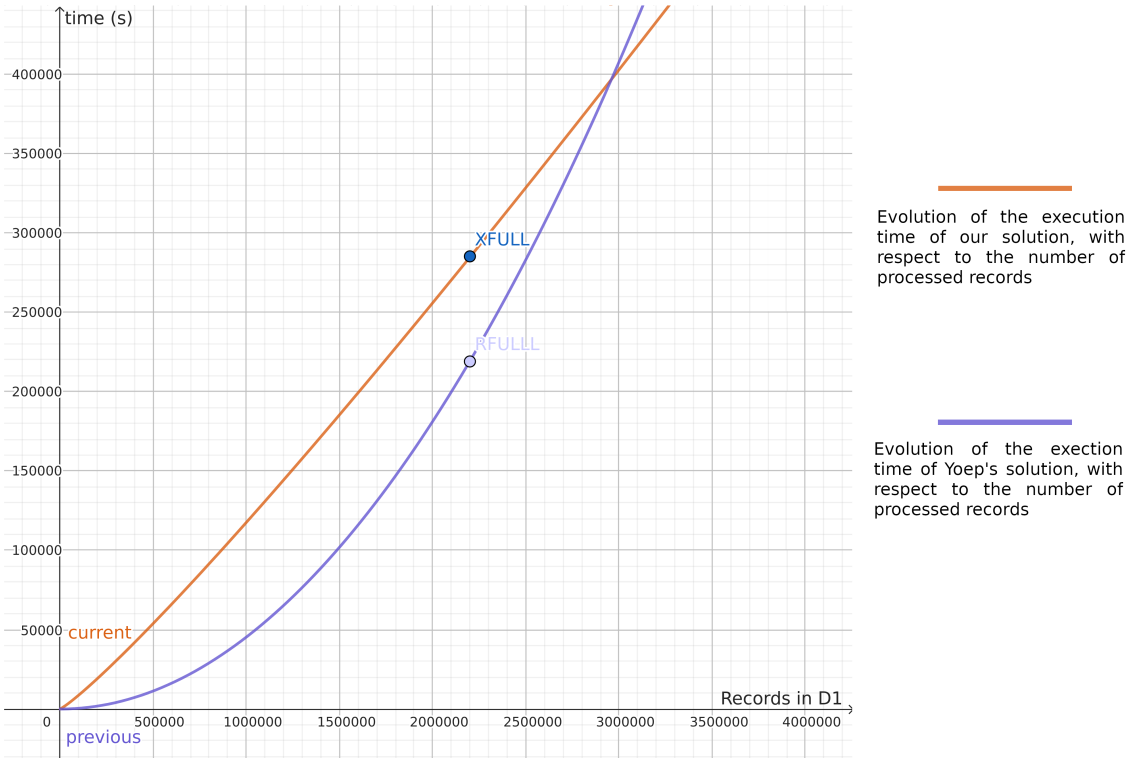

FIGURE 3: Evolution of the execution time of the current and previous solution for a 1:1 matching, with M=10 and l = 32.

Our solution also improves on the solution of Kortekaas et al.[23] considering a cohort selection wtih 1:N matching ($N > 1$). Indeed, the execution time of the current solution represented in Figure 3 would not change whether we match in 1:1 or 1:10. This is because, considering our settings definition (Section III.i.), we do not have any guarantee that the probability to get two under-study elements in a same partition is low. As such, we need to access every element in the cohort to ensure that we will always match all necessary elements, which implies processing M elements at the right and left of every under-study element (with $M = 10$ the maximal cohort size, Table 4). This means that our solution would start outperforming the previous solution in the FULL setting for 1:N matching with $N > 3$. Assuming that the partition is made so that there is a low probability to find two under-study elements in a same partition, we could thus have some further improvements by defining the number of elements to process using the constant N instead of M. In such a case and staying in the FULL setting, we could expect our solution to outperform the

solution of Kortekaas et al.[23] for any 1:N matching case where $N > 1$ (table 4).

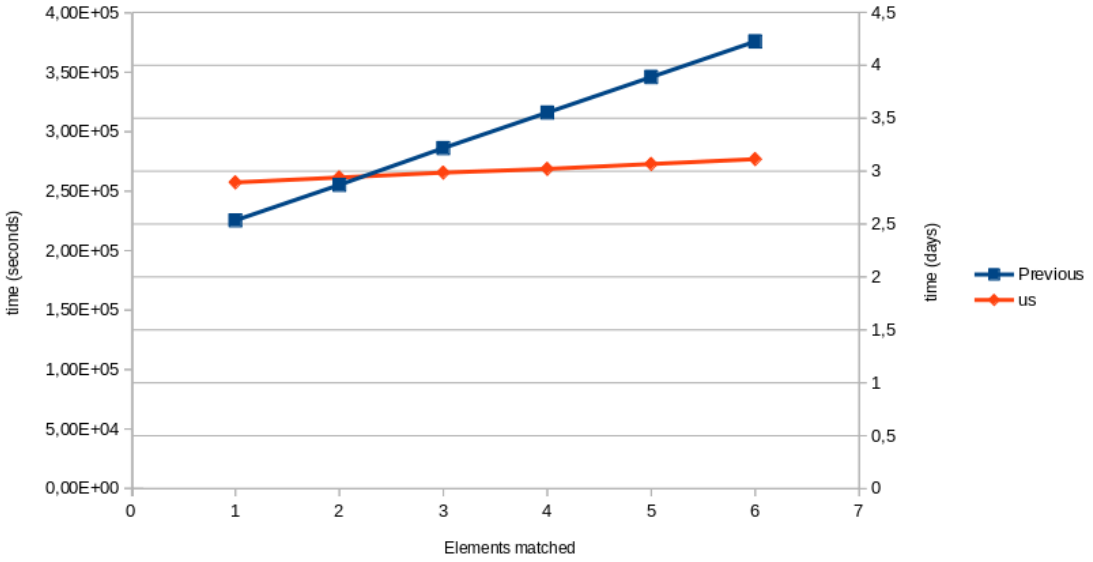

FIGURE 4: Evolution of the execution time of both the previous and current solution given the number of elements N to match for each under-study element in the FULL context

As per the solution of Kortekaas et al.[23], the bottleneck of our runtime is the bandwidth capacity rather than the computational power of our system. Indeed, always considering the FULL setting, the local computation only makes up for 4.4% of the total execution time ($\approx$ 3.5 hours on $\approx$ 79 hours). Figure 5 puts this observation in perspective by putting together the total execution time (orange curve) and the local computation time (green curve). Visually, assuming an optimal bandwidth, there would basically be no communication overhead (i.e. additional time due to the communication) and the two curves would be flushed.

There however does not yet exist communication channels that could support such a communication per second. Indeed, as shown in Figure 6, assuming similar computational power as during our tests and a setting with 2.2M of records to process, our program would generate in average 324 Gbits/s, of which more than two-thirds are due to the sorting network. Ultimately, it is reasonable to expect that executing our protocol in a real case scenario would be slower than our tests, as it is quite unlikely that both involved parties will have access to a bandwidth of 15 Gb/s or more. On the flip side, this also means that it is not necessary to allocate a lot of computational power to the online phase of our protocol, as it would be limited by the bandwidth anyway (note that this remark does not apply when *simulating* the protocol, as the size of the simulated bandwidth depends on the computational power). Assuming the data of both parties are stored in a data centre where they are allocated a 2.5 Gb/s cable, the cohort selection would be about six times slower to execute in practice, resulting in about 19 days of online execution time for our FULL setting (with 2.2M records). In the best case scenario, where both parties have access to a bandwidth of 10 Gb/s, the practical execution would still be about 30% slower resulting in about 4.2 days of online execution time (again for the FULL setting).

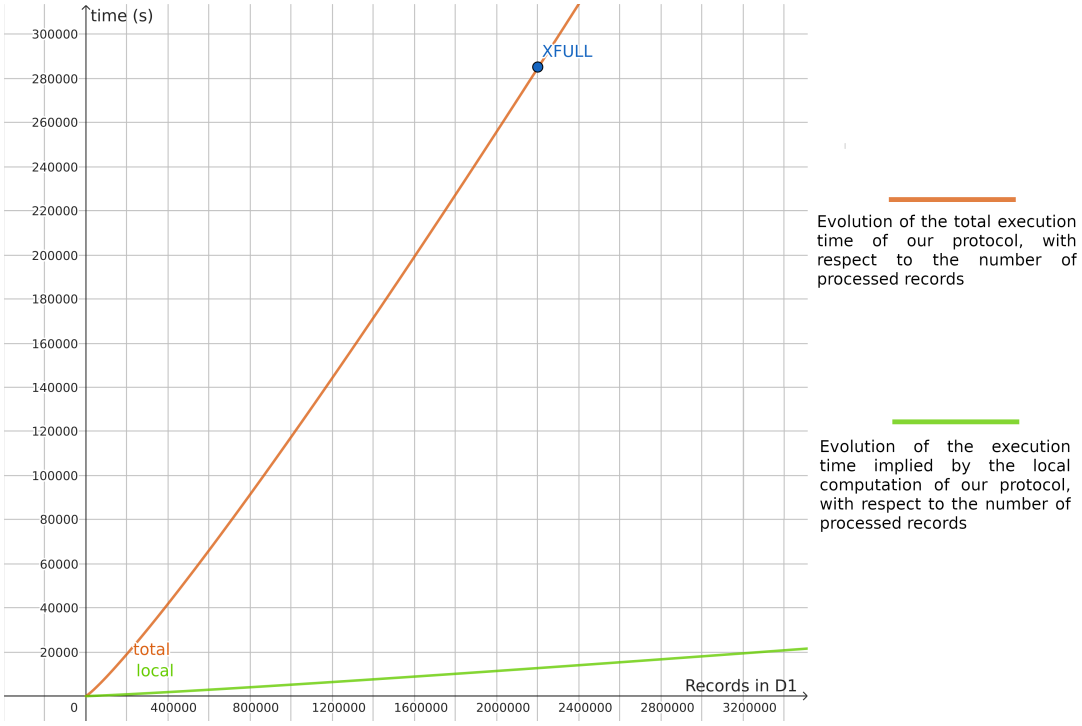

FIGURE 5: Evolution of the local and total execution time of the protocol depending on the number of records in D_1, assuming $|\mathcal{I}_2| = |D_1|/100$ and given $M = 10$

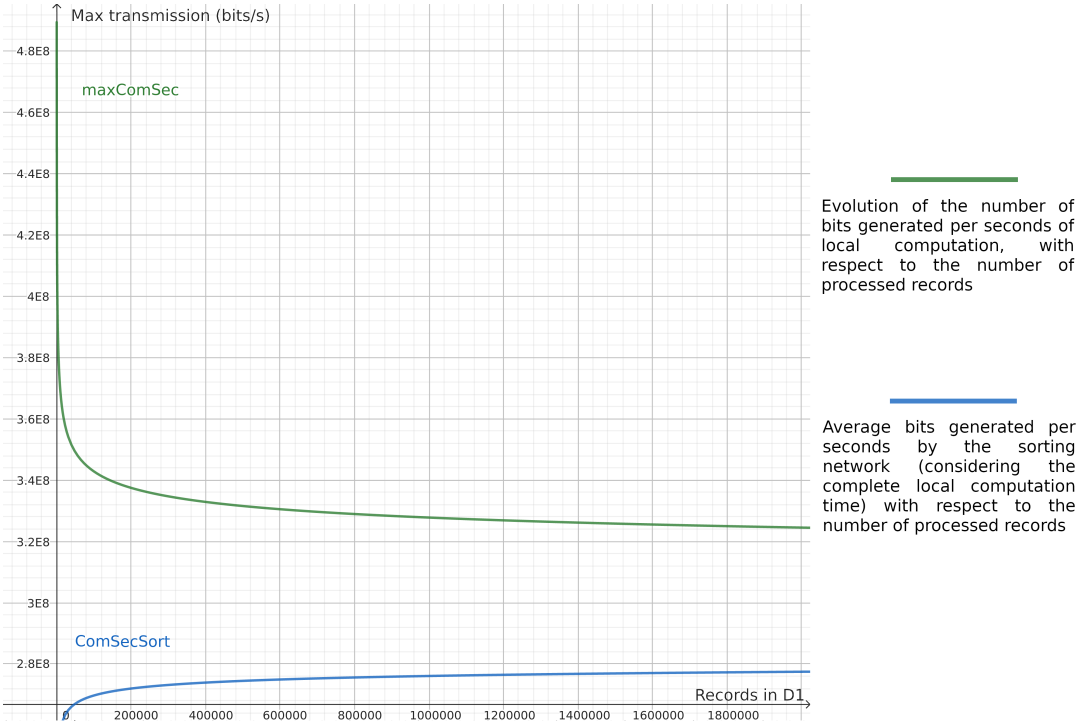

FIGURE 6: Average traffic generated by our program (on our set-up) per second of local computation depending on the number of records in D_1, assuming $|\mathcal{I}_2| = |D_1|/100$ and given $M = 10$

VI. Potential optimisations

Regarding our results, while the initial goal of a sub-quadratic complexity is indeed fulfilled, the large constants lead us to practical results worst than the previous work by Kortekaas et al.[23] when considering settings of about 2.2M records (representative of a cohort selection at a national level in the Netherlands). This slowness is induced by the large amount of data that needs to be communicated between servers during the execution, and in particular during the oblivious swaps needed when executing the sorting network. In this section, we present a potential optimisation to reduce those communications, as well as unsuccessful optimisation ideas.

i. Optimising comparison with constants : Function Secret Sharing[7]

Literature review and application to our case. Function Secret Sharing (FSS) is a concept introduced by Boyle et al.[7], that is, in the idea, comparable to additively sharing the truth table of a simple function between two or more servers. Mainly thought for Private Information Retrieval (PIR), the Distinct Point Function (DPF) is a subclass of FSS where one creates a function $f_{\alpha,\beta}$ that can be evaluated on any element x to compare it with a constant α, and returns a result β if the comparison is successful. Typically, the function's truth table would be shared as a tree, and evaluating $f_{\alpha,\beta}$ on x would be equivalent to seeing which path in the tree is taken by x. This also shows the deterministic nature of FSS which thus implies the need for some precautions before using DPFs. To begin with, we would need to generate during the pre-computation phase one DPF $f_{\langle\alpha\rangle_\gamma,\beta}$ for every comparison we want to do with α, $\langle\alpha\rangle_\gamma$ representing α shared additively with respect to a random γ. From there, evaluating $f_{\langle\alpha\rangle_\gamma,\beta}$ on x would be done in two steps: obtaining $\langle x\rangle_\gamma$ and executing $f_{\langle\alpha\rangle_\gamma,\beta}(\langle x\rangle_\gamma)$ (which only asks for one communication between servers). However, with respect to the initial papers on FSS and DPFs, using DPFs for comparison in our case brings issues related to the key (tree) generation in the distributed setting. How could we generate the decision tree using only our two servers (P_0 and P_1) without leaking the meaning of the different paths? This issue has been tackled in the latest research on FSS [8, 15]. In particular, it got first tackled by Doerner et al.[15] and then further improved by Boyle et al.[8].

Time required to evaluate the DPFs The papers on this subject being focused on PIR, the complexity mentioned is often with respect to a function EvalAll that executes the DPF on every element in the domain. In our case, however, we are interested in the execution time per element (for the Eval operation). In particular, we focus on the work by Doerner et al.[15], as what we would need seems to be closely related to their OramWrite protocol. Focusing on the full version of their paper we notice that their optimised FSS algorithm does not seem to need any communication for the Eval operation. Translating their work into our context, we would thus have a single online communication (during the online phase) to obtain the aforementioned $\langle\alpha\rangle_\gamma$.

Concrete impact This optimisation would impact the communication complexity when setting the *imc* bit, by limiting communication when comparing the counter and the value N (Protocol 2.). The obtained complexity (table 14 and 15) would indeed improve even so slightly on the current ones (table 7 and 10). This would however not make for any significant increase in the online computation time.

TABLE 14: Summary of ABB operations to set the in-matched-cohort bits of a list of n records organised in partitions of at most M elements and which fields are encoded on l bits (if using FSS for comparisons with constants).

Protocol	total
EQ	$2Mn$
XOR	$5 * 2Mln$
AND	$(4 + 2l)2Mn$
NOT	$(2 + 2l)2Mn$
OR	$l \cdot 2Mn$

TABLE 15: Summary of the communication (in bits) generated by executing each of our (sub) protocols on a database of n records with partitions of at most M elements and data encoded on l bits (when using FSS for comparisons with constants). We note $O_{merge} = (2^{\log_2(n)-2} \cdot (\log_2(n)^2 - \log_2(n) + 4) - 1)$

Protocol	complexity (# communication)
Π_{PPCS}	$\log_2(l)^2 \cdot (n-1) + 2(40 + 6l)l \cdot O_{merge} + (\log_2(l)^2 + 8 + \mathbf{6l})l \cdot 2Mn$
Π_{USG}	$\log_2(l)^2 \cdot (n-1) + (40 + 6l)l \cdot O_{merge}$
Π_{MCS}	$(\log_2(l)^2 + 10 + \mathbf{6l})l \cdot 2Mn + (40 + 6l)l \cdot O_{merge}$

ii. Discarded optimisations

Optimising comparison with secrets: Function Secret Sharing[7] Similar to the optimisation of comparisons with constants, we would have liked to optimise comparisons between secret s_1 and s_2. We had two potential solutions: (1) creating DPFs on secret shared elements and (2) comparing $s_1 - s_2$ to 0. Unfortunately, (1) is dependent on the protocol's input as we need to have access to the secret shared elements to execute a distributed FSS Generation algorithm such as the one presented by Doerner et al[15]. This would force us to generate the FSS in the online phase (Section II.i.5)) thus making for a slower option than the one we are currently using. Similarly, proceeding with a subtraction between secrets in (2) translates in practice to asking the ABB to compute $\langle s_1 \rangle - \langle s_2 \rangle$. This is as slow as executing a "lower-than" operation which uses $2l + 1$ multiplication triples against $\log_2(2l)$ multiplication triples for an equality (with l the bit size of the elements). We thus decided not to look further into this idea.

Replacing online communication by local computation using FSS-based ORAM ORAM stands for Oblivious Random Access Memory and denotes the set of techniques used to hide the local access patterns. Such techniques usually find uses in distant or distributed database storage, where analysing the frequency of data accesses may leak sensitive information to the server(s) hosting the data.

The idea of this optimisation would be to replace some of the communication due to sorting (in particular due to swapping the fields ID, FEAT and COST) by some local computation that could be further optimised through code parallelisation. In our theoretical set-up, P_0 and P_1 would be considered as hosting the data, and the secure computation (the Arithmetic Black Box) would act as the client. We would now create two secret shared lists of size $n = |D_1| + |\mathcal{I}_2|$: $L_{DB} = \{ID, feat, cost, us, imc\}$ and $L_{point} = \{\$p\}$ (with $\$p$ pointing towards an element of L_{DB}) that will be used to do the sorting. Notice that L_{point} could

be generated in the off-line phase, as it only needs an empty L_{DB} to be initialised.

There are plenty of publications about ORAM, but the most efficient ones seem to be related to Function Secret Sharing (FSS)[7]. In particular, Boyle et al.[8] and Doerner et al.[15] both published some fairly fast solutions using FSS, which are however built to minimise the computation time of key generation (that needs to be done by the client), replacing it by a heavier computation when evaluating the Tree (that needs to be done by the servers). Applied to our case, the naive "unoptimised" implementation mentioned in [8] seems to be the most promising, as the computation that needs to be done by the client can be executed during the offline phase. The time necessary to compute EvalAll grows linearly in the number of records, so we can expect the time necessary to retrieve an element in a context with 2.2M records to be of 0.001584 seconds, which would mean $0.001584 \cdot 2 \cdot O_{merge} = 7.44 \cdot 10^5$ seconds for a full sort. While this is dependent on the computer setup, it still seems a bit discouraging. However, this time being due to local computation it may very well be further optimisable through parallelization and may sill be interesting given a big enough decrease in the communication complexity. As such, following this setup, and keeping the same protocol idea as before, We are now interested in the new communication complexity of our protocol (Table 16 and 17). Unfortunately, with respect to our current figures (table 10 and 13) this optimisation idea does not seem to lead to a big enough decrease in the communication complexity to counterbalance the increased local computation time introduced by the ORAM.

TABLE 16: Summary of ABB operations to execute an Odd-Even Merge Sort on n records with fields encoded on l bits (if using FSS-based ORAM).

Protocol	total
EQ	0
XOR	$(4+2l) \cdot (2^{\log_2(n)-2}(\log_2(n)^2 - \log_2(n) + 4) - 1)$
AND	$(4+2l) \cdot (2^{\log_2(n)-2}(\log_2(n)^2 - \log_2(n) + 4) - 1)$
NOT	$(2l) \cdot (2^{\log_2(n)-2}(\log_2(n)^2 - \log_2(n) + 4) - 1)$
OR	$l \cdot (2^{\log_2(n)-2}(\log_2(n)^2 - \log_2(n) + 4) - 1)$

TABLE 17: Summary of the communication (in bits) generated by executing each of our (sub) protocols on a database of n records with partitions of at most M elements and data encoded on l bits (when using FSS-based ORAM). We note $O_{merge} = (2^{\log_2(n)-2} \cdot (\log_2(n)^2 - \log_2(n) + 4) - 1)$

Protocol	complexity (# communication)
Π_{PPCS}	$\log_2(l)^2 \cdot (n-1) + 2(\mathbf{8}+6l)l \cdot O_{merge} + (\log_2(l)^2 + 10 + 12l)l \cdot 2Mn$
Π_{USG}	$\log_2(l)^2 \cdot (n-1) + (\mathbf{8}+6l)l \cdot O_{merge}$
Π_{MCS}	$(\log_2(l)^2 + 10 + 12l)l \cdot 2Mn + (\mathbf{8}+6l)l \cdot O_{merge}$

VII. Related Work

Excluding the research from Kortekaas et al.[23] from which this work is a direct successor, there is little to none literature on the topic of Privacy Preserving Cohort Selection (PPCS). In particular, we only denote two other works focusing on Privacy Preserving Cohort Selection assuming horizontally partitioned data[12, 36]. Assuming that all par-

ties involved possess the same amount of information on the individuals in their database (horizontal partitioning) allowed them to implement a hidden joined query-able database through Searchable Encryption. However, we specifically aim at providing an option allowing the research centre not to collect any data about its test subject, which would not be a possibility in those previous works. As such, a setting with horizontally partitioned data was not coherent with our requirements, and their solutions are not directly related to ours.

There are however some significant literature available in the closely related field of Private Set Intersection (PSI). PSI denotes the schemes aiming at privately computing the intersection $X \cap Y$ between two distant sets X and Y [11, 30]. In particular, the latest research in this field showed some schemes made to be used together with MPC to perform some further computation on the intersection instead of returning it directly [13, 25]. While it could theoretically have been used to identify the under-study elements, the way it extracts the intersection could have led to a significant amount of additional computations.

When it comes to sorting obliviously, there exist different possible approaches in the literature. First, here are different categories of sorting algorithms with oblivious data access, of which the sorting networks[4, 35] that we decided to use, and for which $\mathcal{O}(nlog_2(n)^2)$ seems to be the limit for generalised sorting networks (not that for networks tailored to specific list sizes, it is possible to slightly go below this $nlog_2(n)$ barrier[21, Section 5.3.4]). Aside from sorting networks, we also find randomised schemes[18, 27, 28] that usually accept a smaller complexity than the sorting networks but have a low probability of failing. We considered the probability of failing to be too much of an issue given our use case, hence why we decided to go with the slightly slower, but constant, sorting networks. Other oblivious sorting schemes will rely on a trusted third party to do all or part[19] of the permutation in order to hide the access pattern. Aside from the concern we had inherent to [19] (Section IV.iii.), we also considered that using additional third parties would induce unnecessary costs and thus decided to not go with those types of protocols.

VIII. Conclusion

During this Master internship, we built a solution for two-party PPCS on vertically partitioned data. The solution is proven secure in the semi-honest model and runs in sub-quadratic time (and in particular in $\mathcal{O}(n \log_2(n)^2)$), efficiently improving on the complexity of the previous research by Kortekaas et al.[23]

The complexity achieved being sub-quadratic, it fulfils the initial objective of this research. While this does translate into some practical improvements considering our test settings, it is important to notice that it is far from being the case in every use case of interest. In fact, while we performed at best 30% slower than the previous research on our test settings (representative of a 1:1 cohort selection at a national level in a country of about 17M or fewer inhabitants), we can foresee some significant improvement on bigger populations. According to our simulation, we indeed start achieving some practical improvements when working on settings with 3M records or more, thus making our solution faster considering cohort selection at the European level for example. Additionally, considering a cohort selection with 1:N matching (and $N \geq 3$) or solution performs similarly or better than the previous one. We can moreover notice that the efficiency of our solution in 1:N matching can be further improved depending on how the initial partitions have been created. Assuming there is a low probability to get 2 under-study elements in a same partition, we

could indeed slightly modify our protocol in such a way that it would perform similarly or better than the previous solution for any 1:N matching with $N \geq 2$.

Notice that the bandwidth bottleneck will prove to be the biggest limitation in a practical use case. Indeed, testing our program by simulating our two servers on the same local machine allowed us to communicate 15 Gb/s of data, which is not representative of reality. Considering an average data centre, we can expect to have access to a cable with a capacity of about 2.5 Gb/s and up to 10 Gb/s, which would make the execution time 1.30 to 6 times slower. While we propose optimisation ideas, this likely won't have a big enough impact on the communication overhead to solve this bottleneck even though it will slightly mitigate it. As long as we still consider sorting networks, we might not be able to find any satisfying optimisation for our protocol. Except if we have access to an efficient ORAM, we indeed *need* to swap every field of the records to stay oblivious when sorting. Any significant optimisation on a part of the protocol would obviously have a much-welcomed impact but, as shown in Figure 6, the sorting network puts a hard limit on our minimal communication complexity.

m-party setting ($m > 2$) - While we only focused on the two-party setting in our current work, there are some incentives in enabling our solution to work in the m-party setting ($m > 2$). Indeed, as we are working with vertically partitioned data we end up matching the control group exclusively on individuals present in both databases (from the hospital and the research centre), which implies a non-negligible probability that not all test cases will be matched to a control case. Jointly executing a cohort selection with multiple hospitals could help mitigate this issue by extending the set in which we select the control group.
However, notice that some parts of our protocol may have to be modified to fit this case. We indeed have two situations that may arise in the case of m-party PPCS. (1) All m databases are *concatenated* together, and the protocol is executed on the resulting list. This would ask us to modify our condition to set an element as under-study, as detecting a collision would not be equivalent anymore to detecting an under-study element. (2) The $m-1$ hospitals' databases are *joined* together eliminating the duplicates at the same time, and we then concatenate the resulting database with the research centre's database. In this case, our current methods should still work.

Fully malicious model - As we placed ourselves in the semi-honest model, while our protocol is private it is still fully vulnerable to a malicious attacker. As medical data tends to be a recurring target for hackers, it may be of interest to extend the current work to prove its security in the malicious model. Given the current protocol, the translation of the ABB in the fully malicious model should be rather straightforward. The ABB as well as MPC primitives implemented in the ABB indeed all have known secure implementation in the malicious model. It thus remains to tailor the rest of the protocol to the malicious model, to ensure that the adversary cannot possibly diverge from it. Considering the optimisation mentioned in V.i., the feasibility of the FSS distributed Generation protocol in the malicious model has been discussed by Boyle et al. earlier this year [9].

IX. Code availability

You can find the complete implementation of the protocols described in this work in the following GitHub repository: https://github.com/utwente-scs/subquadratic-ppcs.

To execute the proof of concept, you will also need to use the library "Replicated secret sharing", created by my supervisor Yoep Kortekaas, that we modified for the current implementation. You can find the latest version used in this work in the following GitHub directory, under the MIT-License: https://github.com/utwente-scs/rustsecretsharing/tree/from-antoine

References

[1] Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. Secure training of decision trees with continuous attributes. Cryptology ePrint Archive, Paper 2020/1130, 2020.

[2] Guadalupe Aguilar-Madrid, Eduardo Robles-Pérez, Cuauhtémoc Arturo Juárez-Pérez, Isabel Alvarado-Cabrero, Flavio Gerardo Rico-Méndez, and Kelly-García Javier. Case-control study of pleural mesothelioma in workers with social security in mexico. *Am. J. Ind. Med.*, 53(3):241–251, March 2010.

[3] Axel Bacher, Olivier Bodini, Hsien-Kuei Hwang, and Tsung-Hsi Tsai. Generating random permutations by coin tossing: Classical algorithms, new analysis, and modern implementation. *ACM Trans. Algorithms*, 13(2), feb 2017.

[4] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), page 307–314, New York, NY, USA, 1968. Association for Computing Machinery.

[5] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[6] Amos Beimel. Secret-sharing schemes: A survey. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 6639 LNCS:11–46, 2011.

[7] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing: Improvements and Extensions. *Cryptology ePrint Archive*, 2018.

[8] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 121–151, Cham, 2022. Springer Nature Switzerland.

[9] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Information-theoretic distributed point functions. Cryptology ePrint Archive, Paper 2023/028, 2023. https://eprint.iacr.org/2023/028.

[10] Stefan Breitenstein, Antonio Nocito, Milo Puhan, Ulrike Held, Markus Weber, and Pierre-Alain Clavien. Robotic-assisted versus laparoscopic cholecystectomy: outcome and cost analyses of a case-matched control study. *Ann. Surg.*, 247(6):987–993, June 2008.

[11] Hao Chen, Kim Laine, and Peter Rindal. Fast Private Set Intersection from Homomorphic Encryption. *Cryptology ePrint Archive*, 2017.

[12] Arnab Chowdry, Alexander Kompel, and Michael Polcari. Cohort selection with privacy protection, Oct 2018.

[13] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11035 LNCS:464–482, 2018.

[14] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 247–264, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[15] Jack Doerner and Abhi Shelat. Scaling oram for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 523–535, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Ayman El-Menyar, Brijesh Sathian, Bianca M Wahlen, Husham Abdelrahman, Ruben Peralta, Hassan Al-Thani, and Sandro Rizoli. Prehospital administration of tranexamic acid in trauma patients: A 1:1 matched comparative study from a level 1 trauma center. *Am. J. Emerg. Med.*, 38(2):266–271, February 2020.

[17] Anne M. Euser, Carmine Zoccali, Kitty J. Jager, and Friedo W. Dekker. Cohort Studies: Prospective versus Retrospective. *Nephron Clinical Practice*, 113(3):c214–c217, oct 2009.

[18] Michael T. Goodrich. *Randomized Shellsort: A Simple Oblivious Sorting Algorithm*, pages 1262–1277. Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA).

[19] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Lncs 7839 - practically efficient multi-party sorting protocols from comparison sort algorithms. 2012.

[20] Masao Iwagami and Tomohiro Shinozaki. Introduction to Matching in Case-Control and Cohort Studies. *Annals of Clinical Epidemiology*, 4(2):33–40, 2022.

[21] Donald E Knuth. *The art of computer programming: Volume 3: Sorting and Searching*. Addison-Wesley Professional, 1998.

[22] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. Cryptology ePrint Archive, Paper 2009/411, 2009.

[23] Yoep Kortekaas. Privacy-Preserving Cohort Selection over Vertically Partitioned Data. *Internal Archive*.

[24] Yehuda Lindell. Secure Multiparty Computation (MPC). *Cryptol. ePrint Arch.*, 64(1):86–98, jan 2020.

[25] Payman Mohassel, Peter Rindal, and Mike Rosulek. Fast Database Joins and PSI for Secret Shared Data. 2020.

[26] Goldreich Oded. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, USA, 1st edition, 2009.

[27] Olga Ohrimenko, Michael T. Goodrich, Roberto Tamassia, and Eli Upfal. The melbourne shuffle: Improving oblivious storage in the cloud. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 556–567, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[28] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. CacheShuffle: An oblivious shuffle algorithm using caches. May 2017.

[29] Hao Peng, Mingzhi Zhang, Xiaoqin Cai, Jennifer Olofindayo, Anna Tan, and Yonghong Zhang. Association between human urotensin II and essential hypertension–a 1:1 matched case-control study. *PLoS One*, 8(12):e81764, December 2013.

[30] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable Private Set Intersection Based on OT Extension. *Cryptology ePrint Archive*, 21(2), jan 2016.

[31] Jeremy A Rassen, Abhi A Shelat, Jessica Myers, Robert J Glynn, Kenneth J Rothman, and Sebastian Schneeweiss. One-to-many propensity score matching in cohort studies. 21 Suppl 2:69–80, May 2012.

[32] PAUL R. ROSENBAUM and DONALD B. RUBIN. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 04 1983.

[33] Tomas Toft et al. Primitives and applications for multi-party computation. *Unpublished doctoral dissertation, University of Aarhus, Denmark*, 2007.

[34] Thijs Veugen, Frank Blom, Sebastiaan J.A. De Hoogh, and Zekeriya Erkin. Secure comparison protocols in the semi-honest model. *IEEE Journal on Selected Topics in Signal Processing*, 9(7):1217–1228, oct 2015.

[35] Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, jan 1968.

[36] Jiawei Yuan, Bradley Malin, François Modave, Yi Guo, William R. Hogan, Elizabeth Shenkman, and Jiang Bian. Towards a privacy preserving cohort discovery framework for clinical research networks. *Journal of Biomedical Informatics*, 66:42–51, February 2017.

A. ABB primitives

i. Load

We assume (for valid i, j) that P_i owns a secret s (defined over $\mathbb{S} = \mathbb{Z}_2$ or $\mathbb{Z}_{2^{64}}$) to share (in binary or arithmetic representation). It generates β selected uniformly at random over $\mathbb{S}$, set $\langle s \rangle = \beta$ (i.e. $P_i : \langle s \rangle \leftarrow s$) and send $s - \beta$ ($s \oplus \beta$ when sharing bit representation) to P_j. It is clear that it behaves according to the ABB if P_j waits for a message m from P_i and sets $\langle s \rangle = m$ (i.e. $P_j : \langle s \rangle \leftarrow ?$).

$\mathbf{\Pi_{load-Arith}}$: Share a secret (Arithmetic)

Input: P_i: an arithmetic secret s defined over $\mathbb{S}$.
Output: Secret shares of s for P_i, P_j.
Protocol:

$P_i : \langle s \rangle_i \leftarrow \text{rand\_over}(\mathbb{S})$
$P_i : \text{send}(P_j, s - \langle s \rangle_i)$
$P_j : \langle s \rangle_j \leftarrow \text{wait\_for}(P_i, m)$

$\mathbf{\Pi_{load-Bin}}$: Share a secret (Binary)

Input: P_i: a binary secret s defined over $\mathbb{S}$.
Output: Secret shares of s for P_i, P_j.
Protocol:

$P_i : \langle s \rangle_i \leftarrow \text{rand\_over}(\mathbb{S})$
$P_i : \text{send}(P_j, s \oplus \langle s \rangle_i)$
$P_j : \langle s \rangle_j \leftarrow \text{wait\_for}(P_i, m)$

Security intuition: β being uniformly random, we can affirm that $\langle s \rangle_i$ and $\langle s \rangle_j$ are uniformly random and (computationally) independent. It follows that for a corrupted P_j, the value received from P_i is indistinguishable from random.

ii. Reveal

We assume that P_1, P_2 each owns a (arithmetic or binary) share of a secret s, that they want to reveal. On call P_i : $sec \leftarrow \text{reveal}(s)$ from both parties, the ABB will answer with $\langle s \rangle_1 + \langle s \rangle_2$ (resp $\langle s \rangle_1 \oplus \langle s \rangle_2$ in case of binary shares).

$\mathbf{\Pi_{Reaveal-Arith}}$: Reveal a secret (Arithmetic)

Input: P_i: an arithmetic secret share $\langle s \rangle_i$ for $i \in \{1, 2\}$.
Output: revealed secret s.
Protocol:

$P_1 : s \leftarrow \text{reveal}(\langle s \rangle_1)$
$P_1 : s \leftarrow \text{reveal}(\langle s \rangle_2)$
$\text{assert}(s = \langle s \rangle_1 + \langle s \rangle_2)$

$\mathbf{\Pi_{Reavel-Bin}}$: Reveal a secret (Binary)

Input: P_i: a binary secret share $\langle s \rangle_i$ for $i \in \{1, 2\}$.
Output: revealed secret s.
Protocol:

$P_1 : s \leftarrow \text{reveal}(\langle s \rangle_1)$
$P_1 : s \leftarrow \text{reveal}(\langle s \rangle_2)$
$\text{assert}(s = \langle s \rangle_1 \oplus \langle s \rangle_2)$

---

iii. Not

We assume that P_1, P_2 each owns a (binary) share of a secret s. To obtain shares of the secret $\neg s$, P_1 negates its share and P_2 let it as is. Notice that when revealing, given β the hiding, we indeed find $\neg\beta \oplus s \oplus \beta = s \oplus \{1\}^* = \neg s$. Given $i \in \{1, 2\}$:

$\mathbf{\Pi_{not}}$: negates a shared secret

Input: P_i: share $\langle s \rangle_i$ of secret to negate
Output: P_i: $\langle \neg s \rangle_i$.
Protocol:

$P_1 : \langle \neg s \rangle_1 \leftarrow \neg \langle s \rangle_1$
$P_2 : \langle \neg s \rangle_2 \leftarrow \langle s \rangle_2$

Security intuition: No communication between parties is required, and it follows that no party learn more about s or $\neg s$ than what they knew before executing the protocol.

iv. Xor

We assume that P_1, P_2 each owns (binary) shares of two secrets s_1, s_2. To obtain shares of the secret $s_1 \oplus s_2$ both parties simply xor their person shares. Given $i, j \in \{1, 2\}$:

$\mathbf{\Pi_{xor}}$: xor two shared secrets

Input: P_i: share $\langle s_j \rangle_i$ of secrets to xor
Output: P_i: $\langle s_1 \oplus s_2 \rangle_i$.
Protocol:

$P_1 : \langle s_1 \oplus s_2 \rangle_1 \leftarrow \langle s_1 \rangle_1 \oplus \langle s_2 \rangle_1$
$P_2 : \langle s_1 \oplus s_2 \rangle_2 \leftarrow \langle s_1 \rangle_2 \oplus \langle s_2 \rangle_2$

Security intuition: No communication between parties is required, and it follows that no party learn more about s_1, s_2 or $s_1 \oplus s_2$ than what they knew before executing the protocol.

v. And[5]

We assume that P_1, P_2 each own a (binary) share of two secrets s_1, s_2 and of a multiplication triple a,b,c, such that a and b are random and $c = a * b$ and none of the parties knows a,b or c. From there we reveal $A = s_1 \oplus a$ and $B = s_2 \oplus b$, and set $s_1 * s_2 = \langle c \rangle \oplus \langle s_2 \rangle \wedge A \oplus \langle s_1 \rangle \wedge B \oplus A \wedge B$ which gives $x \wedge y \oplus r$ with r (pseudo) randomness depending on ABB's security, and $r = (s_2 \wedge a \oplus \langle s_1 \rangle \wedge b \oplus \langle s_1 \rangle \wedge s_2 \oplus \langle s_2 \rangle \wedge a \oplus \langle s_2 \rangle \wedge s_1)$

$\mathbf{\Pi_{and}}$: or of two secret shares

Input: P_i: shares $\langle s_j \rangle_i$ of secrets to 'AND', shares of a multiplication triple $a * b = c$
Output: P_i: $\langle s_1 \wedge s_2 \rangle_i$.
Protocol:

$P_{i/j} : A \leftarrow \text{Reveal}(s_1 \oplus a)$
$P_{i/j} : B \leftarrow \text{Reveal}(s_2 \oplus b)$
$P_{i/j} : \langle s_1 \wedge s_2 \rangle_{i/j}$
$\leftarrow \langle c \rangle_{i/j} \oplus \langle s_2 \rangle_{i/j} \wedge A \oplus \langle s_1 \rangle_{i/j} \wedge B \oplus A \wedge B$

Security intuition: Our result is indistinguishable from random if $\langle c \rangle_{i/j}$, $\langle s_1 \rangle_{i/j}$, $\langle s_2 \rangle_{i/j}$, A and B are. As a and b are uniformly random unknown elements, A and B are indistinguishable from random as well. Then, our protocol's security thus follows from the security of our secret shared 'Load' (Section A.i.) operation, that guarantees that $\langle c \rangle_{i/j}$, $\langle s_1 \rangle_{i/j}$ and $\langle s_2 \rangle_{i/j}$ are indistinguishable from random.

Communication overhead: This protocol implies 2 reveal operations, and its communication overhead is thus of $2l$ give l the size of our data.

vi. Multiplication[5]

We assume that P_1, P_2 each owns an (arithmetic) share of two secrets s_1, s_2 and of a multiplication triple a,b,c, such that a and b are random and $c = a * b$ and none of the parties knows a,b or c. From there we reveal $A = s_1 - a$ and $B = s_2 - b$, and set $s_1 * s_2 = \langle c \rangle + \langle s_2 \rangle \wedge A + \langle s_1 \rangle \wedge B - A \wedge B$.

$\mathbf{\Pi_{mult}}$: or of two secret shares

Input: P_i: shares $\langle s_j \rangle_i$ of secrets to 'AND', shares of a multiplication triple $a * b = c$
Output: P_i: $\langle s_1 \wedge s_2 \rangle_i$.
Protocol:

$P_{i/j} : A \leftarrow \text{Reveal}(s_1 - a)$
$P_{i/j} : B \leftarrow \text{Reveal}(s_2 - b)$
$P_{i/j} : \langle s_1 \wedge s_2 \rangle_{i/j}$
$\leftarrow \langle c \rangle_{i/j} + \langle s_2 \rangle_{i/j} * A + \langle s_1 \rangle_{i/j} * B - A \wedge B$

Security intuition: Our result is indistinguishable from random if $\langle c \rangle_{i/j}$, $\langle s_1 \rangle_{i/j}$, $\langle s_2 \rangle_{i/j}$, A and B are. As a and b are uniformly random unknown elements, A and B are indistinguishable from random as well. Then, our protocol's security thus follows from the security of our secret shared 'Load' (Section A.i.) operation, that guarantees that $\langle c \rangle_{i/j}$, $\langle s_1 \rangle_{i/j}$ and $\langle s_2 \rangle_{i/j}$ are indistinguishable from random.

Communication overhead: This protocol implies 2 reveal operations, and its communication overhead is thus of $2l$, with l the size of the data.

vii. Or

We assume that P_1, P_2 each owns (binary) shares of two secrets s_1, s_2. The parties then

set $x \vee y = x \wedge y \oplus (x \oplus y)$. Given $i, j \in \{1, 2\}$:

$\mathbf{\Pi_{or}}$: or of two secret shares

Input: P_i: share $\langle s_j \rangle_i$ of secrets to or
Output: P_i: $\langle s_1 \vee s_2 \rangle_i$.
Protocol:

$P_{i/j} : \langle s_1 \vee s_2 \rangle_{i/j} \leftarrow (\langle s_1 \rangle_{i/j} \wedge \langle s_2 \rangle_{i/j}) \oplus (\langle s_1 \rangle_{i/j} \oplus \langle s_2 \rangle_{i/j})$

Security intuition: Our result is indistinguishable from random if $\langle s_1 \rangle_{i/j} \wedge \langle s_2 \rangle_{i/j}$ and $a \oplus b$ (for any a, b) are. Our protocol's security thus follows from the security of our secret shared 'AND' (Section A.v.) and secret shared 'XOR' (Section A.iv.) operation.

Communication overhead: We use one secret shared 'AND' (Section A.v.) and two secret shared 'XOR' (Section A.iv.) operation, so we have an overhead of 2.

viii. Add

1) Arithmetic addition

Two shares-add We assume that P_1, P_2 each owns (arithmetic) shares of two secrets s_1, s_2. As we shared the secrets additively (Section A.i.), we simply add both shares. Given $i, j \in \{1, 2\}$:

$\mathbf{\Pi_{add-secret}}$: add of two arithmetic secret shares

---

Input: P_i: share $\langle s_j \rangle_i$ of secrets to add
Output: P_i: $\langle s_1 + s_2 \rangle_i$.
Protocol:

$P_{i/j} : \langle s_1 + s_2 \rangle_{i/j} \leftarrow \langle s_1 \rangle_{i/j} + \langle s_2 \rangle_{i/j}$

Constant-add We assume that P_1, P_2 each owns a constant c and an (arithmetic) share of a secret s. As we shared the secrets additively (Section A.i.), we simply ask one of our parties to add the constant, and the other one not to change anything. Given $i, j \in \{1, 2\}$:

$\mathbf{\Pi_{add-const}}$: add of an arithmetic secret and a constant

---

Input: P_i: share $\langle s \rangle_i$ and constant c
Output: P_i: $\langle s + c \rangle_i$.
Protocol:

$P_1 : \langle s + c \rangle_1 \leftarrow \langle s_1 \rangle_1 + c$
$P_2 : \langle s + c \rangle_2 \leftarrow \langle s_1 \rangle_2$

Security intuition: No communication between parties is required, and it follows that no party learn more about s or $s \oplus c$ than what they knew before executing the protocol.

2) Binary addition

We assume that P_1, P_2 each owns (binary) shares of two secrets s_1, s_2 of l bits. To obtain shares of the secret $s_1 + s_2$ both parties add (i.e. XOR with borrow) their shares bits by bits. Assuming we access the n^{th} bit of a secret s by $s[n]$, and given $i, j \in \{1, 2\}$:

$\mathbf{\Pi_{bin-add}}$: add of two binary secret shares

Input: P_i: share $\langle s_j \rangle_i$ of secrets to add
Output: P_i: $\langle s_1 + s_2 \rangle_i$.
Protocol:

$P_i : \langle borrow \rangle_i \leftarrow 0$
$P_i : \langle s_1 + s_2 \rangle_i \leftarrow \langle [0...0] \rangle_i$ // N zeros
for $n \in 0..N$ **do**
  $P_i : \langle a\_xor\_b \rangle_i \leftarrow \langle s_1[n] \rangle_i \oplus \langle s_2[n] \rangle_i$
  $P_i : \langle a\_and\_b \rangle_i \leftarrow \langle s_1[n] \rangle_i \wedge \langle s_2[n] \rangle_i$
  $P_i : \langle (s_1 + s_2)[n] \rangle_i \leftarrow \langle a\_xor\_b \rangle_i \oplus \langle borrow \rangle_i$
  $P_i : \langle borrow \rangle_i \leftarrow \langle a\_and\_b \rangle_i \vee (\langle a\_xor\_b \rangle_i \wedge \langle borrow \rangle_i)$
end for

Security intuition: Our result is computed using exclusively our 'XOR' (Section A.iv.), 'AND' (Section A.v.) and 'OR' (Section A.vii.) operations. As such, the security of our secret shared binary addition follows from the security of those three protocols.

Communication overhead: We use two 'XOR' (Section A.iv.), two 'AND' (Section

A.v.) and one 'OR' (Section A.vii.) operation on each bit. As such, we have a communication overhead of $4l$.

ix. Subtraction

1) Arithmetic subtraction

Two shares-sub We assume that P_1, P_2 each owns (arithmetic) shares of two secrets s_1, s_2. As we shared the secrets additively (Section A.i.), we simply subtract both shares. Given $i, j \in \{1, 2\}$:

$\mathbf{\Pi_{sub-secret}}$: add of two arithmetic secret shares

---

Input: P_i: share $\langle s_j \rangle_i$ of secrets to add
Output: P_i: $\langle s_1 - s_2 \rangle_i$.
Protocol:

$P_{i/j} : \langle s_1 - s_2 \rangle_{i/j} \leftarrow \langle s_1 \rangle_{i/j} - \langle s_2 \rangle_{i/j}$

Constant-sub We assume that P_1, P_2 each owns a constant c and an (arithmetic) share of a secret s. As we shared the secrets additively (Section A.i.), we simply ask one of our parties to subtract the constant, and the other one not to change anything. Given $i, j \in \{1, 2\}$:

$\mathbf{\Pi_{sub-const}}$: add of an arithmetic secret and a constant

---

Input: P_i: share $\langle s \rangle_i$ and constant c
Output: P_i: $\langle s - c \rangle_i$.
Protocol:

$P_1 : \langle s - c \rangle_1 \leftarrow \langle s_1 \rangle_1 - c$
$P_2 : \langle s - c \rangle_2 \leftarrow \langle s_1 \rangle_2$

Security intuition: No communication between parties is required, and it follows that no party learn more about s or $s \oplus c$ than what they knew before executing the protocol.

2) Binary subtraction

We assume that P_1, P_2 each owns (binary) shares of two secrets s_1, s_2 of N bits. To obtain shares of the secret $s_1 - s_2$ both parties add (i.e. XOR with borrow) their shares bits by bits. Assuming we access the nth bit of a secret s by $s[n]$, and given $i, j \in \{1, 2\}$:

$\mathbf{\Pi_{bin-sub}}$: add of two binary secret shares

Input: P_i: share $\langle s_j \rangle_i$ of secrets to add
Output: P_i: $(\langle s_1 - s_2 \rangle_i, \langle borrow \rangle_i)$.
Protocol:

$P_i : \langle borrow \rangle_i \leftarrow 0$
$P_i : \langle s_1 - s_2 \rangle_i \leftarrow \langle [0...0] \rangle_i$ // N zeros
for $n \in 0..N$ **do**
$P_i : \langle a\_xor\_b \rangle_i \leftarrow \langle s_2[n] \rangle_i \oplus \langle s_1[n] \rangle_i$
$P_i : \langle a\_not\_b \rangle_i \leftarrow \langle s_2[n] \rangle_i \wedge \neg \langle s_1[n] \rangle_i$
$P_i : \langle (s_1 - s_2)[n] \rangle_i \leftarrow \langle a\_xor\_b \rangle_i \oplus \langle borrow \rangle_i$
$P_i : \langle borrow \rangle_i \leftarrow \langle a\_not\_b \rangle_i \vee (\langle a\_xor\_b \rangle_i \wedge \langle borrow \rangle_i)$
end for

Security intuition: our result is computed using exclusively our 'XOR' (Section A.iv.), 'AND' (Section A.v.) and 'OR' (Section A.vii.) operations. As such, the security of our secret shared binary addition follows from the security of those three protocols.

Communication overhead: We use two 'XOR' (Section A.iv.), two 'AND' (Section A.v.) and one 'OR' (Section A.vii.) operation on each bit. As such, we have a communication overhead of $4 * N$.

x. Equal[22]

We assume that P_1, P_2 each owns (binary) shares of two secrets s_1, s_2 of l bits. To obtain shares of the secret $s_1 = s_2$ both parties first compute $\neg s_1 \oplus s_2$, and then recursively 'AND' the leftmost and rightmost halves of the result until they cannot create halves anymore (i.e. the length is 1). When $s_1 = s_2$, we will have $\neg s_1 \oplus s_2 = [1....1]$ and thus 'AND'ing the halves will give a one. When we do not have equality, at least one of the bits will be zero after the XOR and it will propagate during the 'AND'ing of the halves to give a final result of zero. Assum-

ing we access the n$^t h$ bit of a secret s by $s[n]$, and given $i, j \in \{1, 2\}$:

Π_{eq} : equality test of two binary secret shares

Input: P_i: share $\langle s_j \rangle_i$ of length 2^N
Output: P_i: $\langle s_1 \stackrel{?}{=} s_2 \rangle_i$.
Protocol:

for $n \in 0..N$ **do**
  $P_i : \langle s \rangle_i \leftarrow \neg \langle s_1 \rangle_i \oplus \langle s_2 \rangle_i$
  while $length(s) > 1$ **do**
    $P_i : \langle s \rangle_i \leftarrow \langle s[...\frac{2^N}{2}] \rangle_i \wedge \langle s[\frac{2^N}{2}...] \rangle_i$
  end while
end for

Security intuition: Our result is computed using exclusively our 'XOR' (Section A.iv.) and 'AND' (Section A.v.) operations. As such, the security of our secret shared equality protocol follows by the security of those two protocols.

Communication overhead: We use one 'XOR' (Section A.iv.) and $2log_2(l)$ 'AND' (Section A.v.) operations for this protocol. As such, we have a communication overhead of $2 * log_2(l)^2$.

xi. Lower than

We assume that P_1, P_2 each owns (binary) shares of two secrets s_1, s_2 of l bits. To obtain shares of the secret $s_1 <= s_2$, each party computes the subtraction with borrow of s_1 and s_2, and the result of the comparison is, by definition, the borrow bit. Given $i, j \in \{1, 2\}$:

Π_{lt} : lower than of two binary secret shares

Input: P_i: share $\langle s_j \rangle_i$
Output: P_i: $\langle s_1 < s_2 \rangle_i$.
Protocol:

for $n \in 0..N$ **do**
  $P_i : (\_, \langle s_1 < s_2 \rangle_i) \leftarrow \langle s_1 \rangle_i - \langle s_2 \rangle_i$
end for

Security intuition: Computing this comparison is entirely dependent on our secret shared 'subtraction' (Section A.ix.) operation, and as such the security of our 'lower than' protocol follows from the security of the 'subtraction' protocol

Communication overhead We use one secret shared 'subtraction' (Section A.ix.) operation for our protocol. As such, its communication overhead is of $4 * l$.

xii. Compare-switch[1, 3.1]

We assume that P_1, P_2 each own shares of two secrets s_1, s_2 of 2^N bits, ordered in a list such that s_1 is before s_2. They want to compare $s_1 < s_2$ and inverse the position of both secrets if the comparison is successful:

Π_{leq} : lower or equal of two binary secret shares

Input: P_i: share $\langle s_1 \rangle_i, \langle s_2 \rangle_i$
Output: P_i: $\langle s'_1 \rangle_i, \langle s'_2 \rangle_i$.
Protocol:

for $n \in 0..N$ **do**
  $P_i : \langle b \rangle_i \leftarrow \langle s_1 \rangle_i < \langle s_2 \rangle_i$
  $P_i : \langle s'_{1/2} \rangle_i \leftarrow \langle b \rangle_i \langle s_{1/2} \rangle_i + \langle b \rangle_i \langle s_{2/1} \rangle_i$
end for

Security intuition: Computing this comparison is entirely dependent on our secret shared 'lower than' (Section A.xi.), secret shared 'AND' or 'Mult' in case of Arithmetic shares)(Section A.v., A.vi.) and secret shared 'ADD' (Section A.viii.) operations, and as such the security of our 'compare-switch' protocol follows from the security of the aforementioned protocols.

Communication overhead We use one secret shared 'lower than' (Section A.xi.), two secret shared 'AND' or 'Mult' in case of Arithmetic shares)(Section A.v., A.vi.) and one secret shared 'ADD' (Section A.viii.) operations for our "compare-switch". It follows that its communication overhead is of $8l$, with l our bit size.

B. Odd-even Merge: Ideal Functionality by Batcher[4]

An "s by t" merging network can be built by presenting the odd-indexed numbers of the two input lists to one small merging network (the odd merge), presenting the even-indexed numbers to another small merging network (the even merge) and then comparing the outputs of these small merges with a row of comparison elements. The lowest output of the odd merge is left alone and becomes the lowest number of the final list. The i^{th} output of the even merge is compared with the $i+1^{th}$ output of the odd merge to form the $2i^{th}$ and $2i+I^{th}$ numbers of the final list for all applicable i's. This may or may not exhaust all the outputs of the odd and even merges; if an output remains in the odd or even merge it is left alone and becomes the highest number in the final list.

C. Intuition Composition theorem

We give the intuition using the security proof of Π_{PPCS} as example:

Let's define (for all $z \in \{ABB, USG, MCS\}$) $\mathcal{O}_z$ the oracle to the functionality $\mathcal{F}_z$, with $\mathcal{B}_z$, the optimal adversary to Π_z and β_z its probability to break the protocol's security. We now define S_{PPCS} the simulator for Π_{PPCS} with random tape r, that simulates the view USG using oracle call to $\mathcal{O}_{USG}$, ABB using oracle calls to $\mathcal{O}_{ABB}$ and MCS using oracle calls to $\mathcal{O}_{MCS}$. Noting $\mathcal{A}$ the optimal adversary for $\mathcal{S}_{PPCS}$, it forwards each slice of its view (USG, ABB and MCS) to its respective optimal adversary. Given the answers 'b' (i.e. 0 for 'Real', 1 for 'Simulated') received from every $\mathcal{B}_z$ and their respective probability β_z to win their game, $\mathcal{A}$ will agree with a given adversary $\mathcal{B}_X$ with the following probabilities (for $X, Y, Z \in \{USG, ABB, MCS\}$, $b \in 0, 1$ and $\bar{b} = 1 - b$):
1. As every adversaries $\mathcal{B}_z$ are independent from one another, and $\mathcal{B}_z(b)$ meaning that $\mathcal{B}_z$ returns b:

$$\begin{aligned} P(\mathcal{B}_X(b)) =& P(\mathcal{B}_X(b) \wedge b) + P(\mathcal{B}_X(b) \wedge \bar{b}) = P(\mathcal{B}_X(b)|b)P(b) + P(\mathcal{B}_X(b)|\bar{b})P(\bar{b}) = 1/2 \\ P(\mathcal{B}_X(\bar{b})) =& 1/2 \\ P(b|\mathcal{B}_X(b)) =& P(b \wedge \mathcal{B}_X(b))/P(\mathcal{B}_X(b)) = P(\mathcal{B}_X(b)|b) * P(b)/P(\mathcal{B}_X(b)) = \beta/2P(\mathcal{B}_X(b)) = \beta \end{aligned} \tag{5}$$

2. It follows that:

$$\begin{aligned} P(b|\mathcal{B}_X(b) \wedge \mathcal{B}_Y(b) \wedge \mathcal{B}_Z(\bar{b})) &= P(b \wedge \mathcal{B}_X(b) \wedge \mathcal{B}_Y(b) \wedge \mathcal{B}_Z(\bar{b}))/P(\mathcal{B}_X(b) \wedge \mathcal{B}_Y(b) \wedge \mathcal{B}_Z(\bar{b})) \\ &= P(b \wedge \mathcal{B}_X(b) \wedge \mathcal{B}_Y(b) \wedge \mathcal{B}_Z(\bar{b})) * 8 \\ &= 8 * P(\mathcal{B}_X(b) \wedge \mathcal{B}_Y(b) \wedge \mathcal{B}_Z(\bar{b})|b) * P(b) \\ &= 4 * \beta_X * \beta_Y * (1 - \beta_Z) \end{aligned} \tag{6}$$

Note that if all $\beta \equiv 1/2$ (i.e. Adv. $\mathcal{B}_X \equiv 0$), $P(b|\mathcal{B}_X(b) \wedge \mathcal{B}_Y(b) \wedge \mathcal{B}_Z(\bar{b})) = 1/2$ too, and it is clear that it would be the same for $P(b|\mathcal{B}_X(b) \wedge \mathcal{B}_Y(\bar{b}) \wedge \mathcal{B}_Z(\bar{b}))$ and $P(b|\mathcal{B}_X(\bar{b}) \wedge \mathcal{B}_Y(\bar{b}) \wedge \mathcal{B}_Z(\bar{b}))$. In definitive, assuming all our protocols are secure, the main one would be secure too.

$$\text{Adv}^{S_{PPCS}}(\mathcal{A}) = |Pr[\text{Win}(\mathcal{A})] - \frac{1}{2}| \tag{7}$$